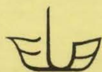
OCTAVIAN BÂSCĂ
LEON LIVOVSCHI

ALGORITMI EURISTICI



Editura Universității din București

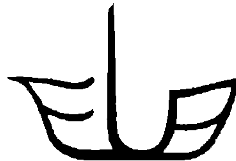
https://biblioteca-digitala.ro / https://unibuc.ro

https://biblioteca-digitala.ro / https://unibuc.ro

OCTAVIAN BÂSCĂ LEON LIVOVSCHI

bd 267570

ALGORITMI EURISTICI



EDITURA UNIVERSITĂȚII DIN BUCUREȘTI
2003

https://biblioteca-digitala.ro / https://unibuc.ro

III 476843

Referenți științifici: Prof. dr. Ion VĂDUVA
Prof. dr. Ioan TOMESCU

12/8/03

© Editura Universității din București
Şos. Panduri, 90–92, Bucureşti – 76235; Telefon/Fax: 410.23.84
E-mail: editura@unibuc.ro
Internet: www.editura.unibuc.ro

Descrierea CIP a Bibliotecii Naționale a României
BÂSCĂ, OCTAVIAN
Algoritmi euristici / Bâscă Octavian, Leon Livovschi – București: Editura Universității din București, 2003
368 p.
Bibliografie
ISBN 973-575-785-0
I. Livovschi, Leon
51

https://biblioteca-digitala.ro / https://unibuc.ro

INTRODUCERE

Aşa cum se menţionează în *Mic Dicţionar Enciclopedic*, termenul *euristic* provine de la cuvântul grecesc ***heurisko*** ceea ce în traducere înseamnă ***a descoperi***. Tot aici sunt date sensurile lui *euristic* cu funcţiile sintactice de adjectiv, substantiv şi în combinaţie cu termenul *algoritm*. Ca adjectiv, *euristic* priveşte procedeele metodologice, fiind ceva care serveşte la obţinerea unor cunoştinţe noi sau care se aplică metodelor descoperirii şi ale invenţiei. Sensul de substantiv al lui *euristică* desemnează o ramură de ştiinţă care are ca obiect studierea activităţii creatoare, a metodologiei şi tehnicii inovaţiei intelectuale. Tot în acelaşi articol mai este menţionat şi termenul de *algoritm euristic* definit ca un algoritm verificat pe cazuri particulare dar nedemostrat ca valabil pentru cazul general sau ca algoritm care dă o soluţie aproximativă unei probleme.

Din cele de mai sus rezultă că noţiunile de euristică şi algoritm euristic sunt noţiuni foarte des utilizate. Dar, cu toată răspândirea metodelor euristice, au fost elaborate până acum destul de puţine lucrări în care să fie studiate şi prezentate sistematic aceste metode.

Prin extindere, aproape orice teorie matematică şi aplicaţiile ei sunt în ultimă instanţă procedee euristice. Pentru rezolvarea unei probleme practice cu ajutorul matematicii se parcurg în general următoarele etape:

1. Construirea unui model matematic.
2. Rezolvarea teoretică a problemei.
3. Interpretarea rezultatelor în cazul real.

În fiecare dintre aceste etape sunt efectuate aproximaţii datorate pe de o parte diferitelor abstractizări, pe de altă parte imperfecţiunii măsurătorilor şi eventual subiectivităţii. Oricât de elaborat ar fi modelul matematic ales, se pot găsi elemente ale lumii reale care nu au fost luate în consideraţie în model şi care eventual ar putea să determine calitatea soluţiei găsite. De foarte multe ori, deşi se operează logic cu mulţimi infinite, implementarea calculelor se face prin mulţimi finite. O influenţă mare o are şi modul de abordare a problemelor. Dăm în acest sens un exemplu.

https://biblioteca-digitala.ro / https://unibuc.ro

S-a făcut un test privind aptitudinile casnice ale bărbaţilor. Una dintre întrebările testului era cea privind indicarea modului de preparare a unui ceai. Majoritatea lor a indicat următorul algoritm de preparare a unui ceai:

Algoritmul 0.1. (***Preparare ceai***)

1. [Apa] Se pune apă în ibric.

2. [Fierbere] Se fierbe apa din ibric.

3. [Finalizare] Se adaugă ceai, zahăr, lămâie şi eventual rom, apoi STOP. ■

Deosebiri esenţiale au apărut în răspunsul la următoarea întrebare în care se punea problema preparării ceaiului în cazul în care în ibric există deja apă. Astfel, o parte dintre bărbaţi a rezolvat problema prin adăugarea la începutul algoritmului precedent a pasului suplimentar:

0. [Verificare] Dacă în ibric este apă, atunci se merge la pasul 2.

Dar au fost şi unii dintre bărbaţi care au găsit o soluţie în care se adăuga la început:

0. [Verificare] Dacă în ibric este apă, atunci se varsă apa din ibric.

Ambele soluţii sunt corecte din punct de vedere al rezultatului final. Dar ambele au atât calităţi cât şi deficienţe. A doua soluţie este în spiritul gândirii matematice de a reduce rezolvarea unei probleme la rezolvarea altei probleme pentru care se cunoaşte soluţia. Se poate lucra astfel *modular*, fiind suficient de a şti că un modul are o mulţime de date de intrare şi o mulţime de date de ieşire fără a fi de interes procedeul cu care se găseşte soluţia. Prima soluţie ţine seama de structura algoritmului de rezolvare, speculând forma soluţiei. Acest mod de gândire poate să ducă la soluţii mai eficiente. În cazul de faţă, nu numai că se economisesc unele operaţii, vărsarea apei şi punerea altei ape în ibric, dar poate să ducă la soluţie pentru unele situaţii care nu se pot rezolva prin a doua soluţie, cum ar fi oprirea apei curente pentru o perioadă de timp din cauza unei defecţiuni la reţeaua de distribuţie.

Scopul acestei cărţi este identificarea şi prezentarea diferitelor clase de algoritmi euristici şi metodele de aplicare ale acestora. Trebuie să recunoaştem că este greu de inclus într-un singur volum toate clasele de algoritmi euristici, numărul lor fiind foarte mare, acest subiect fiind abordat într-o mulţime de lucrări apărute în reviste de specialitate.

https://biblioteca-digitala.ro / https://unibuc.ro

După cunoştinţa autorilor, aceasta este prima monografie din ţara noastră care abordează sistematic acest domeniu cu o mare aplicabilitate. Aşa cum vom menţiona în continuare, diferite tipuri de algoritmi euristici sunt studiate în lucrări din domenii foarte diverse cum ar fi cercetările operaţionale, analiza numerică, teoria sistemelor, structuri de date, matematici financiare, fizică, chimie, biologie, arhitectură, teoria simulării, statistică, medicină şi multe altele.

Structura lucrării este o reflectare a modului în care intervin metodele euristice în rezolvarea diferitelor probleme practice. Bazaţi pe o bogată experienţă didactică şi din dorinţa de a face o carte cu adevărat utilă, am selecţionat pentru a prezenta metodele cu o cât mai mare aplicabilitate. În mod inevitabil, unele din aceste metode au primit o tratare mai amplă, fiind ilustrate cu exemple, iar altele au fost trecute numai în revistă. Pentru diferitele proprietăţi enunţate am inclus şi demonstraţii numai în cazurile cînd acestea dau informaţii suplimentare sau pentru o mai bună înţelegere a noţiunilor.

Nu am inclus în carte metodele tratate în domeniul analizei numerice deşi aceste metode sunt în cea mai mare parte algoritmi euristici. Pentru aceste metode există deja o literatură bogată. Pot fi consultate în acest domeniu cărţi editate în ţara noastră, elaborate de autori români sau traduceri, ([BUC73], [CRA82], [DOR76], [MAR87] etc.) şi în străinătate, de referinţă fiind lucrarea [DEM81].

Am insistat asupra metodelor combinatoriale, a celor care utilizează teoria grafurilor şi în special asupra metodelor de rezolvare a problemelor de planificarea activităţilor. Aceste metode au o foarte mare aplicabilitate în majoritatea domeniilor. Calitatea soluţiilor determinate şi aproximarea cât mai bună a soluţiilor optimale pot să fie hotărâtoare pentru obţinerea unor beneficii materiale şi morale în domeniul bancar, în transporturi, în comunicaţii, în servicii, în informatică, în electrotehnică, în electronică, în medicină şi în multe alte domenii.

Cartea conţine un număr de şapte capitole.

În Capitolul 1 se prezintă caracteristicile claselor de algoritmi euristici. Sunt definite diferitele tipuri de metode aproximative. Se dau metode de analiză a algoritmilor euristici prin care se determină calităţile lor şi metode de sinteză pentru aceşti algoritmi prin care se pot ei elabora. Scopul acestui prim capitol este prezentarea unor instrumente de lucru pentru abordarea problemelor soluţionate prin intermediul algoritmilor euristici. În ultima parte a acestui prim capitol sunt prezentaţi algoritmii genetici şi evolutivi.

https://biblioteca-digitala.ro / https://unibuc.ro

În Capitolul 2 se consideră câteva probleme din domeniul combinatoricii. Sunt studiate probleme cum sunt: împerecherea minimală, problema rucsacului, probleme de aritmetică combinatorie, selectarea şi ordonarea dispozitivelor de orientare, problema superşirului minimal, acoperirea unui set de mulţimi şi minimizarea sumei maximale pe liniile unei matrici. De cele mai multe ori problemele de combinatorică presupun enumerarea unor elemente ale unor mulţimi şi alegerea unor submulţimi ale acestor elemente drept soluţii posibile. Euristicile aplicate în acest caz restrâng mulţimile iniţiale la mulţimi cu mai puţine elemente.

În Capitolul 3 sunt abordate probleme din teoria grafurilor. Dacă am tratat pe larg probleme de drumuri minime care sunt invocate în foarte multe aplicaţii, nu am tratat decât sumar probleme de fluxuri. Acestea din urmă sunt tratate în mai multe lucrări ce au apărut şi în limba română. Sunt date metode de rezolvare pentru problema arborelui lui Steiner, pentru localizarea centrelor unui graf, pentru descompunerea unor grafuri, pentru plasarea facilităţilor într-o reţea, pentru drumuri hamiltoniene şi pentru generarea unor separatori.

În Capitolul 4 se dau mai multe metode de rezolvare euristică a unei probleme particulare din teoria grafurilor şi anume problema comisvoiajorului. Această problemă a permis construirea unor clase importante de euristici ce au putut să fie aplicate la soluţionarea unui mare număr de probleme. Sunt tratate cazuri particulare cu restricţii sau cu folosirea mai multor persoane, probleme de localizare a staţiilor de servire şi diferite aplicaţii.

În Capitolul 5 este indicată o mare varietate de metode pentru rezolvarea problemei planificării activităţilor. Această problemă este una din cele mai frecvent întâlnite în practică. În plus, multe din metodele indicate în acest capitol pot fi adaptate pentru a obţine soluţii şi pentru alte probleme din alte domenii. Am încercat să prezentăm metode cât mai variate şi care să satisfacă diferite condiţii suplimentare impuse planificărilor. Sunt indicate metode de soluţionare polinomiale (bazate pe drumuri critice, pe circuite critice şi metode clasice), metode nepolinomiale (bazate pe programare dinamică şi metode arborescente), cazuri speciale de planificări (procesoare distincte, cu timpi de adaptare, cu profit maxim, minimizarea sumei întârzierilor, cu grafuri liniare, programări stohastice, planificări preemtive şi cu divizarea procesorilor, minimizarea timpului mediu etc.). Sunt date o serie de aplicaţii ale acestor metode.

https://biblioteca-digitala.ro / https://unibuc.ro

În Capitolul 6 se dau metode de soluţionare a problemelor de programare liniară întreagă atât sub forma generală cât şi cu variabile 0-1 şi diferite aplicaţii care duc la rezolvarea unor probleme de acest tip cum ar fi problema depozitării şi procesarea pe mai multe maşini paralele independente.

În Capitolul 7 au fost incluse alte probleme ce nu sunt de tipul celor prezentate în capitolele anterioare şi care se rezolvă tot prin metode euristice. Au fost prezentate metode pentru ajustarea costurilor în cazul incertitudinii cererilor, localizarea deservirii, reaprovizionarea comună, plasarea optimală a băncilor de conturi, descompunerea matricilor de trafic în comunicaţia prin sateliţi, problema monezilor, organizarea memoriilor liniare, arbori binari optimi pentru căutare, problema şahului şi alte probleme.

Modul de prezentare a algoritmilor este cel utilizat în multe lucrări de informatică. Execuţia unui algoritm se realizează în felul următor. Se începe cu efectuarea operaţiilor indicate în primul pas. Terminarea unui algoritm are loc dacă se ajunge la STOP sau se termină de efectuat operaţiile din ultimul pas al algoritmului şi nu se indică efectuarea operaţiilor de la un alt pas. Dacă sunt efectuate operaţiile unui pas şi nu este prevăzută trecerea la un alt pas, atunci se trece la efectuarea operaţiilor prevăzute la pasul următor, în afara cazului, menţionat mai sus, în care s-au terminat de executat operaţiile ultimului pas precum şi în cazul în care în unii paşi se indică executarea repetată a unor paşi pentru diferite valori ale unor parametrii. Fiecare pas conţine la început, incluse între paranteze drepte, un cuvânt sau o expresie care sintetizează operaţiile prevăzute în acel pas.

Pentru marcarea terminării textului corespunzător unor teoreme, leme, definiţii, algoritmi, exemple şi altele am folosit semnul ■ . Începerea textului ce defineşte aceste noţiuni se face prin **litere îngroşate** şi sunt, în general, numerotate. Numerotarea se face prin numărul capitolului din care face parte şi numărul curent de tipul respectiv. Au fost utilizate numere şi pentru unele expresii, figuri, formule şi tabele pentru a fi uşor de referit.

Noţiunile definite în text, corespondentul din diferite limbi străine a unor cuvinte sau expresii şi majoritatea notaţiilor sunt reprezentate prin *litere cursive* pentru a putea să fie mai uşor reperate.

Bibliografia plasată la sfârşitul cărţii, cu toate că este bogată, este departe de a fi completă. Am inclus în special acele lucrări care sunt mai apropiate de aspectele tratate şi care ajută cititorul să aprofundeze problematica algoritmilor euristici.

https://biblioteca-digitala.ro / https://unibuc.ro

Prin modul de prezentare, prin enunţarea unor rezultate foarte puternice, explicarea folosirii acestora şi posibilitatea implementării pe calculator, lucrarea se adresează studenţilor din facultăţile de informatică, matematică, fizică, chimie şi biologie, ai institutelor tehnice şi economice, agronomice şi de medicină, cercetătorilor din domeniile amintite, precum şi personalului din compartimentele de conducere şi control.

Sperăm ca această lucrare să se dovedească utilă şi să se bucure de popularitate între specialişti şi nespecialişti care utilizează metodele informatice pentru rezolvarea problemelor practice şi pentru diferite alte aplicaţii posibile. Plecând de la deviza repetată foarte des de profesorul Grigore C. Moisil:

Cea mai scumpă ştiinţă este mult mai ieftină decât cea mai ieftină neştiinţă!

dorim şi noi ca metodele prezentate aici să fie cunoscute şi aplicate cât mai mult.

Mulţumim pe această cale tuturor celor care au contribuit la finalizarea acestei dorinţe a noastre de a închega o teorie a algoritmilor euristici plecând de la lucrări disparate şi bazându-ne pe experienţa noastră la catedră şi în utilizarea tehnicii de calcul. Ne referim în primul rând la colegii de muncă şi prietenii noştri care ne-au încurajat mereu cu sfaturi şi informaţii utile. Apoi ne referim la colectivul cu care am colaborat la Editura Academiei. Am găsit aici multă înţelegere şi solicitudine din partea doamnei SILVIA CÂNDEA, a domnului PETRE MOCANU, a domnului director IOAN GANEA şi a multor altora pentru pregătirea şi tipărirea acestei cărţi. Nu în ultimul rând vrem să mulţumim soţiilor noastre şi familiilor noastre pentru înţelegerea timpului sacrificat lor ca să putem realiza această lucrare. Ne-am bucura nespus să ştim că aceste sacrificii nu au fost zadarnice şi să putem să înregistrăm reacţii, pozitive s-au critice, care să ne dovedească interesul pentru tratarea subiectului algoritmi euristici. În funcţie de acestea vom şti dacă trebuie să continuăm sau nu să depunem eforturi noi pentru desţelenirea altor metode de tipul celor prezentate aici.

Bucureşti 10.12.1997 AUTORII

https://biblioteca-digitala.ro / https://unibuc.ro

CUPRINS

INTRODUCERE 3

CAPITOLUL 1. ANALIZA ŞI SINTEZA ALGORITMILOR EURISTICI 17

1.1. Algoritmi euristici şi aproximativi 17
1.1.1. Definiţii de bază 17
1.1.2. Algoritmi aproximativi 20
1.1.2.1. Algoritmi cu aproximare absolută 20
1.1.2.1.1. Problema memorării fişierelor pe două benzi 20
1.1.2.2. Algoritmi $f(n)$-aproximativi 21
1.1.2.2.1. Problema comisvoiajorului 21
1.1.2.3. Algoritmi ε-aproximativi 21
1.1.2.3.1. Planificarea activităţilor independente 21
1.1.2.3.2. Problema împachetării 22
1.1.2.3.3. Problema rucsacului 24
1.1.2.4. Scheme aproximative în timp polinomial 27
1.1.2.4.1. Planificarea activităţilor independente 27
1.1.2.4.2. Problema comisvoiajorului 28
1.1.2.5. Scheme de aproximare complet polinomiale 29
1.1.2.5.1. Rotunjirea 30
1.1.2.5.2. Partiţionarea pe intervale 32
1.1.2.5.3. Separarea 33
1.1.2.6. Algoritmi buni în sens probabilistic 35
1.1.3. Utilizarea teoriei NP-completitudinii pentru identificarea de algoritmi aproximativi 37
1.2. Analiza algoritmilor 38
1.2.1. Măsuri de comportare în cel mai rău caz a unui algoritm 38
1.2.2. Algoritmul A^* 42
1.2.2.1. Definiţii şi notaţii suplimentare 44

https://biblioteca-digitala.ro / https://unibuc.ro

1.2.2.2. Căutarea unei soluții optimale cu algoritmul A^* 45
1.2.2.2.1. Proprietățile funcției f^* 45
1.2.2.2.2. Terminare și completitudine 46
1.2.2.2.3. Admisibilitate - garanția pentru o soluție optimală 47
1.2.2.2.4. Comportarea puterii de selectare a euristicilor 48
1.2.2.2.5. Euristici monotone (consistente) 50
1.2.2.3. Relaxarea condițiilor de optimalitate 53
1.2.2.3.1. Ajustarea ponderilor lui g și h 53
1.2.2.3.2. Două versiuni ε-admisibile mai rapide ale lui A^* 54
1.2.2.3.2.1. Ponderarea dinamică 54
1.2.2.3.2.2. A^*_ε - algoritm ce utilizează estimări ale efortului căutării 55
1.2.2.3.2.3. Compararea celor doi algoritmi 55
1.2.3. Euristicile ca informație dată de modele simplificate 58
1.2.3.1. Utilizarea modelelor relaxate 58
1.2.3.1.1. Proveniența euristicilor 58
1.2.3.1.2. Consistența euristicilor bazate pe relaxare 60
1.2.3.2. Generarea mecanică a euristicilor admisibile 61
1.2.3.2.1. Relaxarea sistematică 61
1.2.3.2.2. Observații și concluzii 63
1.2.4. Euristici bazate pe probabilitate 64
1.2.4.1. Euristici bazate pe cel mai plauzibil rezultat 64
1.2.4.2. Euristici bazate pe eșantionare 65
1.2.5. Modele abstracte pentru analiza cantitativă a performanței 66
1.2.5.1. Analiza matematică a performanței 66
1.2.5.2. Exemplul 1: Găsirea unui cel mai scurt drum într-o latice regulată 66
1.2.5.3. Exemplul 2: Găsirea unui cel mai scurt drum într-o rețea de orașe distribuite la întâmplare 71
1.2.5.4. Exemplul 3: Căutarea unui drum optim într-un arbore cu costuri aleatoare 74
1.2.5.4.1. Rezultate 76
1.2.6. Complexitatea și precizia euristicilor admisibile 77
1.2.6.1. Euristicile interpretate drept surse de informație perturbate 77
1.2.6.1.1. Modelele simplificate - surse de semnale perturbate 77
1.2.6.1.2. Un model probabilistic pentru analiza performanței unei euri' ci 78
1.2.6.2. Dominanța stohastică pentru euristici admisibile aleatoare 81

https://biblioteca-digitala.ro / https://unibuc.ro

1.3. Sinteza algoritmilor euristici 83
1.3.1. Clasificarea metodelor euristice 83
1.3.1.1. Morfologia metodelor de identificare a unei prime soluţii 84
1.3.1.2. Morfologia metodelor iterative 85
1.3.2. Metode exacte pentru sinteza algoritmilor 85
1.3.2.1. Metoda Greedy 85
1.3.2.2. Metode arborescente 86
1.3.2.3. Metoda backtracking 88
1.3.2.4. Metoda branch and bound 89
1.3.2.5. Metoda divide et impera 90
1.3.2.6. Metoda programării dinamice 90
1.3.3. Metode generale de construire a unor euristici 92
1.3.3.1. Euristici folosind metoda Greedy 92
1.3.3.2. Euristici folosind metoda backtracking 93
1.3.3.3. Euristici folosind metoda branch and bound 93
1.3.3.4. Euristici folosind metoda divide et impera 95
1.3.3.5. Euristici folosind metoda programării dinamice 95
1.4. Algoritmi genetici şi evolutivi 96
1.4.1. Introducere 96
1.4.2. Algoritm genetic pentru optimizarea unei funcţii 97
1.4.2.1. Reprezentarea 98
1.4.2.2. Populaţia iniţială 98
1.4.2.3. Funcţia de evaluare 98
1.4.2.4. Operatori genetici 99
1.4.2.5. Parametri 99
1.4.2.6. Rezultate experimentale 100
1.4.2.7. Procesul de selectare 100
1.4.3. Algoritm evolutiv pentru rezolvarea problemei liniar-pătratice 101
1.4.3.1. Operatori specializaţi 102
1.4.4. Forma generală a algoritmilor genetici şi evolutivi 105
1.4.5. Observaţii şi comentarii 106

CAPITOLUL 2. PROBLEME COMBINATORIALE 107

2.1. Împerechere minimală 107
2.2. Problema rucsacului 110

https://biblioteca-digitala.ro / https://unibuc.ro

2.2.1. Algoritm euristic eficient pentru problema rucsacului 110
2.2.2. Algoritm aproximativ polinomial pentru problema rucsacului 111
2.2.3. Teste de realizabilitate 111
2.2.4. Cazul densităţilor ordonate ale profitului 113
2.2.5. Algoritmi genetici pentru problema rucsacului 115
2.3. Algoritmi aproximativi pentru probleme de aritmetică combinatorie 118
2.3.1. Modul de prezentare al calculelor 119
2.3.2. Metoda condensării 120
2.4. Selectarea şi ordonarea dispozitivelor de orientare 121
2.4.1. Ordonarea optimală a dispozitivelor în cazul unei singure piese 123
2.4.2. Selectarea şi ordonarea dispozitivelor în cazul unei piese 125
2.4.3. Selectarea şi ordonarea dispozitivelor pentru *m* tipuri de piese 127
2.4.4. Selectarea şi ordonarea dispozitivelor în cazul a *m* tipuri de piese cu *l* orientări 127
2.5. Algoritmi aproximativi pentru problema superşirului minimal 127
2.5.1. Algoritmul împachetării 128
2.5.2. Algoritmul împachetării direcţionate 129
2.5.3. Algoritmul de tip Greedy 130
2.6. Problema acoperirii unui set de mulţimi 131
2.6.1. Prezentarea algoritmului 132
2.7. Minimizarea sumei maximale pe liniile unei matrice 134
2.7.1. Algoritmul *RDI* pentru cazul general 135
2.7.2. Algoritmul *RLPT* pentru cazul $m \times 3$ 137

CAPITOLUL 3. PROBLEME ÎN TEORIA GRAFURILOR 139

3.1. O euristică pentru problema arborelui lui Steiner 139
3.2. Algoritm euristic pentru arbori Steiner rectiliniari 142
3.3. Algoritm euristic pentru identificarea drumurilor minime 146
3.3.1. Metoda corectării etichetelor 147
3.3.2. Algoritmul examinării topologice 147
3.4. Metodă euristică pentru identificarea clicilor de cardinalitate dată şi de cost minimal 150
3.5. Localizarea mai multor centre într-un graf 153
3.6. Descompunerea unui graf 158
3.6.1. Identificarea clicilor (θ, α) maximale 158

https://biblioteca-digitala.ro / https://unibuc.ro

3.6.2. Procedura contractării grafului 160
3.7. Aşezarea facilităţilor într-o reţea planară 162
3.7.1. Euristica triunghiului 162
3.7.2. Euristica expandării roţilor 163
3.7.3. Euristica Greedy 164
3.7.4. Îmbunătăţirea soluţiilor finale 164
3.7.5. Complexitatea euristicilor propuse 165
3.8. Drum hamiltonian într-un graf planar maximal 165
3.9. Algoritm euristic pentru generarea de separatori mici în grafuri arbitrare 168

CAPITOLUL 4. PROBLEMA COMISVOIAJORULUI 171

4.1. Metode euristice pentru problema comisvoiajorului 171
4.1.1. Metoda extinderii bilaterale 171
4.1.2. Metoda reducerii 172
4.1.3. Metoda proximităţii 174
4.1.4. Metode bazate pe pierderi 175
4.1.4.1. Pierdere simplă 176
4.1.4.2. Pierdere compusă 176
4.1.4.3. Folosirea distanţelor ajustate 182
4.1.4.4. Euristici mai rapide 182
4.1.5. Metoda ciclurilor crescătoare 183
4.1.6. Rectificarea turului 183
4.2. Problema restrictivă a comisvoiajorului 188
4.3. Problema comisvoiajorului cu *k* persoane 191
4.4. Localizarea staţiei de deservire în problema comisvoiajorului 193
4.5. Margini pentru problema transportului cu vehicule de capacitate limitată 197
4.5.1. Margini inferioare şi superioare 197
4.5.2. Margini superioare pentru problema comisvoiajorului 198
4.6. Problema orientării 201
4.7. O aplicaţie a problemei comisvoiajorului 203

CAPITOLUL 5. PLANIFICAREA ACTIVITĂŢILOR 207

5.1. Modelarea problemelor de planificarea activităţilor 207
5.1.1. Notaţii utilizate 208

https://biblioteca-digitala.ro / https://unibuc.ro

5.1.2. Modul de execuție 208
5.1.3. Resurse 209
5.1.4. Funcții obiectiv 210
5.1.5. Reprezentarea soluțiilor 211
5.1.6. Modelarea problemei principale de programare 212
5.1.7. Modelarea cu rețele Petri 213
5.2. Metode polinomiale de rezolvare a problemei planificării activităților 215
5.2.1. Algoritmi bazați pe metoda drumului critic 215
5.2.1.1. Mulțimi de potențiale pe un graf conjunctiv 215
5.2.1.2. Metoda potențiale-activități 216
5.2.1.3. Metode seriale de planificare folosind drumuri critice 218
5.2.1.4. Euristica Schrage pentru un singur procesor 220
5.2.1.5. Probleme cu resurse consumabile 222
5.2.1.6. Probleme cu termene de predare 224
5.2.1.7. Metoda PERT 225
5.2.2. Metoda circuitelor critice 226
5.2.2.1 Modelarea problemei centrale repetitive 227
5.2.3. Metode clasice 230
5.2.3.1 Metode de schimb 230
5.2.3.1.1. Programarea pe un procesor 230
5.2.3.1.1.1. Proprietăți ale soluțiilor optime pentru activități dependente 230
5.2.3.1.1.2. Dependențe de tip antiarborescență 231
5.2.3.1.1.3. Euristică Greedy pentru activități cu termene de predare 232
5.2.3.1.2. Probleme de atelier cu două procesoare 233
5.2.3.1.3. Euristici de planificare pe două mașini cu timpi de accesibilitate 234
5.2.3.1.4. Algoritm euristic pentru procesare pe mai multe procesoare 237
5.2.3.1.5. Euristici bazate pe indicele de înclinare 239
5.2.3.1.6. Planificarea pe procesoare cu viteze diferite 241
5.2.3.2. Metode de planificare pe bază de listă 244
5.2.3.2.1. Margini pentru anomaliile din multiprocesare 246
5.2.3.2.2. Activități cu timpi egali și precedențe tip antiarborescențe 249
5.2.3.2.3. Euristici bazate pe nivelurile nodurilor 250
5.2.3.2.4. Programare optimală pentru doi procesori și timpi egali 250
5.3. Metode nepolinomiale de programarea activităților 252
5.3.1. Algoritmi bazați pe programarea dinamică 252

https://biblioteca-digitala.ro / https://unibuc.ro

5.3.1.1. Programarea activităţilor independente 253
5.3.1.2. Programarea activităţilor dependente 256
5.3.2. Metode arborescente 256
5.3.2.1. Minimizarea sumei întârzierilor 256
5.3.2.2. Minimizare durată totală pentru activităţi cu timp de latenţă 258
5.4. Alţi algoritmi aproximativi pentru programarea activităţilor 261
5.4.1. Planificarea activităţilor pe procesoare distincte 261
5.4.2 Altă euristică pentru planificarea pe procesoare diferite 264
5.4.3 Minimizarea duratei totale cu timpi de adaptare 266
5.4.4. Planificări cu profit maxim 268
5.4.5. Minimizarea sumei întârzierilor 269
5.4.5.1. Căutarea locală şi vecinătăţi 270
5.4.5.2. Valorile maxime ale erorilor pentru *K-INT* şi *ADJ* 272
5.4.6. Planificarea activităţilor folosind grafuri liniare 273
5.4.7. Analiza euristicilor pentru programări stohastice 278
5.4.8. Planificări preemtive şi cu divizarea procesorilor 281
5.4.9. Planificări cu minimizarea timpului mediu 286
5.5. Aplicaţii 290
5.5.1. Algoritm efectiv de asamblare a lucrărilor 290
5.5.1.1. Euristici pentru problema *SALB-I* 290
5.5.1.2. Euristici pentru problema *SALB-II* 292
5.5.2. Performanţa unor euristici pentru linii de asamblare 293
5.5.3. Desvoltarea capacităţii cu mai multe tipuri de facilităţi 295

CAPITOLUL 6. PROGRAMARE LINIARĂ ÎNTREAGĂ 299

6.1. Algoritmi euristici pentru programarea liniară întreagă 299
6.1.1. Metodele fazei 1 300
6.1.2. Metodele fazei 2 301
6.1.3. Metodele fazei 3 303
6.1.4. Exemplu de programare liniară întreagă 303
6.2. Algoritm aproximativ pentru programarea 0-1 305

https://biblioteca-digitala.ro / https://unibuc.ro

6.3. Programarea liniară a comenzilor cu variabile 0-1 309
6.4. Problema depozitării 314
6.5. Euristică liniară de procesare pe maşini paralele independente 317

CAPITOLUL 7. ALTE APLICAŢII 321

7.1. Euristici de ajustare a costurilor în cazul incertitudinii cererilor 321
7.2. Euristici pentru problema localizării deservirii 323
7.3. Problema reaprovizionării comune 326
7.4. Plasarea optimală a băncilor de conturi 330
7.5. Descompunerea euristică a matricilor de trafic în comunicarea sateliţilor 333
7.6. Problema monedelor 337
7.7. Organizarea memoriilor liniare 342
7.8. Arbori binari optimi pentru căutare 351
7.9. Euristici în jocul de şah 354

BIBLIOGRAFIE 357

https://biblioteca-digitala.ro / https://unibuc.ro

1 ANALIZA ŞI SINTEZA ALGORITMILOR EURISTICI

Acest capitol cuprinde definirea euristicilor şi algoritmilor aproximativi, clasificări şi principalele metode de analizare şi elaborare ale acestor algoritmi. Pentru majoritatea metodelor sunt date exemple. Unele probleme sunt reluate şi dezvoltate noi metode de soluţionare ale lor în capitolele următoare.

1.1. Algoritmi euristici şi aproximativi

1.1.1. Definiţii de bază

Un algoritm se numeşte *algoritm aproximativ* dacă generează, pentru o problemă P dată, o *soluţie corectă* având o valoare apropiată, într-un sens ce trebuie precizat, de valoarea *soluţiei optimale*.

De exemplu, metoda de extragere a rădăcinii pătrate a unui număr real în reprezentarea zecimală permite determinarea unui număr oricât de apropiat de valoarea reală, considerată valoare optimă. În unele cazuri, valoarea determinată de algoritmul aproximativ este chiar valoarea optimă. Pentru extragerea rădăcinii pătrate a unui număr zecimal aceasta se poate întâmpla acest lucru în cazul în care numărul zecimalelor rădăcinii pătrate este finit.

Un algoritm se numeşte *algoritm euristic* dacă, în contextul său, se folosesc anumite strategii de identificare a soluţiei care asigură obţinerea unor rezultante *rezonabile*, în sensul că aceste rezultate sunt acceptabile conform unor criterii ce se presupun precizate. Cuvântul *euristică* desemnează una sau mai multe strategii folosite în cadrul unui algoritm euristic.

https://biblioteca-digitala.ro / https://unibuc.ro

De exemplu, metoda prin care se poate face *acordarea creditelor la o bancă*, în care se urmăreşte obţinerea unui profit cât mai bun, poate să utilizeze diferite *euristici* mai mult sau mai puţin eficiente cum ar fi:

- Acordarea creditelor în ordine descrescătoare a sumei cerute.
- Acordarea creditelor în ordine crescătoare a raportului dintre suma utä şi garanţiile date pentru rambursare.
- Acordarea creditelor în ordine crescătoare a timpului cerut pentru rambursare.
- Acordarea creditelor în ordine descrescătoare a timpului cerut pentru rambursare.
- Acordarea creditelor în ordinea de prioritate a unor investiţii.

Un algoritm euristic pentru acordarea creditelor poate să combine mai multe euristici pentru a obţine un rezultat cât mai apropiat de profitul maxim. Considerarea uneia sau a alteia dintre euristici se face în funcţie de alţi prametri, care pot să intervină în luarea deciziei, cum ar fi: modul de efectuare a plăţilor, cursul de schimb, valoarea depozitului disponibil etc.

Majoritatea algoritmilor aproximativi au la baza lor euristici, aşa încât în literatura de specialitate nu se face o deosebire netă între aceste două noţiuni.

În continuare, se vor folosi următoarele notaţii şi definiţii:

A - pentru algoritm,

I - o mulţime de date particulare pentru un algoritm A (*intrarea*),

$A(I)$ - rezultatele furnizate de algoritmul A pentru mulţimea I (*ieşirea*),

OPT - algoritm optimal (care determină soluţia optimală),

$OPT(I)$ - rezultatele furnizate de un algoritm optimal ce efectuează calcule în aceleaşi condiţii ca şi algoritmul A.

Se numeşte *performanţă absolută* (în limba engleză *absolute performance ratio*) a unui algoritm de minimizare, respectiv maximizare mărimea:

$$R_A(I) = |A(I)| \,/\, |OPT(I)|, \text{ respectiv } R_A(I) = |OPT(I)| \,/\, |A(I)|.$$

Se observă din definiţie că $R_A(I) \geq 1$.

Se numeşte *eroare a unui algoritm aproximativ A* mărimea R_A definită astfel:

$$R_A = \inf\{r \geq 0 \mid R_A(I) \leq r + 1, \text{ pentru toate problemele individuale ale algoritmului } A\}.$$

Se numeşte eroare asimptotică a unui algoritm aproximativ A mărimea.

$$R_A^{\infty} = \inf\{r \geq 0 \mid \text{există } N \in Z^+ \text{ astfel încât, } R_A(I) \leq r + 1, \text{ pentru orice } I \text{ pentru care } |OPT(I)| \geq N\}.$$

https://biblioteca-digitala.ro / https://unibuc.ro

Pentru o problemă de optimizare P, se numeşte *cea mai bună eroare asimptotică* mărimea:

$$R_{MIN}(P) = \inf\{r \geq 0 \mid \text{pentru rezolvarea problemei } P\text{, există un algoritm aproximativ polinomial } A\text{, astfel încât } R_A^{\infty} = r\}.$$

Algoritmul A, pentru determinarea soluției unei probleme P, se numeşte *algoritm de aproximare absolută* dacă şi numai dacă există o constantă k astfel încât $|OPT(I) - A(I)| \leq k$, pentru orice I.

Algoritmul A se numeşte *algoritm $f(n)$-aproximativ* dacă şi numai dacă:

$$|OPT(I) - A(I)| \,/\, |OPT(I)| \leq f(n).$$

pentru orice I de mărime n. Prin *mărime* a unei mulţimii de date I se înţelege o măsură definită pe mulţimea I.

Algoritmul A_ε se numeşte o *schemă ε-aproximativă* dacă şi numai dacă, pentru orice $\varepsilon > 0$, A_ε generează o soluţie pentru care:

$$|OPT(I) - A(I)| \,/\, |OPT(I)| \leq \varepsilon.$$

O schemă aproximativă se numeşte *schemă aproximativă în timp polinominal* dacă şi numai dacă, pentru orice $\varepsilon > 0$, timpul de calcul este polinomial în măsura definită pe mulţimea I.

O schemă de aproximare pentru care timpul de calcul este polinomial atât în măsura pe mulţimea I cât şi în $1 / \varepsilon$ se numeşte *schemă de aproximare complet polinominală în timp*.

Fie P_1 şi P_2 două probleme. Problema P_1 *se reduce la problem.* P_2, şi se notează $P_1 \alpha P_2$, dacă şi numai dacă există o metodă de rezolvare a lui P_1 cu un algoritm deterministic în timp polinomial utilizând un algoritm deterministic în timp polinomial care rezolvă problema P_2.

Un rol esenţial în studiul complexităţii algoritmilor îl are *problema satisfiabilităţii*. Această problemă constă în identificarea faptului dacă o *formulă booleană* este *adevărată* pentru o mulţime dată de valori ale variabilelor ei.

O problemă P este *NP-dificilă* (în engleză *NP-hard*) dacă şi numai dacă problema satisfiabilităţii se reduce la P.

Un algoritm este *algoritm pseudo-polinomial* în timp dacă are o complexitate polinomială în $LUNGIME(I)$ şi $MAX(I)$, unde $LUNGIME(I)$ este numărul de biţi în reprezentarea mulţimii I şi $MAX(I)$ este valoarea celui mai mare număr din I.

Algoritmii euristici sunt preferaţi algoritmilor exacţi în cazul când ei furnizează soluţii suficient de apropiate de soluţia optimală şi pot fi implementaţi uşor, cu reduceri evidente în timp şi spaţiu.

https://biblioteca-digitala.ro / https://unibuc.ro

1.1.2. Algoritmi aproximativi

În continuare vom prezenta mai mulţi algoritmi ilustrativi pentru diferitele tipuri de aproximare. Pentru fiecare dintre probleme vom da algoritmi de soluţionare şi, eventual, exemple de aplicare ale acestor algoritmi. Pentru majoritatea algoritmilor, vom indica şi alte posibile aplicaţii în diverse domenii de larg interes. Exemple mai complexe vor fi tratate în capitolele următoare.

1.1.2.1. Algoritmi cu aproximare absolută

1.1.2.1.1. *Problema memorării fişierelor pe două benzi*

Punerea problemei. Fiind date două benzi magnetice T_1 şi T_2, de lungimi b_1 şi respectiv b_2 şi n fişiere $f_1, f_2, ..., f_n$ de lungimi $l_1, l_2, ..., l_n$, să se plaseze pe cele două benzi un număr cât mai mare din fişierele date. ■

Aplicaţii posibile: acordarea creditelor la bănci, umplerea containerelor.

Această problemă este *NP-dificilă*. Un algoritm aproximativ pentru soluţionarea ei este următorul:

Algoritmul 1.1. (***Benzi-magnetice***)

1. [Ordonare] Se ordonează fişierele în ordinea crescătoare a lungimii lor; fişierele de aceeaşi lungime se pot lua în orice ordine.
2. [Umplere banda 1] Se încarcă la maximum prima bandă cu fişiere în ordinea stabilită la pasul 1.
3. [Umplere banda 2] Se încarcă la maximum a doua bandă cu fişierele .ămase, considerate tot în ordinea stabilită la punctul 1, şi apoi STOP. ■

Are loc următoarea proprietate, care dovedeşte că Algoritmul 1.1. este algoritm de aproximare absolută:

Teorema 1.1. [HOR78] Fie I mulţimea de fişiere ce urmează a fi memorate şi $A(I)$ numărul de fişiere memorate în conformitate cu algoritmul 1.1. Atunci:

$$|OPT(I) - A(I)| \leq 1,$$

unde $OPT(I)$ este soluţia optimală. ■

Algoritmul 1.1. poate fi generalizat la cazul a k benzi, cu un algoritm de aproximare absolută în care benzile sunt umplute pe rând de fişiere considerate în ordinea crescătoare a lungimii lor, aproximarea fiind în acest caz:

$$|OPT(I) - A(I)| \leq k - 1.$$

https://biblioteca-digitala.ro / https://unibuc.ro

1.1.2.2. Algoritmi $f(n)$-aproximativi

1.1.2.2.1. *Problema comisvoiajorului*

Punerea problemei. Se presupun m oraşe legate între ele fiecare cu fiecare prin drumuri de lungime dată. Se caută un circuit de lungime minimă care să trecă o dată şi numai o dată prin fiecare dintre oraşe. ■

Aplicaţii posibile: stabilirea traseelor poştale sau de altă natură, aprovizionarea magazinelor, planificarea activităţilor.

Această problemă este *NP-dificilă*. În cazul în care distanţele dintre localităţile pe care le parcurge comisvoiajorul satisfac *regula triunghiului* adică:

$$d(a, c) \le d(a, b) + d(b, c), \text{ pentru orice } a, b, c,$$

unde $d(a, b)$ reprezintă distanţa între localităţile a şi b, se poate elabora un algoritm euristic, numit *cel mai apropiat vecin*, şi notat *NN* (din *Nearest Neighbor*).

Algoritmul 1.2. (*NN - Nearest Neighbor*)

1. [Iniţializare] Se pleacă dintr-un oraş oarecare considerat drept oraş iniţial.
2. [Continuarea drumului] Atâta timp cât mai sunt oraşe neparcurse, se alege drept următoarea localitate cea mai apropiată dintre cele neparcurse încă.
3. [Închiderea circuitului] Se revine în oraşul iniţial apoi STOP. ■

Teorema 1.2. [GAR79] Pentru problema comisvoiajorului cu m oraşe şi în ipoteza regulii triunghiului, are loc inegalitatea:

$$(NN(I) - OPT(I)) \,/\, OPT(I) \le 1/2\,(\lceil \log_2 m \rceil - 1).$$

Mai mult, pentru orice valori ale lui m există probleme individuale pentru care:

$$(NN(I) - OPT(I)) \,/\, OPT(I) > 1/3\,(\log_2 (m + 1) - 5/3).$$

O demonstraţie a acestei teoreme se poate găsi în [ROS77]. ■

Din teorema 1.2. rezultă că algoritmul *NN* este un algoritm de aproximare $1/2\lceil \log_2 m \rceil$.

Problema comisvoiajorului va fi tratată pe larg în capitolul 4 unde vor fi indicate multe alte metode de rezolvare şi aplicaţii ce se pot rezolva folosind soluţii ale acestei probleme.

1.1.2.3. Algoritmi ε-aproximativi

1.1.2.3.1. *Planificarea activităţilor independente*

Punerea problemei. Se presupune existenţa a m ($m \ge 2$) procesoare de acelaşi tip. Se cere planificarea pe aceste procesoare a n activităţi independente care necesită

https://biblioteca-digitala.ro / https://unibuc.ro

pentru execuție timpii $t_1, t_2, ..., t_n$, astfel încât timpul total de lucru să fie cât mai mic posibil. ■

Aplicații posibile: tratarea programelor de către sistemele de operare, planificarea sistemelor de servicii, stabilirea planurilor de croire.

Problema obținerii unui timp minim de execuție a mai multor activități independente este *NP-dificilă*. Următoarea strategie de execuție asigură terminarea parcurgerii tuturor activităților într-un timp foarte apropiat de cel optim.

Definiția 1.1. O planificare *LPT* (*longest processing time*) este rezultatul unui algoritm care, ori de câte ori un procesor devine liber, alocă acestui procesor o activitate de durată maximă dintre activitățile nealocate încă. ■

Exemplul 1.1. Fie $m = 3$, $n = 7$ şi $(t_1, t_2, t_3, t_4, t_5, t_6, t_7) = (5, 5, 4, 4, 3, 3, 3)$. Pe figura 1.1.a) este arătată soluția obținută utilizând planificarea *LPT* de durată totală 11 şi pe figura 1.1.b) este dată soluția optimală pentru aceleaşi date de durată totală 9. ■

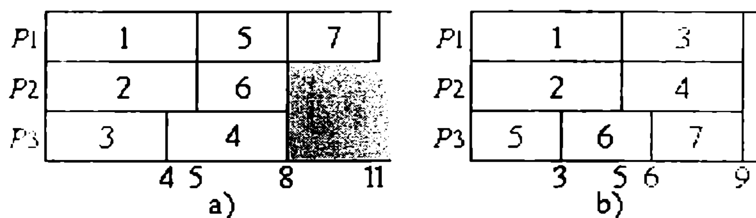


Fig. 1.1.

Teorema 1.3. [HOR78] Fie *OPT*(*I*) timpul optimal şi *LPT*(*I*) timpul furnizat de un algoritm bazat pe planificarea *LPT* de efectuare a mai multor activități pe *m* procesoare. Atunci:

$$(LPT(I) - OPT(I)) / OPT(I) \leq 1 / 3 - 1 / (3m).$$

O demonstrația a acestei teoreme se găseşte în [GRA69]. ■

Există exemple pentru care are loc egalitatea în relația precendentă (vezi exemplul 1.1.)

Din teorema 1.3., rezultă că planificarea *LPT* este, pentru problema planificării activităților independente, o schemă $(1 / 3 - 1 / (3m))$-aproximativă.

1.1.2.3.2. *Problema împachetării*

Punerea problemei. Se consideră *n* obiecte ce urmează să fie încărcate în lăzi de capacitate egală cu *L*. Se cere o strategie care să asigure încărcarea celor *n* obiecte

https://biblioteca-digitala.ro / https://unibuc.ro

într-un număr minim de lăzi, fiecare obiect fiind încărcat integral într-o singură ladă. Obiectul i necesită l_i unităţi de capacitate. Se presupune că $l_i \leq L$, pentru orice i=1,2,...,n. Altfel problema nu ar avea soluţie pentru că ar exista obiecte care nu încap în lăzi.■

Aplicaţii posibile: alocarea memoriei, planificarea activităţilor pe clase de prioritate, coletărie.

Problema este *NP-dificilă.*

Pentru rezolvarea acestei probleme există mai multe euristici dintre care menţionăm următoarele patru euristici simple:

1. *First Fit* (*FF*). Lăzile se presupun numerotate şi iniţial sunt goale. Obiectele sunt parcurse în ordinea 1, 2, ..., n. Pentru a împacheta obiectul i, se caută cel mai mic indice j astfel încât lada j este completă pînă la nivelul r, $r \leq L - l_i$, şi se introduce obiectul i în lada j.
2. *Best Fit* (*BF*). Condiţiile iniţiale pentru obiecte şi lăzi sunt aceleaşi ca mai sus. Pentru obiectul i, se caută cel mai mic indice j astfel încât lada j este umplută până la nivelul r, $r \leq L - l_i$, r fiind cât mai mare posibil, şi se introduce obiectul i în lada j.
3. *First Fit Decreasing* (*FFD*). Se ordonează obiectele în ordinea descrescătoare a lungimilor lor, $l_i \geq l_{i+1}$, pentru i = 1, 2, ..., n, şi se aplică euristica *FF*.
4. *Best Fit Decreasing* (*BFD*). Se ordonează obiectele ca la euristica *FFD* şi se aplică euristica *BF*.

Exemplul 1.2. Fie L = 10, n = 6 şi (l_1, l_2, l_3, l_4, l_5, l_6) = (5, 6, 3, 7, 5, 4). O soluţie optimală a problemei este arătată în figura 1.2., fiind indicate numai dimensiunile obiectelor puse în lăzi. Aplicând metoda *FF* se obţine o soluţie cu 4 lăzi ce conţin în ordine elementele (1, 3), (2, 6), (4) şi (5). Metoda *BF* dă o soluţie tot cu 4 lăzi, care conţin elementele (1, 5), (2, 3), (4) şi (6). Metodele *FFD* şi *BFD* dau soluţia optimală, abstracţie făcând de o interschimbare între prima şi ultima ladă la soluţia prezentată în figura 1.2. ■

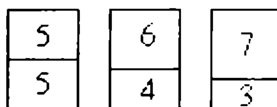


Fig. 1.2.

Se pot construi exemple pentru care euristicile *FFD* şi *BFD* dau rezultate mai bune decât *FF* şi *BF* dar şi exemple în care *FF* sau *BF* sunt mai eficiente.

https://biblioteca-digitala.ro / https://unibuc.ro

Teorema 1.4. [GAR79] Pentru toate problemele individuale de împachetare are loc inegalitatea:

$$FF(I) \leq 17/10\ OPT(I) + 2.$$

În afară de aceasta, există probleme pentru care OPT(I) este oricât de mare şi:

$$FF(I) \geq 17/10\ (OPT(I) - 1).$$

O demonstraţie a acestei teoreme se poate găsi în [JOH74a]. ■

Teorema 1.5. [GAR79] Pentru toate problemele individuale de împachetare are loc inegalitatea:

$$FFD(I) \leq 11/9\ OPT(I) + 4.$$

Există probleme individuale pentru care $OPT(I)$ este oricât de mare şi:

$$FFD(I) \geq 11/9\ OPT(I).$$

O demonstraţie a acestei teoreme se poate găsi în [JOH74a]. ■

Pentru metodele *BF* şi *BFD* se pot demonstra proprietăţi asemănătoare. Acestea dovedesc faptul că cele patru metode fac parte din clasa metodelor ε - aproximative.

1.1.2.3.3. *Problema rucsacului*

Punerea problemei. Fiind date mulţimile $P = \{p_1, p_2, ..., p_r\}$ şi $W = \{w_1, w_2, ..., w_r\}$, unde P este mulţimea profiturilor obiectelor 1, 2, ..., r şi W este mulţimea ponderilor acestor obiecte, se cere:

să se maximizeze $\sum p_i\delta_i$ cu condiţia $\sum w_i\delta_i \leq M$ şi $\delta_i \in \{0, 1\}$, pentru $i = 1, 2, ..., r$,

unde M reprezintă capacitatea rucsacului în care se încarcă obiectele. Cu alte cuvinte, să se determine o submulţime de obiecte, a căror pondere totală să nu depăşească M, capacitatea rucsacului, şi să dea un profit total maxim posibil. ■

Aplicaţii posibile: încărcarea containerelor, stabilirea rutelor pe calea ferată.

Se presupune că obiectele au fost ordonate în ordinea necrescătoare a densităţilor profitului, adică $p_i / w_i \geq p_{i+1} / w_{i+1}$, pentru $i = 1, 2, ..., r - 1$. Fie $L(I, P, W, M)$ profitul obţinut pe calea umplerii, în ordinea necrescătoare a rapoartelor p_i / w_i, a acelei părţi din rucsacul de capacitate M care a rămas neocupată după ce obiectele din I au fost introduse în rucsac. P şi W au semnificaţiile de mai sus şi se presupune că $r > 0$ şi $\sum_{i \in I} w_i \leq M$.

Algoritmul 1.3. ***L(I, P, W, M)***

1. [Iniţializări] Se face $L = \sum_{i \in I} p_i$, $D = M - \sum_{i \in I} w_i$ şi $i = 1$.

https://biblioteca-digitala.ro / https://unibuc.ro

2. [Testare obiect] Dacă $i \notin I$ şi $w_i \leq D$, atunci se face $I = I \cup \{i\}$, $L = L + p_i$ şi $D = D - w_i$.
3. [Următorul obiect] Se face $i = i + 1$; dacă $i \leq r$, atunci se trece la pasul 2.
4. [Terminare] Furnizează L şi STOP. ■

Este evident că acest algoritm are complexitatea $O(r)$.

Pentru rezolvarea problemei rucsacului se poate folosi următoarea euristică. Pentru un k dat, se generează succesiv toate cele $\binom{r}{k}$ submulţimi de obiecte. Dacă submulţimea I curent generată satisface condiţia $\sum_{i \in I} w_i > M$, unde M este capacitatea rucsacului, ea este eliminată. Altfel spaţiul rămas liber în rucsac, adică $M - \sum_{i \in I} w_i$, este încărcat utilizând, în ordinea descrescătoare a raportului p_i / w_i, obiecte ce nu aparţin mulţimii I şi fără a depăşi pe M. Dintre toate aceste încărcări se alege aceea ce are valoarea profitului maximă.

În algoritmul următor, vom considera că r este numărul total de obiecte, *ponderea unei combinaţii* este suma ponderilor obiectelor ei şi k este un întreg nenegativ care defineşte ordinul algoritmului.

Algoritmul 1.4. ε-*APROX*(P, W, M, r, k)

1. [Iniţializare] Se face $PMAX = 0$.
2. [Maximizare profit] Pentru toate combinaţiile I de mărime cel mult $k \leq r$ şi pondere cel mult M se face:

$$PMAX = \max\{PMAX, L(I, P, W, M)\}.$$

3. ĂTerminareÎ Furnizează $PMAX$ şi STOP. ■

Exemplul 1.3. Fie problema rucsacului cu $r = 8$ obiecte, mărimea rucsacului fiind $M = 110$, profiturile $P = \{11, 21, 31, 33, 43, 53, 55, 65\}$ şi ponderile $W = \{1, 11, 21, 23, 33, 43, 45, 55\}$. Obiectele sunt ordonate necrescător în raport cu p_i / w_i.

Soluţia optimală este definită de obiectele 1, 2, 3, 5 şi 6. Profitul optimal este $P^* = 159$ şi ponderea este $D^* = 109$.

Pentru diferiţi k se obţin următoarele soluţii aproximative:

Pentru $k = 0$, $PMAX$ este chiar marginea inferioară $L(\varnothing, P, W, M)$; $PMAX = 139$; $\delta = (1,1,1,1,1,0,0,0)$; $D = \Sigma\, \delta_i w_i = 89$; $100\,(P^* - PMAX) / P^* = 2000 / 159 = 12.6\%$. Se analizează o singură combinaţie: $I = \varnothing$.

Pentru $k = 1$, $PMAX = 151$; $\delta = (1,1,1,1,0,0,1,0)$; $D = 101$; $100\,(P^* - PMAX)/P^* = 800 / 159 = 5.03\%$. Se analizează 9 combinaţii: $I = \varnothing$, $I = \{1\}$, $I = \{2\}$, ..., $I = \{8\}$.

https://biblioteca-digitala.ro / https://unibuc.ro

Pentru $k = 2$, $PMAX = P^* = 159$; $\delta = (1,1,1,0,1,1,0,0)$; $D = 109$. Se analizează 37 de combinaţii: $I = \emptyset, I = \{1\}, I = \{2\}, ..., I = \{8\}, I = \{1,2\}, I = \{1,3\}, ..., I = \{7,8\}$. Se determină chiar soluţia optimală. ■

Teorema 1.6. [HOR78] Algoritmul *ε-APROX* este un algoritm ε-aproximativ pentru care $|(P^* - PMAX) / P^*| < 1 / (k + 1)$, pentru $k > 0$. Timpul de lucru are ordinul $O(kr^{k+1})$ şi memoria necesară este de ordinul $O(r)$.

Demonstraţie. a) Să demonstrăm că $|(P^* - PMAX) / P^*| < 1 / (k + 1)$. Să presupunem că soluţia optimală a fost obţinută prin introducerea în rucsac a următoarelor j obiecte cu $\hat{p}_1, \hat{p}_2, ..., \hat{p}_j$ şi $\hat{w}_1, \hat{w}_2, ..., \hat{w}_j$. Pentru $j \le k$, algoritmul *ε-APROX* furnizează soluţia P^*, deoarece toate combinaţiile de mărime mai mică sau egală cu k sunt testate, printre ele fiind şi soluţia optimală. Fie $j > k$. Presupunem că variabilele din soluţia optimală au fost ordonate astfel încât:

1. $\hat{p}_1, \hat{p}_2, ..., \hat{p}_k$ sunt cele mai mari k valori $\hat{p}_i$ din P^* şi
2. celelalte j-k obiecte rămase sunt ordonate necrescător după densităţile profitului:

$$\hat{p}_{k+1} / \hat{w}_{k+1} \ge \hat{p}_{k+2} / \hat{w}_{k+2} \ge ... \ge \hat{p}_j / \hat{w}_j.$$

Condiţia 1. implică $\hat{p}_{k+t} \le P^* / (k + 1)$, pentru $t = 1,2,...,j$-k. La o anumită etapă a pasului 2 al algoritmului *ε-APROX*, combinaţia I corespunde şirului $\hat{p}_1, \hat{p}_2, ..., \hat{p}_k$. Să considerăm mai în detaliu această etapă. În procesul calculului marginii inferioare a soluţiei $L(I,P,W,M)$ fie (p_m, w_m) primul obiect din P^*, adică unul dintre obiectele ce au profiturile $\hat{p}_{k+1}, ..., \hat{p}_j$ care nu este introdus în rucsac. Dacă un astfel de obiect nu există, atunci s-a obţinut soluţia optimală. Dacă acest obiect există, atunci se observă că:

1. Un spaţiu mai mic decât w_m din capacitatea rucsacului este neocupat.
2. Partea din rucsac umplută până acum cu algoritmul marginii inferioare a fost ocupată cu o densitate a profitului mai mare sau egală cu cea utilizată pentru ocuparea părţii rămase folosind obiecte din $\hat{p}_{k+1}, \hat{p}_{k+2}, ..., \hat{p}_r$ în această ordine, deci $\sum_{i=k+1}^{m-1} \hat{p}_i \le L(I, P, W, M)$.
3. Maximul densităţii la această etapă este p_m / w_m, aşa încât spaţiul rămas al rucsacului nu poate contribui cu mai mult decât p_m pentru obţinerea lui P^*.

Rezultă că, în virtutea punctelor 2. şi 3. de mai sus:

$$P^* = P_I + \sum_{i=k+1}^{j} \hat{p}_i = P_I + \sum_{i=k+1}^{m-1} \hat{p}_i + \sum_{i=m}^{j} \hat{p}_i < P_I + L(I,P,W,M) + p_m.$$

Deoarece $p_m \le P^* / (k + 1)$, se obţine $P^* - (P_I + L(I, P, W, M)) < P^* / (k + 1)$. Prin definiţie, $PMAX \ge P_I + L(I, P, W, M)$. Prin urmare:

$$|(P^* - PMAX) / P^*| < 1 / (k + 1), \text{ pentru } k > 0.$$

https://biblioteca-digitala.ro / https://unibuc.ro

b) Pentru orice k, algoritmul *ε-APROX* are o complexitate în timp polinomial.

Deoarece numărul de combinaţii de mărime j este $\binom{r}{j}$, rezultă că pasul 2 este executat de $\sum_{j=1}^{k}\binom{r}{j} < kr^k$ ori. Fiecare execuţie a pasului 2 necesită un timp de ordinul cel mult r. Prin urmare, timpul total are ordinul $O(kr^{k+1})$.

Datele memorate sunt δ, I şi valorile câtorva variabile simple. Rezultă că memoria necesară are $O(r)$. ■

Teorema ce urmează precizează cât de bună este marginea superioară menţionată în teorema precedentă.

Teorema 1.7. [HOR78] Pentru fiecare k există probleme ale rucsacului pentru care $|OPT - PMAX| / OPT$ se apropie oricât de mult de $1 / (k + 1)$. ■

1.1.2.4. Scheme aproximative în timp polinomial

1.1.2.4.1. *Planificarea activităţilor independente*

Vom prezenta o altă metodă pentru rezolvarea problemei enunţată în paragraful 1.1.2.3.1. şi anume vom folosi următoarea euristică:

1. Pentru k dat ($k < n$) se caută o soluţie optimă de planificare a k activităţi cu cele mai mari durate de execuţie.
2. Pentru cele $n - k$ activităţi rămase se aplică strategia *LPT*.

Exemplul 1.4. Fie $m = 2$, $n = 6$; $(t_1, t_2, t_3, t_4, t_5, t_6) = (8, 6, 5, 4, 4, 1)$ şi $k = 4$. Cele mai mari 4 duratele de execuţie sunt 8, 6, 5, 4. Pe figura 1.3.a) este arătată o planificare optimală ce necesită 12 unităţi de timp pentru aceste 4 activităţi. Utilizând strategia *LPT* pentru cele două activităţi rămase se obţine soluţia din figura 1.3.b) cu timpul total 15. Soluţia optimală are timpul total 14 şi este arătată pe figura 1.3.c). ■

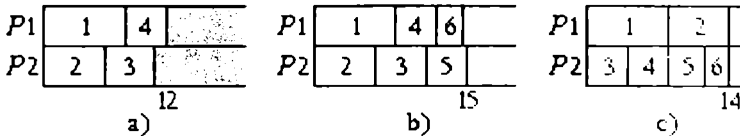


Fig. 1.3.

Teorema 1.8. [HOR78] Fie o problemă de planificare a activităţilor cu m procesoare. Dacă $A(I)$ este timpul generat de euristica precedentă atunci:

$$|A(I) - OPT(I)| / OPT(I) \le (1 - 1/m) / (1 + \lfloor k/m \rfloor).$$

Timpul necesar pentru obţinerea soluţiei are ordinul:

$$O(n \log n + m^{((m-1)/\varepsilon - m)}), \text{ unde } \varepsilon \le (1 - 1/m) / (1 + \lfloor k/m \rfloor). \ ■$$

Teorema anterioară arată că euristica propusă este o schemă de aproximare în timp polinomial.

https://biblioteca-digitala.ro / https://unibuc.ro

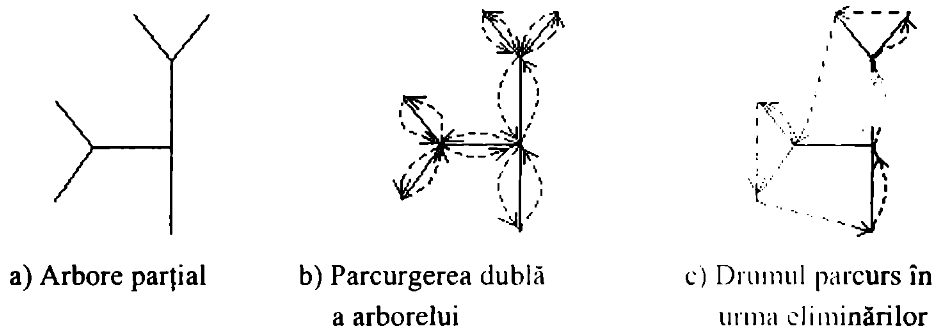
1.1.2.4.2. *Problema comisvoiajorului*

Vom prezenta o altă euristică pentru rezolvarea problemei comisvoiajorului enunţată în paragraful 1.1.2.2.1. Pentru această problemă pot fi elaboraţi algoritmi polinomiali care se bazează pe utilizarea noţiunilor din teoria grafurilor de *arbore parţial*, *graf eulerian* şi *drum eulerian* pe care le definim în continuare.

Un *arbore parţial* al unui graf *G* neorientat este un subgraf conex, aciclic şi care conţine toate nodurile lui *G*. Un *arbore parţial minimal* este un arbore parţial pentru care suma lungimilor muchiilor este minimă. Pentru construirea unui arbore parţial minimal se cunosc algoritmi polinomiali.

Un graf se numeşte *eulerian* dacă este conex şi toate nodurile lui au grad par. Se numeşte *ciclu eulerian* într-un graf un ciclu ce parcurge exact o singură dată fiecare muchie. Un ciclu eulerian poate fi identificat în timp polinomial.

O euristică ce permite elaborarea unui algoritm polinomial pentru problema comisvoiajorului, cu proprietatea inegalităţii triunghiului, este descrisă în continuare şi ilustrată pe figura 1.4. Mai întâi se determină un arbore parţial minimal pentru graful dat (figura 1.4.a). Apoi se construieşte un drum ce constituie dubla parcurgere în sensuri opuse a fiecărei muchii a arborelui (figura 1.4.b). În continuare, din acest drum se elimină localităţile deja parcurse (figura 1.4.c). Inegalitatea triunghiului asigură obţinerea unui drum ce nu poate fi mai mare decât cel precedent.



a) Arbore parţial   b) Parcurgerea dublă a arborelui   c) Drumul parcurs în urma eliminărilor

Fig. 1.4.

Teorema 1.9. [GAR79] Pentru toate problemele comisvoiajorului în care este satisfăcută inegalitatea triunghiului, notând cu *MST* euristica anterioară, are loc inegalitatea:

$$MST(I) < 2\ OPT(I).\ \blacksquare$$

Din cele de mai sus rezultă că euristica propusă este o schemă de aproximare în timp polinomial.

https://biblioteca-digitala.ro / https://unibuc.ro

1.1.2.5. Scheme de aproximare complet polinomiale

În continuare se prezintă trei tehnici de aproximare complet polinomiale, exemplificate pentru problema cunoscută sub numele de *programare liniară booleană*.

Punerea problemei: Se cere identificarea valorii:

$$\max \sum_{i=1}^{n} p_i x_i,$$

cu condiţiile:

$$\sum_{i=1}^{n} a_{ij} x_i \leq b_j, \text{ pentru } j = 1, 2, ..., m,$$

$$x_i = 0 \text{ sau } 1, \text{ pentru } i = 1, 2, ..., n,$$

$$p_i,\ a_{ij} \geq 0, \text{ pentru } i = 1, 2, ..., n \text{ şi } j = 1, 2, ..., m. \blacksquare$$

Aplicaţii posibile: aprovizionare, dietă, încărcare vagoane, croire.

Fără a micşora generalitatea se poate prespune că $a_{ij} \leq b_j$, $i=1,...,n$ şi $j = 1,..., n$. Altfel valoarea lui x_i trebuie să fie 0 şi deci variabila respectivă poate fi eliminată.

Dacă $1 \leq k \leq n$, atunci asignarea $x_i = y_i$, $i = 1, 2, ..., k$, se numeşte *asignare efectuabilă* dacă şi numai dacă există cel puţin o soluţie astfel încât x_i să aibă valoarea y_i, pentru $i = 1, 2, ..., k$. O *completare* a unei asignări efectuabile $x_i = y_i$ este orice soluţie $y_1, ..., y_n$ cu $x_i = y_i$, pentru $i = 1, 2, ..., k$.

Fie $x_i = y_i$ şi $x_i = z_i$, $i = 1, 2, ..., k$, două asignări efectuabile pentru care $y_j \neq z_j$ măcar pentru un j, $1 \leq j \leq k$, Dacă $\Sigma p_i y_i = \Sigma p_i z_i$, atunci $y_1, ..., y_k$ *domină* pe $z_1, ..., z_k$ dacă şi numai dacă există o completare $y_1,..., y_k, y_{k+1}, ..., y_n$ astfel încât $\sum_{i=1}^{n} p_i y_i \geq \sum_{i=1}^{n} p_i z_i$, pentru toate completările $z_1, ..., z_n$ ale lui $z_1, ..., z_k$.

Un mod de rezolvare a problemei enunţate constă în generarea sistematică a tuturor asignărilor efectuabile plecând de la asignarea nulă. Fie $S^{(i)}$ mulţimea tuturor asignărilor efectuabile pentru $x_1, x_2, ..., x_i$. Atunci $S^{(0)}$ este asignarea nulă şi $S^{(n)}$ este mulţimea tuturor completărilor. Soluţia problemei este dată de o asignare din $S^{(n)}$ care maximizează funcţia obiectiv. Identificarea soluţiei constă în generarea lui $S^{(i+1)}$ din $S^{(i)}$, pentru $1 \leq i < n$. Dacă o mulţime $S^{(i)}$ conţine două asignări efectuabile $y_1, ..., y_k$ şi $z_1, ..., z_j$ astfel încât $\Sigma\, p_i y_i = \Sigma\, p_i z_i$, atunci utilizarea unor reguli de dominare, dacă există, permite eliminarea asignărilor dominate.

Există cazuri în care regulile de dominare permit eliminarea unor asignări efectuabile, chiar dacă $\Sigma\, p_i y_i \neq \Sigma\, p_i z_i$. Drept exemplu se poate vedea problema rucsacului (vezi paragraful 1.1.2.3.3. şi Secţiunea 5.5. din [HOR78]).

În continuare vom descrie metode euristice de rezolvare a acestei probleme.

https://biblioteca-digitala.ro / https://unibuc.ro

1.1.2.5.1. *Rotunjirea*

Ideea acestei euristici constă în schimbarea suficient de mică a valorilor lui p_i în q_i astfel încât noua problemă obținută să admită soluții destul de apropiate de soluțiile problemei inițiale şi să fie uşor de calculat.

Exemplul 1.5. Fie $(p_1, p_2, p_3, p_4) = (1.1, 2.1, 1001.6, 1002.3)$ valorile problemei I date şi valorile rotunjite $(q_1, q_2, q_3, q_4) = (0, 0, 1000, 1000)$ ale problemei I' aproximative. Diferența maximă de 7.1 dintre soluțiile problemelor I şi I' are loc dacă $x_i = 1$, $1 \le i \le 4$ este o soluție pentru I (şi deci şi pentru I'). Deoarece o soluție efectuabilă este $x_1 = x_2 = x_3 = 0$, $x_4 = 1$, rezultă că $OPT(I) \ge 1002.3$. Rezultă că $(OPT(I) - OPT(I')) / OPT(I) \le 0.007$. Pentru problema I asignările efectuabile variabilelor (x_1, x_2, x_3, x_4) în mulțimile $S^{(0)}, ..., S^{(4)}$ sunt următoarele:

$S^{(0)} = \{(0000)\}$;

$S^{(1)} = S^{(0)} \cup \{(1000)\}$;

$S^{(2)} = S^{(1)} \cup \{(0100), (1100)\}$;

$S^{(3)} = S^{(2)} \cup \{(0010), (1010), (0110), (1110)\}$;

$S^{(4)} = S^{(3)} \cup \{(0001), (1001), (0101), (1101), (0011), (1011), (0111), (1111)\}$.

Mulțimile $S^{(i)}$ determină următoarele mulțimi de valori:

$S^{(0)} \to \{0\}$;

$S^{(1)} \to \{0, 1.1\}$;

$S^{(2)} \to \{0, 1.1, 2.1, 3.2\}$;

$S^{(3)} \to \{0, 1.1, 2.1, 3.2, 1001.6, 1002.7, 1003.7, 1004.8\}$;

$S^{(4)} \to \{0, 1.1, 2.1, 3.2, 1001.6, 1002.3, 1002.7, 1003.4, 1003.7, 1004.4, 1004.8, 1005.5, 2003.9, 2005, 2006, 2007.1\}$.

În mod asemănător, pentru problema I' rezultă mulțimile:

$S^{(0)} \to \{0\}$;

$S^{(1)} \to \{0\}$;

$S^{(2)} \to \{0\}$;

$S^{(3)} \to \{0, 1000\}$;

$S^{(4)} \to \{0, 1000, 2000\}$.

Ținând seama că pentru problemele I, respectiv I', sumele $\sum_{i=0}^{n} |S^{(i)}|$ au valorile 31, respectiv 8, rezultă că problema I' poate fi rezolvată într-un timp aproximativ de patru ori mai mic decât cel necesar pentru rezolvarea problemei I. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Teorema 1.10. [HOR78] Fie *LB* o estimare pentru $OPT(I)$ astfel încât $OPT(I) \geq LB$ şi un ε dat. Atunci, utilizând euristica descrisă mai înainte, se poate obţine o soluţie a problemei I' ce satisface condiţia:

$$(OPT(I) - OPT(I')) \,/\, OPT(I) \leq \varepsilon$$

într-un timp de ordinul $O(n^3/\varepsilon)$. ■

Rotunjirea, aşa cum a fost descrisă mai sus ca metodă generală, conduce la scheme aproximative în timp de $O(n^3/\varepsilon)$. Euristica propusă poate fi specializată însă în vederea obţinerii unor soluţii mai eficiente. Un exemplu ilustrativ îl constitue *problema rucsacului*.

Fie I datele problemei rucsacului, ε eroarea dorită şi $OPT(I)$ valoarea soluţiei optimale. Mai întâi se caută o estimare bună *UB* pentru $OPT(I)$. Pentru aceasta, cele n obiecte se ordonează astfel încât $p_i / w_i \geq p_{i+1} / w_{i+1}$, pentru $i = 1, 2, ..., n$, p_i fiind profitul şi w_i ponderea obiectului i. Apoi se caută cel mai mare j astfel încât $\sum_{i=1}^{j} w_i \leq M$. Dacă $j = n$, atunci soluţia optimală este w_i, $i = 1, 2, ..., n$ şi $OPT(I) = \Sigma p_i$. Fie $j < n$ şi $UB = \sum_{i=1}^{j+1} p_i$. Se poate arăta că $1/2\, UB \leq OPT(I) < UB$. Ţinând seama de felul în care a fost ales j, rezultă inegalitatea $OPT(I) < UB$. Cealaltă inegalitate, $1/2\, UB \leq OPT(I)$, rezultă din faptul că:

$$OPT(I) \geq \sum_{i=1}^{j} p_i \text{ şi } OPT(I) \geq \max\{\sum_{i=1}^{j} p_i, p_{j+1}\}.$$

Prin urmare:

$$2\, OPT(I) \geq \sum_{i=1}^{j+1} p_i = UB.$$

Fie $\delta = UB\varepsilon^2/9$. Cele n obiecte se împart în două clase MARI şi MICI, unde MARI include toate obiectele cu $p_i > UB\varepsilon/3$ şi MICI conţine restul obiectelor. Fie r numărul obiectelor din MARI. Fiecare p_i din MARI se înlocuieşte cu $q_i = \lfloor p_i/\delta \rfloor$, această înlocuire reprezentând o rotunjire. Urmează rezolvarea problemei rucsacului utilizând aceste r obiecte cu q_i-urile ce le corespund şi aplicând metoda programării dinamice. Fie $S^{(r)}$ mulţimea perechilor (x, y) astfel obţinute. Pentru fiecare pereche (x, y) din $S^{(r)}$, capacitatea $M - y$ rămasă liberă este încărcată cu obiecte din MICI în ordinea necrescătoare a rapoartelor p_i / w_i. Soluţia este dată de umplerea cu valoare maximă posibilă.

Exemplul 1.6. Fie $n = 5$, $(p_1,p_2,p_3,p_4,p_5) = (w_1,w_2,w_3,w_4,w_5) = (1,2,10,$[illegible]$0,1000)$, $M = 1112$ şi $\varepsilon = 1/10$. Obiectele sunt în ordinea necrescătoare a rapoartelor p_i / w_i, $UB = \sum_{i=1}^{5} p_i = 1113$, $\delta = 3.71 / 3$ şi $\varepsilon\, UB / 3 = 37.1$. Rezultă că MICI include obiectele 1, 2 şi 3 şi MARI include elementele 4 şi 5. Valorile q_4 şi q_5 sunt respectiv $\lfloor p_4/\delta \rfloor = 94$ şi $\lfloor p_5 / \delta \rfloor = 946$.

https://biblioteca-digitala.ro / https://unibuc.ro

Pentru problema rucsacului cu $n = 2$, $M = 1112$, $(q_4, w_4) = (94, 100)$ şi $(q_5, w_5) = (946, 1000)$, rezultă pentru $S^{(0)}$ valorile $\{(0, 0)\}$; pentru $S^{(1)}$ valorile $\{(0, 0), (94, 100)\}$ şi pentru $S^{(2)}$ valorile $\{(0, 0), (94, 100), (946, 1000), (1040, 1100)\}$. Încărcând (0, 0) din MICI se obţine perechea (13, 13). Pentru perechile (94, 100), (946, 1000) şi (1040, 1100) se obţine (107, 113), (959, 1013), respectiv (1043, 1103). Răspunsul este dat de perechea (1043, 1103), care corespunde mulţimii $_1, x_2, x_3, x_4, x_5) = (1, 1, 0, 1, 1)$ şi $\Sigma p_i x_i = 1103$. ■

Teorema 1.11. [IBA75] Algoritmul descris anterior este ε-aproximativ pentru problema rucsacului. ■

Timpul necesar pentru ordonarea mărimilor p_i / w_i are ordinul $O(n \log n)$. Valoarea UB poate fi calculată într-un timp $O(n)$. Deoarece $OPT(I) \leq UB$, există cel mult $UB / \delta = 9 / \varepsilon^2$ perechi în $S^{(i)}$ în soluţia pentru MARI şi timpul necesar pentru obţinerea lui $S^{(r)}$ este $O(r / \varepsilon^2) \leq O(n / \varepsilon^2)$. Completarea lui $S^{(r)}$ cu obiecte din MICI necesită un timp $O(|\text{MICI}|)$ şi, deoarece $|S^{(r)}| \leq 9 / \varepsilon^2$, rezultă că MICI se încheie în cel mult $O(n / \varepsilon^2)$ unităţi de timp. Rezultă că timpul total al algoritmului descris este de $O(n (\log n + 1 / \varepsilon^2))$.

1.1.2.5.2. *Partiţionarea pe intervale*

Această euristică constă în generarea unei clase restrânse de asignări ıtru $S^{(0)}$, $S^{(1)}, \ldots, S^{(n)}$. Fie P_i maximul sumei $\sum_{j=1}^{i} p_j x_j$ al asignărilor generate pentru $S^{(i)}$. Intervalul $[0, P_i]$ este împărţit în subintervale de lungime $P_i\varepsilon / (n - 1)$, ultimul subinterval putând fi de lungime mai mică. Toate asignările efectuate în $S^{(i)}$ cu $\sum_{j=1}^{i} p_j x_j$ în acelaşi subinterval se consideră ca având acelaşi profit $\sum_{j=1}^{i} p_j x_j$ şi se folosesc criterii de dominare pentru reţinerea uneia dintre aceste asignări. Cu $S^{(i)}$ astfel obţinut, se generează $S^{(i+1)}$. Deoarece numărul de subintervale pentru fiecare $S^{(i)}$ este cel mult $\lceil n / \varepsilon \rceil + 1$, rezultă că $\sum_{i=1}^{n} |S^{(i)}| = O(n^2 / \varepsilon)$.

Eroarea introdusă în fiecare asignare efectuabilă în urma eliminării din $S^{(i)}$ a altor asignări este mai mică decât lungimea subintervalului. Această eroare se poate propaga aditiv de la $S^{(1)}$ până la $S^{(n)}$. Fie $PART(I)$ valoarea optimală obţinută utilizând partiţionarea pe intervale şi $OPT(I)$ valoarea optimală a problemei de m‹ ımizare. Rezultă că:

$$OPT(I) - PART(I) \leq (\varepsilon \sum_{i=1}^{n-1} P_i) / (n - 1)$$

https://biblioteca-digitala.ro / https://unibuc.ro

şi, deoarece $P_i \le OPT(I)$,

$$(OPT(I) - PART(I)) / OPT(I) \le \varepsilon.$$

În multe cazuri, algoritmul propus poate fi accelerat plecând de la o estimare LB a lui $OPT(I)$ astfel încât $OPT(I) \ge LB$. Lungimea subintervalelor poate fi luată egală cu $LB\ \varepsilon / (n - 1)$ în loc de $P_i\ \varepsilon / (n - 1)$. Pentru o asignare având o valoare mai mare ca LB, mărimea subintervalului se calculează în modul descris mai înainte.

Exemplul 1.7. Fie problema rucsacului enunţată în exemplul precedent. Mulţimile corespunzătoare de valori pentru $S^{(i)}$ sunt: pentru $S^{(0)}$ valoarea {. }, pentru $S^{(1)}$ valorile {0, 1}, pentru $S^{(2)}$ valorile {0, 1, 2, 3}, pentru $S^{(3)}$ valorile {0, 1, 2, 3, 10, 11, 12, 13}, pentru $S^{(4)}$ valorile {0, 1, 2, 3, 10, 11, 12, 13, 100, 101, 102, 103, 110, 111, 112, 113}, pentru $S^{(5)}$ valorile {0, 1, 2, 3, 10, 11, 12, 13, 100, 101, 102, 103, 110, 111, 112, 113, 1000, 1001, 1002, 1003, 1010, 1011, 1012, 1013, 1100, 1101, 1102, 1103, 1110, 1111, 1112}, cu menţiunea că, deoarece $p_k = w_k$, în $S^{(i)}$ au fost considerate numai valorile p_k. Pentru $\varepsilon = 1/10$ şi $OPT(I) \ge LB \ge 1000$ se obţine o lungime a subintervalului $LB\ \varepsilon / (n - 1) = 1000/40 = 25$. Intervalele sunt [0, 25), [25, 50), ... Toate valorile din $S^{(0)}$, $S^{(1)}$, $S^{(2)}$, $S^{(3)}$ fiind mai mici decât 25, rezultă că mulţimile valorilor corespunzătoare lor coincid, fiind toate {0}. În $S^{(4)}$ valorile mai mici ca 25 se înlocuiesc cu 0 şi cele mai mari ca 100 se înlocuiesc cu 100. Rezultă că mulţimea valorilor pentru $S^{(4)}$ este {0, 100}. În mod asemănător se obţine mulţimea valorilor pentru $S^{(5)}$ care este {0, 100, 1000, 1100}. Aplicând noii probleme metoda programării dinamice se obţine soluţia $PARTI(I) = 1100$, care corespunde mulţimii de valori $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 0, 1, 1)$ al formulării iniţiale. Această soluţie ai poate fi îmbunătăţită utilizând o euristică prin care se schimbă unele valori x_i din 0 în 1. Eroarea soluţiei obţinute mai sus este:

$$(OPT(I) - PART(I)) / PART(I) = 12/1112 < 0.011 < \varepsilon. \blacksquare$$

1.1.2.5.3. *Separarea*

Fie o problemă pentru care s-a obţinut un $S^{(i)}$ cu soluţii având valorile $\sum_{j=1}^{i} p_j x_j$: 0, 3.9, 4.1, 7.8, 8.2, 11.9, 12.1. În ipoteza unor intervale de lungime $P_i\varepsilon / (n - 1) = 2$, subintervalele sunt [0, 2), [2, 4), [4, 6), [6, 8), [8, 10), [10, 12) şi [12, 14). Fiecare din valorile precedente aparţine unor intervale distincte, aşa încât euristica partiţionării nu poate fi aplicată. Există însă trei perechi de asignări care diferă prin mai puţin decât $P_i\varepsilon / (n - 1)$. Aplicând reguli de dominare, se pot reţine numai patru asignări, eroarea introdusă fiind de cel mult $P_i\varepsilon / (n - 1)$. Mai precis, fie $a_0, a_1, ..., a_r$ valori distincte ale sumei $\sum_{j=1}^{i} p_j x_j$ din $S^{(i)}$. Fie $a_0 < a_1 < ... < a_r$. O nouă mulţime I poate fi obţinută din $S^{(i)}$

https://biblioteca-digitala.ro / https://unibuc.ro

parcurgând şirul precedent de la stânga la dreapta şi reţinând un element numai dacă valoarea lui este mai mare decât ultimul reţinut cu $P_i\varepsilon / (n - 1)$. Procedeul propus este descris de următorul algoritm:

Algoritmul 1.5. (***Separare***)

1. [Iniţializare] Se face I = {asignarea corespunzătoare lui a_0} şi $XP = a_0$.
2. [Ciclare] Pentru $j = 1, 2, ..., r$ se execută pasul 3, apoi STOP.
3. [Modificare asignare] Dacă $a_j > XP + P_j\varepsilon / (n - 1)$, atunci se introduce asignarea corespunzătoare lui a_j în I şi se face $XP = a_j$. ■

Exemplul 1.8. Fie problema rucsacului cu profiturile $(p_1, p_2, p_3, p_4, p_5) = (3, 1, 5.1, 5.1, 5.1)$ şi $LB\varepsilon / (n - 1) = 2$. Folosind algoritmul descris, se obţin următoarele mulţimi de valori: pentru $S^{(0)}$: {0}, pentru $S^{(1)}$: {0, 3}, pentru $S^{(2)}$: {0, 3}, pentru $S^{(3)}$: {0, 3, 5.1, 8.1}, pentru $S^{(4)}$: {0, 3, 5.1, 8.1, 10.2, 13.2} şi pentru $S^{(5)}$: {0, 3, 8.1, 10.2, 13.2, 15.3, 18.3}. ■

Observaţie. Pentru problema din acest exemplu, metoda partiţionării furnizează o soluţie mai simplă. Aplicând euristica partiţionării, rezultă mulţimile de valori: pentru $S^{(0)}$: {0}, pentru $S^{(1)}$: {0, 3}, pentru $S^{(2)}$: {0, 3, 4}, pentru $S^{(3)}$: {0, 3, 4, 8.1}, pentru $S^{(4)}$: {0, 3, 4, 8.1, 13.2} şi pentru $S^{(5)}$: {0, 3, 4, 8.1, 13.2, 18.3}. Această secvenţă descrie o soluţie mai simplă decât cea din exemplul precedent.

În încheierea acestei prezentări se poate pune întrebarea: Ce fel de probleme NP-dificile pot avea scheme de aproximare complet polinomiale? Răspunsul la această întrebare este dat în continuare.

Definiţia 1.2. Fie L o problemă oarecare, I mulţimea de date, $LUNGIME(I)$ numărul de biţi în reprezentarea mulţimii I şi $MAX(I)$ valoarea celui mai mare număr din I. Fără a micşora generalitatea, se poate presupune că toate numerele din I sunt întregi. Pentru un polinom ales p, fie L_p problema L restrânsă la datele din I pentru care $MAX(I) \leq p(LUNGIME(I))$. Problema L este *tare NP-dificilă* dacă şi numai dacă există un polinom p astfel încât L_p este NP-dificilă. ■

Drept exemple de probleme tare NP-dificile se pot cita: determinarea unui ciclu hamiltonian, problema comisvoiajorului, determinarea unei clici maximale etc.

Teorema 1.12. [HOR78, GAR79] Fie L o problemă de optimizare pentru care soluţiile corespunzătoare tuturor datelor au drept valoare un întreg pozitiv. Pentru toate datele din I ale lui L, se presupune că $OPT(I)$ este mărginit de o funcţie polinomială p în variabilele $LUNGIME(I)$ şi $MAX(I)$, adică au loc inegalităţile:

$$0 < OPT(I) < p(LUNGIME(I), MAX(I)).$$

https://biblioteca-digitala.ro / https://unibuc.ro

Dacă L are o schemă de aproximare complet polinomială în timp, atunci L are şi un algoritm exact de complexitate un polinom în $LUNGIME(I)$ şi $MAX(I)$. ■

Consecinţă. Teorema precedentă nu este aplicabilă pentru probleme L tare NP-dificile. Într-adevăr, în acest caz ar rezulta că pentru L_p există *algoritmi corecţi* de complexitate polinomială în $LUNGIME(I)$ şi $MAX(I)$, ceea ce contrazice faptul că L_p este NP-dificilă. Rezultă că pentru L_p, deci şi pentru L, nu există scheme de aproximare complet polinomiale în raport cu timpul.

1.1.2.6. Algoritmi buni în sens probabilistic

Există mulţi algoritmi cu performanţă nemărginită care pot rezolva *aproape întotdeauna* exact o problemă sau generează o soluţie mult îndepărtată ca valoare de cea a soluţiei optimale. Asemenea algoritmi sunt *buni în sens probabilistic.*

Pentru o problemă de mărime n, se defineşte mai întâi *spaţiul eşantion* S_n. *Spaţiul eşantion total* este produsul cartezian $S_1 \times S_2 \times \ldots \times S_n \times \ldots$. Un *element* al spaţiului eşantion total este o secvenţă $X = x_1, x_2, \ldots, x_n, \ldots$ astfel încât $x_i \in S_i$.

Definiţia 1.3. Un algoritm A rezolvă o problemă L *aproape întotdeauna* dacă, pentru fiecare eşantion X, numărul de x_i pentru care nu se rezolvă problema L este finit cu probabilitatea 1. ■

În continuare sunt prezentaţi doi algoritmi pentru probleme NP-dificile pe grafuri pentru care spaţiul eşantion se construieşte pornind de la o funcţie $p(n)$, astfel încât $0 \le p(n) \le 1$ şi $n \ge 0$. Un graf eşantion cu n noduri se construieşte incluzând fiecare muchie (i, j), $i \neq j$ cu probabilitatea $p(n)$.

Primul algoritm prezentat, elaborat de Posa [POS76], identifică un *ciclu hamiltonian* într-un graf nedirecţionat. Euristica algoritmului începe cu alegerea arbitrară a unui nod, de exemplu 1, ca nod de start. În continuare, se construieşte succesiv un drum P de lungime crescătoare, al cărui nod final se notează cu k. La fiecare iteraţie a algoritmului, se încearcă mărirea lungimii drumului P identificând o muchie (k, j) incidentă nodului final k. Pentru această muchie (k, j) au loc trei posibilităţi:

1. $j = 1$ şi P include toate nodurile grafului. În acest caz, ciclul hamiltonian a fost găsit.
2. j nu aparţine lui P. În acest caz drumului P i se adaugă muchia (k, j), j devenind nodul final al lui P.
3. j se află deja pe drumul P. Există o singură muchie $e = (j, m)$ în P astfel încât eliminarea lui e şi includerea lui (k, j) în P are drept rezultat un drum simplu. Muchia e este eliminată şi (k, j) se adaugă la P, nodul final fiind acum m.

https://biblioteca-digitala.ro / https://unibuc.ro

Se mai pune condiţia ca la pasul 3 să nu fie generate două drumuri de aceeaşi lungime având acelaşi nod final, pentru a nu se intra într-un ciclu infinit. Pentru o alegere convenabilă a reprezentării datelor, algoritmul descris are un timp de lucru de $O(n^2)$, unde n este numărul de noduri din graf.

Teorema 1.13. [POS76] Pentru $p(n) \approx (\alpha \ln n / n)$, $\alpha > 1$, algoritmul precedent identifică un ciclu hamiltonian (aproape întotdeauna). ■

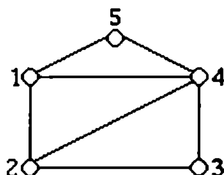


Fig. 1.5.

Exemplul 1.9. Fie graful din figura 1.5. Iniţial drumul P conţine ,odul 1. Presupunând că a fost aleasă muchia (1,4), drumul P este extins (vezi cazul 2) la {1,4}. Dacă se alege în continuare muchia (4,5), rezultă drumul {1,4,5}. Singura alegere în continuare o constituie muchia (1,5), adică are loc cazul 3 şi P devine {1,5,4}. În continuare se consideră muchiile (4,3) şi (3,2) şi se obţine drumul {1,5,4,3,2}. Considerând în continuare muchia (2,1) se obţine ciclul hamiltonian {1,5,4,3,2,1}. ■

Definiţia 1.4. O mulţime N de noduri ale grafului $G(V, E)$ se numeşte *independentă* dacă şi numai dacă nodurile din orice pereche din N nu sunt adiacente în G. O mulţime de noduri este *independent maximală* dacă este independentă şi are cel mai mare număr de noduri dintre mulţimile independente. ■

Un algoritm de tip Greedy pentru determinarea unei mulţimi independente maximale a unui graf este:

Algoritmul 1.6. ***INDEP*(*V*, *E*)**

1. [Iniţializare] Se face $N = \varnothing$.
2. [Explorare] Atâta timp cât există un nod $i \in V - N$ care nu este adiacent cu noduri din N, se face $N = N \cup \{i\}$.
3. [Terminare] Se furnizează N şi STOP. ■

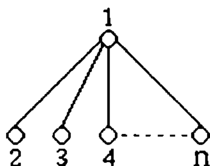


Fig. 1.6.

https://biblioteca-digitala.ro / https://unibuc.ro

Exemplul 1.10. Un exemplu pentru care algoritmul descris se dovedeşte neeficient este arătat în figura 1.6. Dacă se pleacă de la $N = \varnothing$ şi se alege $i = 1$, atunci algoritmul furnizează rezultatul $N = \{1\}$ cu toate că mulţimea independentă maximală este, în acest caz $\{2, 3, ..., n\}$. ■

Totuşi, pentru anumite distribuţii de probabilitate are loc următoarea proprietate:

Teorema 1.14. [KAR77] Dacă $p(n) = c$, pentru o anumită constantă c, atunci pentru orice $\varepsilon > 0$ are loc inegalitatea:

$$(OPT(I) - A(I)) / OPT(I) \leq 0.5 + \varepsilon. \blacksquare$$

1.1.3. Utilizarea teoriei NP-completitudinii pentru identificarea de algoritmi aproximativi

Există probleme pentru care teoria NP-completitudinii permite demonstrarea inexistenţei soluţiilor aproximative. Câteva dintre aceste rezultate sunt menţionate în teoremele ce urmează.

Teorema 1.15. [HOR78] În ipoteza P ≠ NP, problema aproximării [illegible]solute a încărcării rucsacului este NP-dificilă. ■

Un rezultat asemănător are loc în cazul problemei aproximării absolute a clicii maximale.

Teorema 1.16. [HOR78] Problema clicii maximale se reduce la problema aproximării absolute a clicii maximale. ■

Rezultate asemănătoare au loc pentru probleme de ε-aproximare.

Teorema 1.17. [HOR78] Problema ciclului hamiltonian se reduce la problema ε-aproximativă a comisvoiajorului. ■

Teorema 1.18. [HOR78] Problema partiţiei se reduce la problema ε-aproximativă a programării întregi. ■

Teorema 1.19. [HOR78] Problema ciclului hamiltonian se reduce la problema ε-aproximativă a asignării pătratice. ■

Teorema 1.20. [GAR79] Dacă P ≠ NP, atunci pentru problema aproximării absolute a unei mulţimi maximale independente nu există un algoritm aproximativ polinomial. ■

Teorema 1.21. [GAR79] Fie *MIN* o problemă de minimizare, valorile soluţiei problemei fiind numere întregi nenegative. Să presupunem că pentru un $k \in Z^+$ dat, problema: "Fiind dată o problemă individuală *I* din *MIN*, este adevărat că $OPT(I) \leq k$?" este NP-dificilă. Atunci, dacă P ≠ NP, nu există un algoritm aproximativ

https://biblioteca-digitala.ro / https://unibuc.ro

polinomial A pentru rezolvarea problemei MIN cu o eroare $R_A < 1/k$ şi problema MIN nu poate fi rezolvată cu o schemă aproximativă polinomială. ■

Teorema 1.22. [GAR79] Dacă P≠NP, atunci pentru problema colorării grafurilor nu există un algoritm aproximativ polinomial cu eroarea asimptotică $R_A^\infty < 1/3$. ■

Teorema 1.23. [GAR79] Dacă P ≠ NP, atunci pentru problema colorării grafurilor nu există un algoritm aproximativ polinomial cu eroarea asimptotică $R_A^\infty < 1$.

Demonstraţia acestei teoreme se face prin metoda *creşterii* erorii. ■

Teorema 1.24. [GAR79] Dacă P ≠ NP, atunci pentru problema comisvoiajorului nu există un algoritm aproximativ polinomial cu eroarea asimptotică $R_A^\infty < \infty$. ■

1.2. Analiza algoritmilor

Fiind dat un algoritm euristic pentru rezolvarea unei clase de probleme, identificarea unor proprietăţi formale ale acestui algoritm, numită *analiza algoritmului*, constituie un aport esenţial pentru cunoaşterea eficienţei lui. În analiza unui algoritm se determină resursele utilizate pentru executarea lui (în special memoria ocupată), timpul de execuţie (în cazul cel mai defavorabil, mediu etc.) exprimat adesea prin numărul de operaţii necesare, determinarea calităţilor soluţiilor furnizate cum ar fi eroarea faţă de soluţia optimă, determinarea unei soluţii în timp finit şi alte calităţi care vor fi definite în continuare. Analiza algoritmilor este făcută, în primul rând, pentru a furniza criterii de alegere între mai multe metode de soluţionare şi, în al doilea rând, pentru a evalua oportunitatea folosirii lor. De exemplu, obţinerea soluţiei după un număr mare de operaţii poate face algoritmul practic inaplicabil.

1.2.1. Măsuri de comportare în cel mai rău caz a unui algoritm

În acest paragraf vom studia comportarea în cel mai rău caz a unor algoritmi, prezentând exemple ilustrative.

Problema acoperirii I. Fiind dată o familie finită $F = \{S_1, S_2, ..., S_p\}$ de mulţimi finite, se cere identificarea unei subfamilii $F' \subseteq F$ astfel încât $\bigcup_{S \in F'} S = \bigcup_{S \in F} S$. ■

Cu $SC(k)$ se notează subproblema cu familii în care orice mulţime nu conţine mai mult de k elemente. Algoritmul propus, de $O(n \log n)$, este următorul.

Algoritmul 1.7. (***Acoperire***)

1. [Iniţializări] Se face $SUB = \emptyset$, $UNCOV = \bigcup_{S \in F} S$, $N = |F|$, $SET(i) = S_i$, $i = 1, 2, ..., N$.

https://biblioteca-digitala.ro / https://unibuc.ro

2. [Terminare] Dacă $UNCOV = \varnothing$, atunci furnizează *SUB* şi STOP.
3. [Alegere] Se alege $j \leq N$ asfel încât $|SET(j)|$ este maxim.
4. [Modificări] Se face $SUB = SUB \cup \{S_j\}$, $UNCOV = UNCOV - SET(j)$, $SET(i) = SET(i) - SET(j)$, pentru $i = 1, 2, ..., N$, $i \neq j$, apoi se face $SET(j) = \varnothing$.
5. [Ciclare] Se trece la pasul 2. ■

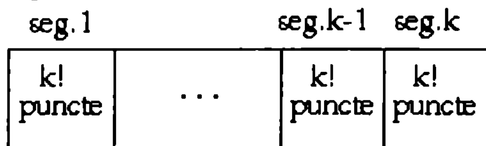


Fig. 1.7.

Exemplul 1.11. Fie problema ilustrată pe figura 1.7., în care mulţimea T ce trebuie acoperită constă din $k \bullet k!$ puncte, grupate în k segmente de câte $k!$ puncte şi numerotate de la 1 la k. Mulţimea F constă din mulţimi de câte k puncte, fiecare mulţime care conţine un punct din segmentul i conţine cel puţin câte un punct din segmentul ce urmează după i. Subacoperirea optimală F_0 constă din l mulţimi disjuncte, fiecare conţinând câte un punct din fiecare dintre cele k segmente. Subacoperirea identificată de algoritm este constituită din $k! / k$ mulţimi disjuncte cu câte k elemente care acoperă segmentul k, $k! / (k - 1)$ mulţimi, cu câte $k - 1$ elemente din segmentul $k - 1$ şi un punct din segmentul k, prin care se acoperă segmentul $k - 1$, ... şi $k!$ mulţimi ce conţin câte un punct din fiecare segment, prin care se acoperă segmentul 1. Algoritmul *Acoperire* identifică mai întâi $k! / k$ mulţimi ce acoperă segmentul k, deoarece ele au indice maxim şi pot fi alese în virtutea pasului 3. În continuare, se aleg $k! / (k - 1)$ mulţimi care acoperă segmentul $k - 1$. Acest proces continuă până când se realizează subacoperirea F_1 care conţine

$$k!(1 / k + 1 / (k-1) + ... + 1) = k!\left(\sum_{j=1}^{k}(1/j)\right)$$

mulţimi. Subacoperirea optimală F_0 cuprinde $k!$ mulţimi, de unde rezultă că $r(A, F_0) \geq \sum_{j=1}^{k}(1/j)$, unde cu $r(A, F_0)$ s-a notat măsura comportării relative a algoritmului *Acoperire* în cel mai rău caz. Se poate demonstra că suma din dreapta inegalităţii reprezintă chiar marginea superioară a comportării algoritmului C_1, adică

$$r(A, OPT(SC(k))) \leq \sum_{j=1}^{k}(1/j). \blacksquare$$

Problema colorării grafurilor. Pentru un graf $G(N, A)$ cu N noduri şi A arce, se cere determinarea unei atribuiri de culori nodurilor grafului astfel încât oricare două noduri adiacente să aibă culori diferite. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Pentru rezolvare se propune următorul algoritm:

Algoritmul 1.8. (*Colorare*)

1. [Inițializare] Se face $NECOLORAT = N$, $COLORAT(i) = \varnothing$, $i = 1, 2, ..., |N|$, $I = 1$.
2. [Terminare] Dacă $NECOLORAT=\varnothing$, furnizează $COLORAT(i)$, $i=1,2,...,I$ şi STOP.
3. [Culoare nouă] Dacă fiecare nod din *NECOLORAT* este conectat cu un nod din $COLORAT(I)$, atunci se face $I = I + 1$.
4. [Alegere] Se alege nodul x din *NECOLORAT* de grad maximal printre nodurile ce nu sunt conectate la nici-un nod din $COLORAT(I)$.
5. [Colorare] Se face $COLORAT(I) = COLORAT(I) \cup \{x\}$ şi $NECOLORAT = NECOLORAT - \{x\}$.
6. [Ciclare] Se trece la pasul 2. ■

Acest algoritm se comportă rezonabil pentru exemple de complexitate mică. Există însă şi exemple pentru care algoritmul este neeficient.

Exemplul 1.12. Fie graful $G_k = (N_k, A_k)$ cu nodurile şi arcele:

$$N_k = \{a_1, ..., a_k, b_1, ..., b_k\}, A_k = \{(a_i, b_j) \mid i \neq j\}.$$

Pentru cazul $k = 3$, graful corespunzător este arătat în figura 1.8. Acest graf este 2-colorabil, cele două culori fiind atribuite nodurilor a_i, respectiv b_i. Deoarece toate nodurile au acelaşi grad, algoritmul 1.8. le colorează în ordinea a_1, b_1, a_2, b_2, ..., a_k, b_k atribuindu-le culorile $h(a_1) = h(b_1) = 1$, $h(a_2) = h(b_2) = 2$, ..., $h(a_k) = h(b_k) = k$. Rezultă că $r(A, G_k) = k / 2$, unde $r(A, G_k)$ este raportul dintre numărul de culori furnizat de algoritmul 1.8. şi numărul optimal (= 2). ■

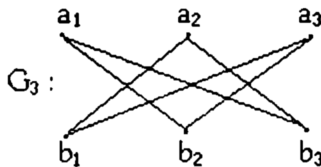


Fig. 1.8.

Un mod de ameliorare a acestui algoritm constă în alegerea altui criteriu de identificare a lui x în pasul 4. Fie *Colorare-nou* algoritmul obținut din algoritmul *Colorare* prin folosirea următoarei forme a pasului 4:

4. Fie *POS* submulțimea lui *NECOLORAT* ce cuprinde nodurile neconectate la nici-un nod din $COLORAT(I)$. Fie x nodul din *POS* conectat cu cel mai mic număr de alte noduri din *POS*.

https://biblioteca-digitala.ro / https://unibuc.ro

Exemplul 1.13. Există o secvenţă de grafuri $\{H_k\}$ 2-colorabile astfel încât H_k are 2^k noduri şi numărul culorilor dat de algoritmul *Colorare-nou* este $k + 1$. Secvenţa H_1 - H_4 este arătată în figura 1.9. Graful H_{k+1} se obţine din graful H_k prin adăugarea a 2^k noduri şi arce, pe figura 1.9. aceste noduri fiind încercuite. Algoritmul *Colorare-nou* aplicat lui H_{k+1} atribuie mai întâi o aceeaşi culoare nodurilor încercuite şi apoi trece la colorarea grafului H_k. Deoarece H_1 necesită două culori, rezultă că H_k se colorează cu $k + 1$ culori şi $r(A, H_k) = (k + 1) / 2$. ■

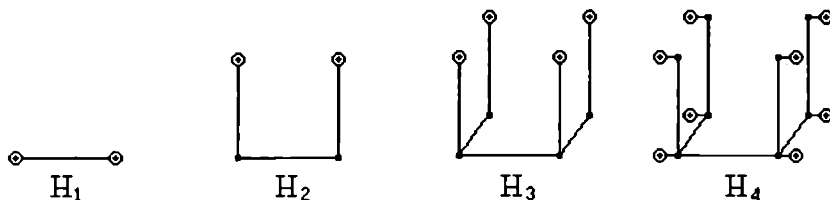


Fig. 1.9.

Problema clicii maximale. Se numeşte *clică* a unui graf neorientat $G(N, A)$ un subgraf pentru care oricare două noduri sunt adiacente. Problema *clicii maximale* constă în determinarea unei clici cu cel mai mare număr de noduri conţinută într-un graf nedirecţionat. ■

Un algoritm euristic simplu pentru rezolvarea ei este descris în continuare.

Algoritmul 1.9. (***Clică-maximală***)

1. [Iniţializări] Se face $SUB = \varnothing$, $REST = N$.
2. [Terminare] Dacă $REST = \varnothing$, atunci furnizează SUB şi STOP.
3. [Alegere] Se alege $y \in REST$ nodul conectat cu cele mai multe alte noduri din $REST$.
4. [Modificări] Se face $SUB = SUB \cup \{y\}$, $REST = REST - \{$nodurile neconectate cu $y\}$.
5. [Ciclare] Se trece la pasul 2. ■

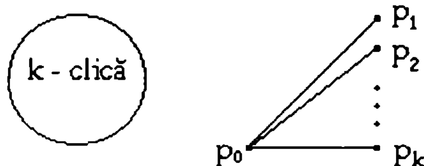


Fig. 1.10.

Exemplul 1.14. Pe figura 1.10. este ilustrat un graf cu $2k + 1$ noduri, ce conţine o clică cu k noduri, pentru care algoritmul 1.9. identifică numai clici cu două noduri. Nodul p_0 este primul identificat, deoarece este singurul conectat cu alte k noduri şi, drept consecinţă, este identificată o clică cu două noduri şi anume muchia (p_0, p_1). Rezultă că $r(A, G) = k / 2$. ■

https://biblioteca-digitala.ro / https://unibuc.ro

O altă variantă, în cadrul căreia se pleacă de la N şi se elimină succesiv noduri până ce rămâne o clică, poate uneori să aibă, de asemenea, un comportament ineficient. Algoritmul construit folosind principiul enunţat mai sus, pe care l-am numit *Clică-maximală-nou*, are forma următoare.

Algoritmul 1.10. (***Clică-maximală-nou***)

1. [Iniţializare] Se face $SUB = N$.
2. [Terminare] Dacă SUB este clică, atunci furnizează SUB şi STOP.
3. [Alegere] Se alege $y \in SUB$ un nod conectat cu cel mai mic număr de noduri din SUB.
4. [Modificare şi reluare] Se face $SUB = SUB - \{y\}$ şi se trece la pasul 2. ■

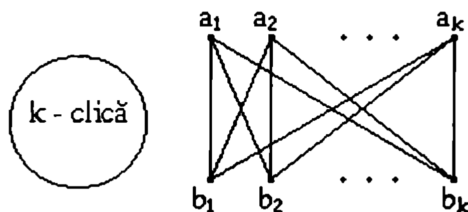


Fig. 1.11.

Exemplul 1.15. Figura 1.11. ilustrează un graf cu $3k$ noduri ce conţine o clică de mărime k şi pentru care algoritmul 1.10. dă următorul rezultat. În virtutea pasului 3, algoritmul elimină cele k vîrfuri ale clicii şi identifică în graful din dreapta o clică cu două vîrfuri. În concluzie $r(A, G) = k / 2$. ■

1.2.2. Algoritmul A^*

Un exemplu remarcabil privind problema analizei algoritmilor îl constituie algoritmul A^* elaborat de N. J. Nilsson (1982) şi prezentat conform cu J. Pearl (1984).

În cele ce urmează se presupune că pentru o problemă dată se pot specifica următoarele trei componente: *stările*, *mutările* şi *soluţiile*. Mulţimea stărilor se numeşte *spaţiul stărilor* problemei sau *spaţiul problemei*. Fiecărei probleme, pentru care sunt specificate stările şi mutările, i se poate asocia un *graf al spaţiului stărilor*, ale cărui noduri, respectiv, arce corespund stărilor, respectiv, mutărilor. Stării iniţiale, adică celei din care începe căutarea soluţiei, îi corespunde un nod numit *start* şi soluţiilor le corespund *drumuri soluţie* ce se încheie cu noduri *finale*, numite prin abuz de limbaj şi *noduri soluţie*.

În prezentarea algoritmului A^* se folosesc următoarele notaţii şi termeni:

https://biblioteca-digitala.ro / https://unibuc.ro

- s - nodul start,
- n - nod oarecare,
- n' - succesorul nodului n,
- $g(n)$ - costul drumului de la s la n,
- $h(n)$ - o estimare (evaluare) a costului drumului de la n la un nod soluţie,
- $f(n) = g(n) + h(n)$ - estimarea (evaluarea) costului unei soluţii,
- $c(n, n')$ - costul arcului de la n la n',
- *generarea unui nod* - calculul codului nodului pe baza codului nodului *tată* despre care se spune că a fost *explorat*,
- *expandarea unui nod* - generarea tuturor succesorilor direcţi ai unui nod tată,
- *DESCHIS* - mulţimea nodurilor generate ce urmează a fi expandate,
- *ÎNCHIS* - mulţimea nodurilor expandate (adică ai căror succesori sunt accesibili procedurii de căutare a soluţiei).

Ideea de bază a algoritmului A^* constă în folosirea unei *funcţii euristice de evaluare* $f(n)$ în vederea identificării nodului n din graful spaţiului stărilor ce urmează a fi expandat în vederea continuării identificării unui drum soluţie. Se presupune că nodul ales pentru expandare este unul pentru care $f(\bullet)$ are o valoare minimă şi, dacă există două drumuri orientate către acest nod, cel cu o valoare mai mare a lui $f(\bullet)$ este eliminat. Algoritmul A^* face parte dintr-o familie de algoritmi numiţi în lim' engleză *best-first* şi este aplicabil numai grafurilor SAU. În procesul de căutare a soluţiei are loc uneori *reluarea* parcurgerii unor ramuri ale arborelui de traversare T (vezi mai jos) generat de algoritm. Mai precis, dacă la o anumită etapă un drum în T s-a dovedit puţin promiţător în ceea ce priveşte eficienţa soluţiei, parcurgerea lui este întreruptă şi se trece la generarea şi parcurgerea altui drum. Dacă se constată că parcurgerea acestui nou drum este inutilă, are loc reluarea parcurgerii drumului anterior. Această strategie se numeşte *inspectarea grafului* (*graph-search*).

Algoritmul 1.11. **(A^*)**

1. [Iniţializare] Se include nodul start s în *DESCHIS*.
2. [Terminare] Dacă *DESCHIS* este vidă, atunci STOP.
3. [Alegere] Se elimină din *DESCHIS* şi se include în *ÎNCHIS* un nod n pentru care valoarea funcţiei f este minimă.
4. [Soluţie] Dacă n este nod soluţie, atunci se furnizează soluţia obţinută a listă a pointerilor de la n la nodul iniţial s.

https://biblioteca-digitala.ro / https://unibuc.ro

5. [Nod care nu e soluție] Altfel se parcurge nodul n generând toți succesorii săi și se asociază pointeri de la aceștia către n. Pentru fiecare succesor n' al lui n:
   a) Dacă n' nu este în *DESCHIS* sau *ÎNCHIS*, atunci se estimează $h(n')$ (o estimare a costului celui mai bun drum de la n' la un nod soluție) și se calculează $f(n') = g(n') + h(n')$, unde $g(n') = g(n) + c(n, n')$ și $g(s) = 0$.
   b) Dacă n' este deja în *DESCHIS* sau *ÎNCHIS*, atunci se orientează pointerii săi dealungul drumului ce furnizează un cel mai mic $g(n')$.
   c) Dacă n' necesită o reajustare a pointerului și este în *ÎNCHIS*, atunci se introduce n' în *DESCHIS*.
6. [Ciclare] Se trece la pasul 2. ■

1.2.2.1. Definiții și notații suplimentare

În continuare se presupune, dacă nu este precizat altfel, că spațiul căutării soluțiilor este un graf SAU, identificarea unei soluții constând în găsirea unui drum între nodul s și un nod soluție. De asemenea, se presupune că estimarea euristică $h(n)$ este calculabilă în orice nod al grafului. În locul formulării *nodul* n_j *este succesorul nodului* n_i se folosește notația $n_j \in SCS(n_i)$, SCS fiind *operatorul succesiune.* Se presupune că grafurile au un singur nod start s. Mulțimea nodurilor soluție este notată cu Γ și elementele ei cu γ. Orice arc are un cost $c(n_i, n_j) \geq \delta > 0$, cu δ dat, și costul unui drum este egal cu suma costurilor arcelor componente. Dacă un nod n_j este accesibil dintr-un nod n_i, atunci mulțimea tuturor drumurilor de la n_i la n_j este notată cu $P_{n_i - n_j}$ și cu $P_{n\text{-}\gamma}$ se notează orice drum de la n la γ, adică $F \quad \in P_{n\text{-}\Gamma}$. Mulțimea celor mai ieftine drumuri de la n_i la n_j este notată cu $P^*_{n_i - n_j}$ și costul celui mai ieftin drum, cu $k(n_i, n_j)$.

Costurilor drumurilor optimale conținând pe s sau un element din Γ li se atribuie următoarele nume:

- $f^*(n)$ - costul celui mai ieftin drum soluție ce trece prin nodul n,
- $g^*(n)$ - costul celui mai ieftin drum de la s la un nod n, adică:

$$g^*(n) = k(s, n),$$

- $h^*(n)$ - costul celui mai ieftin drum de la n la Γ, adică:

$$h^*(n) = \min_{\gamma \in \Gamma} k(n, \gamma),$$

- C^* - cel mai ieftin cost al drumului de la s la Γ, adică:

$$C^* = h^*(s).$$

https://biblioteca-digitala.ro / https://unibuc.ro

Cu Γ^* este notată mulţimea nodurilor *soluţii optimale*, adică submulţimea nodurilor accesibile din s pe o cale sau mai multe căi de cost C^*. Această mulţime poate eventual să fie vidă sau să conţină mai multe elemente, dar în cele mai multe cazuri ea conţine un singur element care ar trebui determinat.

Algoritmul A^* explorează graful G al spaţiului stărilor utilizând un arbore de traversare T. La fiecare etapă a căutării, arborele T este definit de pointerii pe care A^* îi atribuie nodurilor generate, ramificaţiile din T având sens opus pointerilor. Ori de câte ori este necesară evidenţierea faptului că un drum dat face parte din T, la o anumită etapă a căutării, acest drum este notat cu PP, de exemplu $PP_{n_1-n_2} \in T$, indicii fiind nodurile între care se consideră drumul dat. De asemenea, se face distincţie între u. nod aflat *dealungul* unui drum P, dacă P este în T, de un nod situat pe P, dacă P nu este în T.

Uniunea tuturor ramificaţiilor din arborii de traversare construiţi în timpul căutării reprezintă subgraful *explorat* al lui G şi este notat G_e. De notat că un drum P poate aparţine lui G_e dar să nu aparţină nici unui arbore de traversare din G_e.

În continuare sunt studiate câteva proprietăţi de bază ale algoritmului A^*.

Completitudine. Un algoritm este *complet* dacă se încheie cu o soluţie, atunci când există cel puţin una.

Admisibilitate. Un algoritm este *admisibil* atunci când garantează furnizarea unei soluţii *optimale*, atunci când există o astfel de soluţie.

Dominanţă. Un algoritm A_1 *domină* algoritmul A_2 dacă orice nod expandat de A_1 este de asemenea expandat de A_2. Algoritmul A_1 *domină strict* pe A_2 dacă A_1 domină pe A_2 dar A_2 nu domină pe A_1. Un sinonim al termenului *domină* îl constituie expresia: *mai eficient decât*. Aceasta deoarece, în general, A_1 e andează un număr mai mic de noduri decât A_2, deci se obţin costuri mai mici pentru A_1.

Optimalitate. Un algoritm se numeşte *optimal* într-o clasă de algoritmi dacă el domină toate elementele acestei clase.

1.2.2.2. Căutarea unei soluţii optimale cu algoritmul A^*

1.2.2.2.1. *Proprietăţile funcţiei f^**

Funcţia f de selectare (evaluare) a nodurilor utilizată în A^* este suma a două funcţii $g(n)$ şi $h(n)$, unde:

- $g(n)$ - este costul drumului *curent* PP_{s-n} de la s la n, cu $g(s) = 0$, s fiind nodul start,
- $h(n)$ - este o estimare a lui $h^*(n)$, asfel încât $h(\gamma) = 0$, γ fiind nod soluţie.

https://biblioteca-digitala.ro / https://unibuc.ro

Pentru specificarea costurilor *actuale* P_{s-n} şi $P_{n-\gamma}$ dealungul unui drum P, se foloseşte notaţia $g_P(n)$ respectiv $h_P(n)$. Evident că pentru orice drum P au loc:

$$g_P(n) \geq g^*(n) \text{ şi } h_P(n) \geq h^*(n). \quad (1.1)$$

Atunci când g şi h coincid cu valorile lor optimale, suma lor:

$$f^*(n) = g^*(n) + h^*(n)$$

măsoară *costul optimal pe toate drumurile soluţie ce trec prin n*. În particular, pentru nodul iniţial s şi orice nod soluţie $\gamma \in \Gamma^*$ au loc relaţiile:

$$f^*(s) = h^*(s) = g^*(\gamma) = f^*(\gamma) = C^*$$

şi, în plus, dacă $n^* \in P^*_{s-\Gamma}$ este un nod oarecare de pe un drum optimal, atunci este satisfăcută relaţia:

$$f^*(n^*) = C^*. \quad (1.2)$$

Pentru demonstrarea validităţii relaţiei (1.2), se observă că $n^* \in P^*_{s-\Gamma}$ implică existenţa unui drum soluţie P trecând prin n şi având costul C^*. Deci pe acest drum $g_P(n) + h_P(n) = C^*$. În virtutea optimalităţii lui g^* şi h^* din (1.1), rezultă că $g^*(n) + h^*(n) \leq C^*$, dar această inegalitate nu poate fi strictă. În caz contrar ar exista un drum P' pentru care $g_{P'}(n) + h_{P'}(n) < C^*$, ceea ce ar contrazice optimalitatea lui C^*.

Importanţa funcţiei $f^*(n)$ rezultă din comportarea ei în noduri ce nu aparţin unui drum soluţie optimal. Pentru un asemenea nod n trebuie satisfăcută condiţia:

$$f^*(n) > C^*, n \notin P^*_{s-\Gamma}. \quad (1.3)$$

Într-adevăr, presupunând că $g^*(n) + h^*(n) \leq C^*$, ar rezulta că există un drum soluţie P' trecând prin n pentru care $g_{P'}(n) + h_{P'}(n) \leq C^*$, deci P' ar fi o soluţie optimală, ceea ce contrazice ipoteza că n nu aparţine unui drum soluţie optimal.

1.2.2.2.2. *Terminare şi completitudine*

Algoritmul A^* se termină întotdeauna pentru grafuri finite. Afirmaţia rezultă din faptul că numărul de drumuri aciclice în asemenea grafuri este finit şi algoritmul A^* adaugă noi legături în arborele de traversare al grafului la orice expandare a unui nod. Orice nouă legătură reprezintă un nou drum aciclic astfel încât până la urmă mulţimea de drumuri este epuizată.

Algoritmul A^* este de asemenea complet pe grafuri finite. Un eşec are loc numai atunci când mulţimea *DESCHIS* este găsită vidă şi dacă există un drum soluţie $P_{s-\gamma}$, atunci mulţimea *DESCHIS* nu poate fi vidă înainte ca $P_{s-\gamma}$ să fie depistată. Altfel ar exista măcar un nod $n' \in P_{s-\gamma}$ în *DESCHIS* care este expandat fără a genera un nou succesor. Aceasta contrazice presupunerea că n' aparţine drumului soluţie. Orice astfel de nod, în afară de nodul soluţie, are cel puţin un succesor aflat pe drumul soluţie.

https://biblioteca-digitala.ro / https://unibuc.ro

1.2.2.2.3. *Admisibilitate - garanţia pentru o soluţie optimală*

Definiţia 1.5. O funcţie euristică h este *admisibilă* dacă, pentru orice nod n:

$$h(n) \leq h^*(n). \blacksquare \quad (1.4)$$

În cele ce urmează, se presupune că relaţia (1.4) este satisfăcută.

Lema 1.1. La orice moment, înainte ca A^* să se încheie, există un nod n' în *DESCHIS* situat pe $P^*_{s-\gamma}$ cu $f(n') \leq C^*$.

Demonstraţie. Fie un drum optimal $P^*_{s-\gamma} \in P^*_{s-\Gamma}$:

$$P^*_{s-\gamma} = s, n_1, n_2, ..., n', ..., \gamma,$$

unde n' este primul nod de pe $P^*_{s-\Gamma}$ aparţinând mulţimii *DESCHIS*. Există cel puţin un nod din *DESCHIS* situat pe $P^*_{s-\Gamma}$ deoarece γ nu este în *ÎNCHIS* înainte de încheiere. Deoarece toţi predecesorii lui n' sunt în *ÎNCHIS* şi drumul $s, n_1, n_2, ..., n'$ este optimal, rezultă că pointerii atribuiţi lui n' sunt dealungul drumului $P^*_{s-n'}$, deci $g(n') = g^*(n')$. Din admisibilitatea lui h se deduce inegalitatea:

$$f(n') = g^*(n') + h(n') \leq g^*(n') + h^*(n') = f^*(n')$$

şi, deoarece $n' \in P^*_{s-\gamma}$, rezultă că (vezi (1.2)) $f^*(n') = C^*$, deci $f(n') \leq C^*$. ■

Lema 1.2. (Corolar al lemei 1.1.) Fie n' primul nod de pe drumul optimal $P^*_{s-n''}$ către nodul n'', nu neapărat din Γ, aparţinând mulţimii *DESCHIS*. Atunci $g(n') = g^*(n')$, adică A^* a identificat deja un drum optimal de pointeri către n' (deci n' aparţine lui $P^*_{s-n''}$) şi acest drum rămâne neschimbat în procesul căutării.

Demonstraţie. Afirmaţia rezultă direct din lema 1.1., unde egalitatea $g(n') = g^*(n')$ a fost dedusă fără să se ţină seama de faptul că γ este nod soluţie. ■

Teorema 1.25. A^* este admisibil.

Demonstraţie. Să presupunem că A^* se încheie cu un nod soluţie $t \in \Gamma$ pentru care $f(t) = g(t) > C^*$. Algoritmul A^* inspectează satisfacerea de către noduri a criteriului de terminare numai după ce au fost selectate pentru expandare. Rezultă că, dacă nodul t a fost ales pentru expandare, este satisfăcută condiţia:

$$f(t) \leq f(n), \text{ pentru orice } n \in DESCHIS.$$

De aici se deduce că, imediat înainte de încheiere, pentru toate nodurile din *DESCHIS* are loc inegalitatea $f(n) > C^*$, în contradicţie cu lema 1.1. care garantează existenţa măcar a unui nod n' cu $f(n') \leq C^*$. Prin urmare, pentru nodul terminal t, are loc relaţia $g(t) = C^*$, cu semnificaţia că A^* furnizează un drum optimal. ■

https://biblioteca-digitala.ro / https://unibuc.ro

1.2.2.2.4. *Comportarea puterii de selectare a euristicilor*

Puterea estimării $h(\bullet)$ a unei euristici este definită de valoarea selecției induse de h şi depinde evident de *acuratețea* acestei estimări. Dacă $h(\bullet)$ estimează costul căutării în mod exact (adică $h = h^*$), atunci A^* explorează numai noduri situate pe drumuri optimale. Pe de altă parte, dacă nu este folosită nici o euristică ($h = 0$), rezultă o căutare în lățime, drept rezultat având loc expandarea tuturor nodurilor accesibile din s pe un drum de cost mai mic decât C^*. Cazurile cele mai des întâlnite sunt situate între aceste două extreme. Este evidentă proprietatea următoare: cu cât h este mai apropiat de h^*, cu condiția să rămână admisibilă, cu atât este mai eficientă căutarea soluției.

Definiția 1.6. O euristică h_2 este *euristică mai informată* decât h_1 dacă ambele sunt admisibile şi $h_2(n) > h_1(n)$, pentru orice nod n nonsoluție. Similar, un algoritm A^* ce utilizează h_2 se consideră mai informat decât altul ce utilizează h_1. ∎

Teorema 1.26. (Condiție necesară pentru expandarea unui nod) Orice nod expandat de A^* nu poate avea o valoare a lui f mai mare ca C^*, adică $f(n) \leq C^*$ pentru toate nodurile n expandate.

Demonstrație. Să presupunem contrariul, adică n este selectat din *DESCHIS* dacă $f(n) > C^*$. Această ipoteză contrazice lema 1.1. în virtutea căreia mulțimea *DESCHIS* conține un nod n' astfel încât $f(n') \leq C^*$. Deci nodul n' va fi ales pentru expandare în locul lui n. ∎

Teorema 1.27. (Condiție suficientă pentru expandarea unui nod) Orice nod din *DESCHIS*, pentru care $f(n) < C^*$, va fi până la urmă expandat de A^*.

Demonstrație. Să presupunem că la o anumită etapă, nodul n cu $f(n$ C^* este găsit în *DESCHIS*. Algoritmul A^* se încheie cu $f(t) = C^*$ după ce l-a selectat pe t pentru expandare din *DESCHIS*. Faptul că nodul t şi nu n a fost selectat, înseamnă că, ori n a fost selectat înaintea lui t, ori valoarea $f(n)$ a fost între timp modificată astfel încât depăşeşte pe C^*. Dar valoarea lui f nu poate fi modificată decât prin deplasarea în adâncime, astfel încât numai prima alternativă, adică expandarea lui n, este singura posibilitate. ∎

De notat că nici una din teoremele 1.26. sau 1.27. nu conține o condiție necesară şi suficientă, deoarece soarta nodurilor din *DESCHIS* pentru care are loc egalitatea $f(n) = C^*$ nu este menționată de nici una dintre teoreme şi rămâne să fie decisă de regulile *tie-breaking* utilizate în implementarea algoritmului A^*. O regulă *tie-breaking* este un criteriu de alegere a uneia dintre mai mult de două alternative, echivalente din punct de vedere al continuării căutării soluției.

https://biblioteca-digitala.ro / https://unibuc.ro

Condiţiile de expandare din teoremele 1.26. şi 1.27., chiar dacă sunt utile, nu furnizează criterii adecvate pentru determinarea eficienţei unui algoritm A^*. Aceasta se datoreşte faptului că în aceste teoreme sunt invocaţi doi parametri locali, care pot varia o dată cu etapa de evoluţie a algoritmului şi nu sunt, prin urmare, detectabili direct din datele externe. Primul este funcţia $g(n)$, care nu este în realitate o proprietate a nodului n, dar depinde de drumul către n depistat de A^* la un moment dat. Al doilea este condiţia din teorema 1.27. că n se află în *DESCHIS* înainte de a fi selectat pentru expandare. Faptul că un nod dat n este în *DESCHIS* depinde nu de n însuşi, ci de comportarea lui A^* în procesul de explorare a drumurilor care duc la n. Următoarele două rezultate rectifică aceste dificultăţi.

Definiţia 1.7. Un drum P este *C-mărginit* dacă orice nod n dealungul acestui drum satisface condiţia $g_P(n) + h(n) \leq C$. Similar, dacă în relaţia precedentă are loc inegalitatea strictă pentru orice n de pe P, atunci P este *strict C-mărginit*. Dacă este necesară menţionarea euristicii folosită pentru afirmarea inegalităţii precedente, se foloseşte notaţia *C*(h)*-mărginit*. ■

Teorema 1.28. O condiţie suficientă pentru A^* în vederea expandării unui nod n este existenţa unui drum P strict C^*-mărginit de la s la n.

Demonstraţie. Presupunem contrariul, adică existenţa unui drum P strict C^*-mărginit de la s la n şi că, la încheierea căutării, nodul său final n nu a fost încă expandat. Fie n' primul nod din *DESCHIS* de pe P (la încheierea căutării). Deoarece toţi predecesorii lui n' sunt în *ÎNCHIS*, valoarea lui g pe care A^* o atribuie lui n' nu poate fi mai mare decât $g_P(n')$; deci:

$$f(n') = g(n') + h(n') \leq g_P(n') + h(n') < C^*.$$

Dar, dacă $f(n') < C$ la încheierea căutării, atunci n' trebuie să fi fost ales pentru expandare în locul nodului soluţie, care este candidat a încheia căutarea cu $f = C^*$. Acest fapt contrazice ipoteza că un nod n' din *DESCHIS* există pe P şi, prin urmare, n trebuie expandat. ■

Teorema 1.29. O condiţie necesară pentru ca A^* să expandeze un nod n este existenţa unui drum C^*-mărginit de la s la n.

Demonstraţie. Dacă A^* expandează un nod n, atunci n trebuie să fie în *DESCHIS* şi, de asemenea, $f(n) \leq C^*$ (teorema 1.26.). Fie acum drumul PP al pointerilor atribuiţi lui n la momentul expandării lui. Fiecare predecesor n' al lui n dealungul lui PP este în *ÎNCHIS* şi, prin urmare, a fost ales pentru expandare, poate mai mult decât o dată, la un moment precedent, moment la care, cum rezultă din teorema 1.26., este satisfăcută inegalitatea $g'(n') + h(n') \leq C^*$. Valoarea curentă a lui

https://biblioteca-digitala.ro / https://unibuc.ro

$g(n')$ dealungul lui PP nu poate fi mai mare decât valoarea lui $g'(n')$ în momentul expandării. Prin urmare, fiecare nod n' de pe PP satisface condiția $f(n') \le C^*$, cu semnificația că PP este şi el C^*-mărginit. ■

Teoremele 1.28. şi 1.29. joacă un rol major în analiza performanţelor lui A^*. Teorema următoare confirmă intuiţia noastră privind tendinţa de a alege pe h, posibil, cât mai ridicat.

Teorema 1.30. Dacă A_2^* este mai informat decât A_1^*, atunci A_2^* domină pe A_1^*.

Demonstraţie. Fie h_2 şi h_1 euristicile utilizate de A_2^*, respectiv, A_1^* astfel încât $h_2(n) > h_1(n)$ pentru orice nod n care nu este soluţie şi să presupunem că A_2^* expandează un nod oarecare n. Din teorema 1.29. se ştie că există un drum C^*-mărginit de la s la n ales cu ajutorul lui h_2. Deoarece $h_2(n') > h_1(n')$, pentru orice nod n' de pe P, rezultă că P este strict mărginit dacă a fost ales de h_1. ■

1.2.2.2.5. *Euristici monotone (consistente)*

În secţiunea precedentă s-a presupus, implicit, că eficienţa lui A^* poate fi măsurată cu numărul de noduri neexpandate. În realitate, un nod poate fi expandat de mai multe ori şi, prin urmare, nu numărul de noduri expandate ci numărul de *operaţii de expandare* trebuie avut în vedere la aprecierea eficienţei. În continuare se arată că, în anumite condiţii, algoritmul A^* nu parcurge, în vederea introducerii în *DESCHIS*, unele noduri din *ÎNCHIS* şi, prin urmare, timpul necesar reexpandării poate fi în acest fel micşorat.

Condiţia care permite evitarea redeschiderilor este o anumită proprietate a lui h ce este satisfăcută de orice funcţie h^*, adică o *consistenţă*. În secţiunile precedente s-a văzut că orice funcţie h^*, în afară de estimarea corectă a costurilor optimale către Γ, satisface anumite relaţii între $h^*(n)$ şi evaluările lui h^* ale descendenţilor lui n. Relaţiile (1.2) şi (1.3) implică faptul că cel mai ieftin drum constrâns să treacă prin n nu poate fi mai puţin costisitor decât cel mai ieftin drum generat fără această restricţie, adică $g^*(n) + h^*(n) \ge h^*(s)$, pentru orice n, sau $k(s, n) + h^*(n) \ge h^*(s)$, unde $k(s, n)$ este costul celui mai ieftin drum de la s la n.

Această formă a *inegalităţii triunghiului* pentru geodezice este valabilă pentru orice pereche de noduri, nu neapărat conţinând pe s. Aşadar, dacă n' este un descendent al lui n, are loc relaţia:

$$h^*(n) \le k(n, n') + h^*(n'), \text{ pentru orice } (n, n'),$$

care este automat satisfăcută şi pentru nodurile n' pentru care $k(n, n') = \infty$ şi care nu sunt descendente ale lui n.

https://biblioteca-digitala.ro / https://unibuc.ro

Rezultă că este rezonabil să se considere că, dacă procesul de estimare a lui $h(n)$ este consistent, el trebuie să moştenească proprietatea geodezică a lui $h^*(n)$ şi să satisfacă condiţia:

$$h(n) \leq k(n, n') + h(n'), \text{ pentru orice } (n, n'). \tag{1.5}$$

Definiţia 1.8. O funcţie euristică $h(n)$ este *consistentă* dacă relaţia (1.5) este satisfăcută pentru orice pereche de noduri n şi n'. ■

Definiţia 1.9. O funcţie euristică $h(n)$ este *monotonă* dacă satisface relaţia:

$$h(n) \leq c(n, n') + h(n'), \text{ pentru orice } (n, n'), n' \in SCS(n), \tag{1.6}$$

unde $c(n, n')$ este costul arcului de la n la n'. ■

Monotonia pare, la prima vedere, mai puţin restrictivă decât consistenţa, deoarece ea se referă la două noduri succesive. Se poate însă arăta, prin inducţie după adâncimea k a descendenţilor lui n, că (1.6) implică (1.5). Prin urmare are loc următoarea teoremă.

Teorema 1.31. Monotonia şi consistenţa sunt proprietăţi echivalente. ■

Se poate de asemenea deduce legătura dintre consistenţă şi admisibili' 'e.

Teorema 1.32. Orice euristică consistentă este şi admisibilă.

Demonstraţie. Înlocuid pe n' din (1.5) cu orice nod soluţie $\gamma \in \Gamma$, se obţine:

$$h(n) \leq k(n, \gamma) + h(\gamma), \text{ pentru orice } n.$$

Deoarece $h(\gamma) = 0$ şi $k(n, \gamma) = h^*(n)$ pentru orice nod $\gamma \in \Gamma^*$, rezultă că are loc condiţia de admisibilitate (vezi (1.4)):

$$h(n) \leq h^*(n). \ ■$$

Vom demonstra avantajele utilizării euristicilor consistente sau monotone.

Teorema 1.33. Un algoritm A^* ce utilizează o euristică monotonă găseşte drumurile optimale pentru toate nodurile expandate, adică:

$$g(n) = g^*(n), \text{ pentru orice } n \in \hat{I}NCHIS.$$

Demonstraţie. Să presupunem că A^* alege pentru expandare un nod pentru care $g(n) > g^*(n)$. Fie un drum optimal P^*_{s-n} de la s la n. Dacă n este singurul nod din *DESCHIS* de pe drumul P^*_{s-n}, atunci este evident că toţi predecesorii lui n pe P^*_{s-n} au fost expandaţi şi, prin urmare, $g(n) = g^*(n)$ (Lema 1.2.). Presupunerea $g(n) > g^*(n)$ implică faptul că P^*_{s-n} conţine măcar un nod adiţional în *DESCHIS* şi, ca de obicei, fie n' primul nod din *DESCHIS* de pe P^*_{s-n}. Se poate arăta că n' şi nu n trebuie ales pentru expandare. Lema 1.2. menţionează că $g(n') = g^*(n')$ şi, prin urmare, în virtutea consistenţei, rezultă că:

$$f(n') = g^*(n') + h(n') \leq g^*(n') + k(n', n) + h(n),$$

https://biblioteca-digitala.ro / https://unibuc.ro

unde $k(n', n)$ este costul celui mai ieftin drum de la n' la n. Suma $g^*(n') + k(n', n)$ este egală cu $g^*(n)$ deoarece n' este un predecesor al lui n dealungul lui P^*_{s-n} şi, de aceea,

$$f(n') \leq g^*(n) + h(n).$$

Rezultă că ipoteza $g(n) > g^*(n)$ implică:

$$f(n') < f(n),$$

fapt care împiedică alegerea lui n în loc de n'. Prin urmare, rezultă că $g(n) = g^*(n)$, ceea ce completează demonstraţia teoremei. ■

Teorema 1.34. Monotonia implică faptul că secvenţa valorilor lui f corespunzătoare nodurilor expandate de A^* este nedescrescătoare.

Demonstraţie. Fie n_2 nodul expandat imediat după n_1. Dacă n_2 este în *DESCHIS*, atunci, când n_1 a fost expandat, din regula de selectare a nodurilor rezultă că $f(n_1) \leq f(n_2)$. În caz contrar, n_2 trebuie să fie noul venit în *DESCHIS*, adică un succesor al lui n_1 pentru care:

$$g(n_2) = g(n_1) + c(n_1, n_2)$$

şi pentru care, în virtutea monotoniei:

$$f(n_2) = g(n_1) + c(n_1, n_2) + h(n_2) \geq g(n_1) + h(n_1) = f(n_1).$$

Rezultă că secvenţa valorilor lui f nu poate fi descrescătoare. ■

Monotonia simplifică semnificativ condiţiile de expandare a nodurilor.

Teorema 1.35. Dacă f este monotonă, atunci condiţia necesară pentru expandare este:

$$g^*(n) + h(n) \leq C^*$$

şi condiţia suficientă este definită de inegalitatea strictă:

$$g^*(n) + h(n) < C^*.$$

Demonstraţie. Condiţia necesară rezultă din teoremele 1.26. şi 1.31. Condiţia suficientă se bazează pe natura nedescrescătoare a lui $g^* + h$ dealungul unui drum optimal de la s la un nod arbitrar n. Afirmaţia este adevărată, deoarece, dacă n' este un predecesor al lui n dealungul unui drum optimal P^*_{s-n}, el trebuie să satisfacă relaţia:

$$g^*(n) = g^*(n') + c(n', n),$$

care, împreună cu monotonia lui h, implică:

$$g^*(n) + h(n) = g^*(n') + c(n', n) + h(n) \geq g^*(n') + h(n').$$

Rezultă că dacă n satisface condiţia:

$$g^*(n) + h(n) < C^*,$$

atunci şi toţi predecesorii lui de pe P^*_{s-n} trebuie să o satisfacă. Prin urmare P^*_{s-n} este strict C^*-mărginit şi, în virtutea teoremei 1.28., n va fi expandat de A^*. ■

https://biblioteca-digitala.ro / https://unibuc.ro

1.2.2.3. Relaxarea condiţiilor de optimalitate

Rezultatele precedente arată că A^* posedă unele proprietăţi utile privind folosirea euristicilor alese. În particular, dacă euristica utilizată este admisibilă, atunci A^* garantează obţinerea unui drum optimal.

Experienţa arată, totuşi, că în multe cazuri A^* utilizează un timp mare în procesul de discriminare a drumurilor a căror cost nu variază mult de la unul la altul. În acest caz, proprietatea de admisibilitate devine un cusur în loc de o virtute. Ea forţează pe A^* să consume un timp disproporţionat de mare în selectarea celui mai bun dintre candidaţii aproape egali şi împiedică pe A^* să încheie căutarea unui drum suboptimal, dar, pe de altă parte, acceptabil.

În continuare se prezintă două forme de reprezentare a lui f ca funcţie de g şi h şi efectele lor asupra eficienţei algoritmului A^*.

1.2.2.3.1. *Ajustarea ponderilor lui* g *şi* h

În vederea obţinerii unor versiuni mai rapide ale algoritmului A^*, care nu asigură neapărat generarea unor soluţii optimale, este indicat să se examineze separat efectele lui g şi h.

În ipoteza că $f(n) = g(n)$, strategia căutării coincide cu cea a *costului uniform*, care este o variantă a căutării în lăţime. În această variantă, în loc să se parcurgă nivele de adâncime egală, sunt parcurse nivele de cost egal. Procedura *cost uniform* se realizează pe calea selectării pentru expandare a unui nod din *DESCHIS* de cost minimal, unde costul fiecărui nod este definit ca fiind costul drumului de la nodul start la acest nod.

În ipoteza că $f(n) = h(n)$, strategia căutării coincide cu cea a căutării în *lăţime* (*BF*). În acest caz, dacă alegerea lui $h(n)$ nu este suficient de consistentă, adică nu reflectă suficient de bine particularităţile problemei studiate, atunci este posibilă atribuirea de valori mici ale lui $h(n)$ unor drumuri prea lungi, drumuri care, datorită lipsei criteriului furnizat de g, nu pot fi evitate.

În vederea echilibrării particularităţilor menţionate, se poate utiliza o funcţie de evaluare ponderată:

$$f_w(n) = (1 - w)\, g(n) + w\, h(n),$$

care, pentru w = 0, 1/2, 1 corespunde strategiilor cost uniform, A^* şi *BF*. Prin variaţia continuă a lui w între 0 şi 1 se pot obţine diferite variante ale algoritmului A^*, în vederea rezolvării convenabile a unor probleme.

https://biblioteca-digitala.ro / https://unibuc.ro

Rezultatele empirice obţinute pentru *jocul 15* (15 plăcuţe pe un tabel 4 × 4 care trebuie aduse dintr-o stare iniţială într-o stare finală prin mutări admise a câte unei plăcuţe pe locul liber dacă acesta este vecin ei) arată că f_w asigură o performanţă optimală (număr mediu minim de noduri expandate) în banda $1/2 < w < 1$. Simulări mai extensive pentru *jocul 8* (8 plăcuţe pe un tabel 3 × 3) au arătat că f_w atinge o performanţă optimală în vecinătatea lui $w = 1/2$ şi creşterea lui w dincolo de 1/2 poate provoca o creştere a numărului de noduri expandate.

1.2.2.3.2. *Două versiuni ε- admisibile mai rapide ale lui* A*

Deteriorarea performanţei lui $f_w = (1 - w)\, g + w\, h$ pentru $w > 1/2$, cuplată cu riscul permanent de încheiere cu o soluţie de cost mai mare decât cel acceptabil, a determinat încercările de a răspunde la întrebarea: poate fi asigurată o creştere a vitezei cu preţul unei descreşteri *mărginite* a calităţii soluţiei? În continuare sunt prezentate două abordări ale acestei probleme: *ponderarea dinamică* şi A^*_ε. Un algoritm este considerat *ε-admisibil* dacă furnizează un rezultat diferit de cel optimal prin factorul $1 + \varepsilon$.

1.2.2.3.2.1. Ponderarea dinamică

În locul menţinerii lui w constant dealungul căutării, este natural să se varieze dinamic valoarea lui astfel încât valoarea lui h să fie ponderată mai mult atunci când adâncimea căutării creşte. Pentru aceasta se poate utiliza funcţia de evaluare:

$$f(n) = g(n) + h(n) + \varepsilon\,(1 - d(n) / N)\, h(n)$$

unde $d(n)$ este adâncimea nodului n şi N este adâncimea (anticipată) a nodului soluţie dorit. La nivele mici ale arborelui de căutare ($d \ll N$), h are o pondere de ordinul $1 + \varepsilon$, fapt care încurajează căutarea în adâncime. La nivele adânci, atunci când terminarea se apropie, căutarea presupune o ponderare egală a lui g şi h, fapt care evită o încheiere prematură.

Nu este greu de văzut că dacă $h(n)$ este admisibilă, algoritmul este ε-admisibil, adică se găseşte un drum de cost cel mult $(1 + \varepsilon)\, C^*$. Afirmaţia se bazează pe faptul că înaintea terminării, pentru primul nod n' din *DESCHIS* de pe orice soluţie optimală are loc egalitatea $g(n') = g^*(n')$ (vezi lema 1.2.) şi prin urmare:

$$f(n') \le g^*(n') + h^*(n') + \varepsilon\,(1 - d(n') / N)\, h^*(n') \le C^* + \varepsilon\, h^*(n') \le (1 + \varepsilon)\, C^*.$$

Rezultă că algoritmul nu se poate încheia cu un nod soluţie al cărui cost este mai mare decât $(1 + \varepsilon)\, C^*$.

https://biblioteca-digitala.ro / https://unibuc.ro

1.2.2.3.2.2. A_ε^* - algoritm ce utilizează estimări ale efortului căutării

Algoritmul A_ε^* utilizează două liste: *DESCHIS* şi *FOCAR*. *FOCAR* este o sublistă a lui *DESCHIS* ce conţine numai acele noduri pentru care f nu deviază faţă de valoarea cea mai mică a lui f printr-un factor mai mare decât 1 + ε, adică:

$$FOCAR = \{n \mid f(n) \leq (1 + \varepsilon) \min_{n' \in OPEN} f(n')\}.$$

Operaţiile lui A_ε^* sunt identice cu cele ale lui A^*, cu excepţia că A_ε^* selectează din *FOCAR* nodul cu valoarea h_F minimă, unde $h_F(n)$ este o a doua funcţie euristică care estimează efortul computaţional necesar completării căutării plecând din n.

Raţiunea acestei variante este simplă. În acord cu estimările disponibile ale lui $f(\bullet)$, toate nodurile din *FOCAR* au aproximativ drumuri soluţie egale. Deci în loc să se consume timp pentru identificarea celui mai bun dintre ele, este mai indicat să se genereze porţiunea rămasă a soluţiei şi nodul care oferă cel mai mic efort de completare este unul cu valoare minimă a lui h_F. Este clar că pentru $\varepsilon = 0$, A_ε^* se reduce la A^*.

Euristica h pentru *formarea* mulţimii *FOCAR* este de natură total diferită faţă de euristica h_F utilizată pentru *selectarea* nodurilor din *FOCAR*. Prima euristică anticipează reducerea *calităţii soluţiei* datorită efortului necesar continuării căutării ei. A doua euristică estimează *efortul computaţional* pentru completarea căutării.

Teorema 1.36. A_ε^* este ε-admisibil, adică el găseşte întotdeauna o soluţie al cărui cost nu depăşeşte costul optimal printr-un factor mai mare decât 1 + ε.

Demonstraţie. Adoptând următoarele notaţii:

n_0 = un nod în *DESCHIS* având o valoare f optimală,

t = nod terminal (căutat pentru expandare şi găsit a fi nod soluţie),

n_1 = primul nod din *DESCHIS* de pe o soluţie optimală,

$C(t) = f(t) = g(t)$ = costul soluţiei găsite,

rezultă:

$f(n_1) \leq C^*$ deoarece h este admisibilă,

$f(n_0) \leq f(n_1)$ deoarece *DESCHIS* este f-ordonată,

$f(t) \leq f(n_0)\,(1 + \varepsilon)$ deoarece t este ales din *FOCAR*.

Prin urmare:

$$C(t) = f(t) \leq f(n_1)\,(1 + \varepsilon) \leq C^*\,(1 + \varepsilon). \blacksquare$$

1.2.2.3.2.3. Compararea celor doi algoritmi

Ponderarea dinamică este un algoritm uşor de implementat; acest algoritm memorează doar o singură listă şi foloseşte o singură funcţie euristică. A_ε^* oferă pe de

https://biblioteca-digitala.ro / https://unibuc.ro

altă parte multe alte avantaje. În timp ce ponderarea dinamică este limitată la probleme în care adâncimea soluţiei optimale (N), sau măcar o bună margine superioară a ei, este cunoscută *a priori*, A_ε^* îşi menţine ε-admisibilitatea chiar când valoarea N este complet necunoscută. Mai mult, separarea totală a celor două euristici utilizate de A_ε^* îi asigură un mecanism în vederea incorporării unor estimări sofisticate a costului computaţional necesar completării căutării. Acest fapt implică elaborarea unor funcţii de drumul de la s la n, nu întotdeauna descriptibile de g şi h. De exemplu, dacă problema comisvoiajorului este considerată în cazul unei reţele cu localităţi rar distribuite, numărul de arce din porţiunea neexplorată a grafului constituie uzual o mai validă estimare a efortului computaţional rezidual decât proporţia $1 - depth(n) / N$ a oraşelor neexplorate, ce reprezintă ghidul algoritmului ponderare dinamică.

Compararea performanţelor celor doi algoritmi pe mai multe exemple arată comportarea ceva mai bună a algoritmului A_ε^*. Ambii algoritmi au o cea mai mare utilitate în probleme *hard* asigurând o economie cuprinsă între 60% şi 90%.

Exemplul 1.16. În cartea lui Nilsson, ideea reluării parcurgerii unor ramuri anterioare ale arborelui generat în procesul căutării soluţiei este ilustrată pe următorul exemplu. Fie o bază de date iniţială DB1, căreia îi sunt aplicate regulile de transformare R1 şi R3, având drept rezultat bazele DB2 şi DB3 (vezi figura 1.12.). Dacă sistemul de control decide că această cale este nepromiţătoare, înregistrările DB2 şi DB3 sunt şterse şi are loc întoarcerea, urmată de aplicarea regulii R2 având drept rezultat baza DB4. Dacă însă sistemul de control menţine înregistrările bazelor DB2 şi DB3 şi baza DB4 se dovedeşte inutilă, atunci se poate relua căutarea imediat după DB2 sau DB3. ■

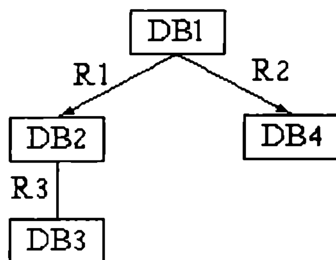


Fig. 1.12.

Exemplul 1.17. Strategia de inspectare a grafului, menţionată mai înainte şi utilizată în algoritmul A^*, este ilustrată pe exemplul ce urmează [NIL82, pag. 67].

Să presupunem că procesul de căutare a generat graful de căutare din figura 1.13.a) în care arborele de traversare T este definit de pointerii reprezentaţi de săgeţile

https://biblioteca-digitala.ro / https://unibuc.ro

paralele cu arcele grafului. Nodurile marcate cu cercuri pline sunt în *ÎNCHIS* şi cele marcate cu cercuri goale sunt în *DESCHIS*, în momentul când algoritmul a ales pentru expandare nodul 1. Arcele grafului se presupun de cost unitar. Atunci când nodul 1 este expandat, este generat nodul 2, unicul său succesor. Dar nodul 2, care are tată nodul 3 în arborele de căutare (traversare), a fost deja generat, este în *ÎNCHIS* şi are succesorii 4 şi 5. De asemenea, trebuie notat faptul că tatăl nodului 4 în arborele de căutare este nodul 6, deoarece cel mai scurt drum (de cost minim) de la nodul *s* către nodul 4 trece prin nodul 6. Deoarece algoritmul a identificat un drum către nodul 2 ce trece prin nodul 1, de cost mai mic decât drumul anterior prin nodul 3, nodul tată al nodului 2 în arborele de căutare este schimbat din 3 în 1. Costurile drumurilor către descendenţii nodului 2 în graful de căutare, adică drumurile către nodurile ,i 5, sunt recalculate. Aceste costuri sunt acum mai mici decât anterior şi tatăl nodului 4 este schimbat din nodul 6 în nodul 2. Arborele de căutare (traversare), ajustat este definit de arcele din figura 1.13.b). ■

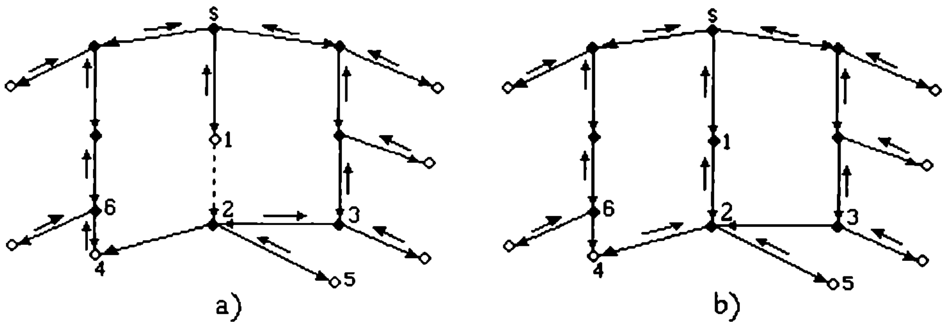


Fig. 1.13.

Exemplul 1.18. Un exemplu ilustrativ simplu privind semnificaţia şi velul de utilizare al funcţiei de evaluare *f* îl constituie *jocul 8* [NIL82, pag. 73]. Fiind date opt plăcuţe pătrate numerotate de la 1 la 8 şi plasate pe o tablă pătrată de latură trei ca în figura 1.14.a) se cere rearanjarea lor ca în figura 1.14.b), mutând succesiv orice plăcuţă pe un loc neocupat.

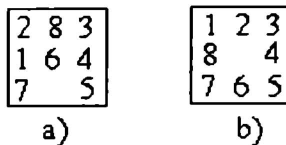


Fig. 1.14.

Drept funcţie de evaluare se consideră $f(n) = d(n) + w(n)$, unde $d(n)$ este adâncimea unui nod n în arborele de căutare şi $w(n)$ este numărul plăcuţelor incorect

https://biblioteca-digitala.ro / https://unibuc.ro

plasate pe tabla corespunzătoare nodului *n*. De exemplu, pentru configurația din figura 1.14.a) valoarea lui *f* este egală cu 0 + 4 = 4.

Rezultatul aplicării strategiei *inspectarea grafului* în cazul exemplului considerat este ilustrat în figura 1.15. Valoarea lui *f* pentru fiecare nod este încercuită; numerele neîncercuite arată ordinea în care nodurile sunt expandate.

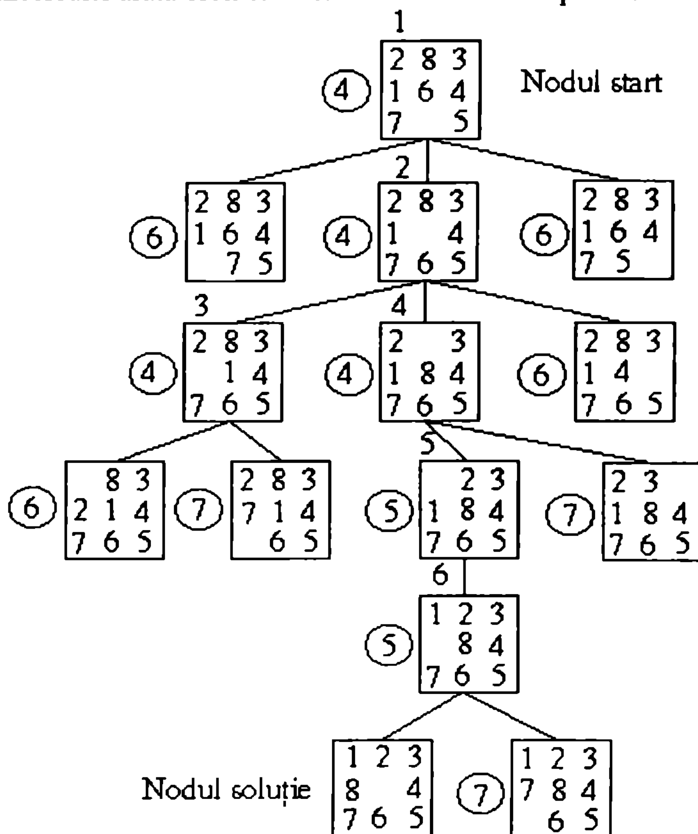


Fig. 1.15.

Soluția problemei studiate poate fi obținută și cu ajutorul altor metode de căutare, dar utilizarea funcției de evaluare are drept rezultat expandarea unui număr mai mic de noduri. ■

1.2.3. Euristicile ca informație dată de modele simplificate

1.2.3.1. Utilizarea modelelor relaxate

1.2.3.1.1. *Proveniența euristicilor*

Fie exemplul jocului 8, care constă în rearanjarea a 8 plăcuțe, numerotate de la 1 la 8, pe un tabel (tăbliță) 3×3 astfel încât să fie așezate ca în figura 1.16., dreapta.

https://biblioteca-digitala.ro / https://unibuc.ro

Rearanjarea plăcuţelor este permisă numai prin deplasarea ortogonală, adică orizontală sau verticală, a plăcuţelor de pe o poziţie adiacentă pătratului liber. Plecând de la configuraţia (s) şi procedând în modul descris, se obţin configuraţiile A, B şi C. Care dintre aceste trei alternative este mai promiţătoare? Utilizarea *căutării exhaustive* este nepractică în cazul când configuraţia iniţială este mult deosebită de cea finală. Decizia privind alegerea celei mai promiţătoare alternative, evitând căutarea exhaustivă, necesită utilizarea unei *estimări* a eficienţei căutării. În cazul jocului 8 există două euristici de estimare a proximităţii dintre două stări. Prima este numărul de plăcuţe incorect aşezate prin care diferă două stări (configuraţii) şi este notată h_1. A doua este suma distanţelor (orizontale şi verticale) dintre plăcuţele aşezate diferit în două configuraţii şi este notată cu h_2. Pentru stările A, B, C din figură, proximităţile lor la starea soluţie sunt:

$$h_1(A) = 2,\ h_1(B) = 3,\ h_1(C) = 4,$$
$$h_2(A) = 2,\ h_2(B) = 4,\ h_2(C) = 4.$$

Evident, ambele euristici decid că starea A este cea mai apropiată de soluţie şi, prin urmare, ea va fi explorată înaintea stărilor B şi C.

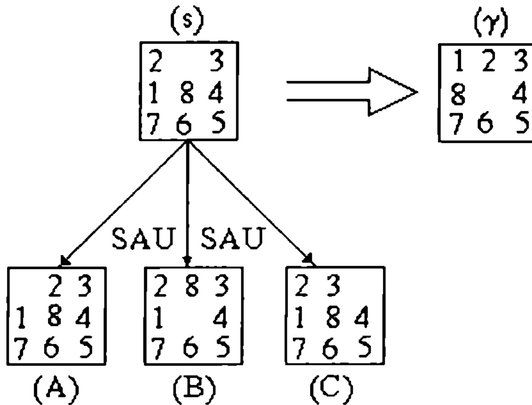


Fig. 1.16.

În cadrul încercării de motivare a raţionalităţii alegerii euristici h_2 drept euristică admisibilă, se pot întâlni argumente cum sunt cele ce urmează:

1. Fie o soluţie a problemei, nu neapărat optimală. Pentru a satisface condiţiile stării soluţie, fiecare plăcuţă trebuie să parcurgă o traiectorie de la poziţia iniţială la cea finală, costul total al soluţiei (număr de paşi) fiind suma costurilor traiectoriilor individuale. Orice traiectorie trebuie să conţină măcar atâţia paşi câţi sunt daţi de distanţa h_2 (numită şi *distanţă Manhattan*) dintre configuraţia iniţială şi cea finală şi, prin urmare, suma acestor distanţe nu poate depăşi costul total al soluţiei.

https://biblioteca-digitala.ro / https://unibuc.ro

2. Dacă este posibilă deplasarea fiecărei plăcuţe *independent* de celelalte, atunci se poate muta plăcuţa 1 dealungul celei mai scurte traiectorii în poziţia sa finală, apoi se poate proceda la fel cu plăcuţa 2 şi aşa mai departe până la obţinerea configuraţiei soluţiei. În total sunt efectuaţi măcar atâţia paşi câţi dă suma distanţelor. În consecinţă, suma acestor distanţe nu poate depăşi costul total al soluţiei.

Primul argument este analitic. În cadrul său este selectată o proprietate ce trebuie satisfăcută de orice soluţie astfel încât să fie asigurată o traiectorie a fiecărei plăcuţe către poziţia ei finală. Deci se caută costul minimal ce satisface această proprietate. Al doilea argument are o natură operaţională. El descrie o procedură pentru rezolvarea unei probleme auxiliare similară, în cadrul căreia regulile jocului au fost *relaxate* astfel încât plăcuţele pot fi aşezate una peste alta. În locul căutării unei *funcţii* $h(n)$ ce aproximează pe $h^*(n)$, se completează soluţia obţinută pe calea relaxării, se *numără* paşii necesari şi numărul obţinut se foloseşte drept o estimare a lui $h^*(n)$.

Această a doua schemă de căutare ilustrează modelul (paradigma) menţionată în continuare: Euristicile sunt descoperite consultând modele simplificate ale domeniului problemei studiate.

1.2.3.1.2. *Consistenţa euristicilor bazate pe relaxare*

Este interesant de notat că euristicile generate de optimizări pe modele relaxate sunt garantate a fi *consistente* (sau monotone). Această proprietate este uşor de verificat după cum se vede din figura 1.17., unde $h(n)$ şi $h(n')$ sunt euristicile atribuite respectiv nodurilor n şi n'. Aceste euristici descriu costul minim al completării soluţiei din aceste noduri, în cadrul unui model relaxat comun ambelor noduri. Soluţia optimală $h(n)$ trebuie să satisfacă condiţia $h(n) \leq c'(n, n') + h(n')$, unde $c'(n, n')$ este costul relaxat al arcului (n, n'). Altfel $c'(n, n') + h(n')$ ar constitui, în locul lui $h(n)$, costul optimal al drumului din n. Costul relaxat $c'(n, n')$ nu poate depăşi, prin definiţie, costul original $c(n, n')$. Rezultă că:

$$h(n) \leq c(n, n') + h(n'),$$

care, împreună cu condiţia tehnică h(nod soluţie) = 0, completează condiţiile de consistenţă (vezi 1.2.2.2.5.).

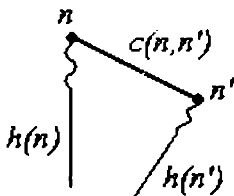


Fig. 1.17.

https://biblioteca-digitala.ro / https://unibuc.ro

Această proprietate are implicaţii computaţionale şi psihologice. Din punct de vedere computaţional, ea garantează că A^*, dirijat de euristicile unui model relaxat, va evita efortul de a redeschide noduri din *ÎNCHIS* şi a redirecţiona pointerii lor. Pointerii atribuiţi oricărui nod expandat de algoritm sunt deja direcţionaţi dealungul drumului optimal către acest nod.

Din punct de vedere psihologic, dacă se presupune că oamenii descoperă euristici pe calea consultării unui model relaxat, se poate explica faptul că majoritatea euristicilor elaborate sunt admisibile şi consistente.

1.2.3.2. Generarea mecanică a euristicilor admisibile

1.2.3.2.1. *Relaxarea sistematică*

Eliminarea restricţiilor în cadrul relaxării poate fi sistematizată astfel încât euristicile naturale, cum sunt cele prezentate în paragraful 1.2.3.1.1., să fie generate în mod mecanic. Pentru aceasta este convenabilă reprezentarea domeniului problemei prin restricţiile ce guvernează aplicabilitatea şi impactul diferitelor transformări în acest domeniu. Fie, de exemplu, jocul 8. Este total nepractic să se specifice mulţimea mutărilor legale cu ajutorul listei perechilor de stări, dinaintea şi după efectuarea fiecărei mutări. O reprezentare mai naturală a jocului constă în specificarea mutărilor permise cu ajutorul a două mulţimi de condiţii, o condiţie fiind adevărată înaintea unei mutări şi a doua după efectuarea acestei mutări. De exemplu, acţiunile pot fi reprezentate prin trei liste:

1. **Lista precondiţiilor** - reprezentând conjuncţia predicatelor adevărate înaintea efectuării unei acţiuni.
2. **Lista adăugare** - lista predicatelor ce trebuie adăugate în vederea descrierii mulţimii stărilor în urma aplicării unei acţiuni.
3. **Lista ştergere** - lista predicatelor ce nu mai sunt adevărate după ce o acţiune a fost aplicată şi, în consecinţă, sunt eliminate.

În continuare această reprezentare este folosită în formalizarea schemei de relaxare pentru jocul 8.

Mai întâi se definesc trei predicate primitive:

$PE(x, y)$ - plăcuţa x se află pe căsuţa (locaţia) y,

$LIBER(y)$ - căsuţa y nu conţine plăcuţe (e liberă),

$AD(y, z)$ - căsuţa y este adiacentă cu căsuţa z,

https://biblioteca-digitala.ro / https://unibuc.ro

unde variabila x reprezintă plăcuțele $X_1,X_2,...,X_8$ și variabilele y și z parcurg mulțimea locațiilor $C_1,C_2,...,C_9$. Cu toate că predicatul *LIBER* poate fi definit în termeni de *PE*:

$$LIBER(y) \Leftrightarrow \forall x \sim PE(x, y),$$

unde ~ notează negația, este convenabil să se folosească acest predicat în mod explicit, drept un predicat independent. Folosind aceste primitive, orice stare este reprezentată printr-o listă de nouă predicate de forma:

$$PE(X_1, C_1), PE(X_2, C_2), ..., PE(X_8, C_8), LIBER(C_9),$$

împreună cu descrierea tabelului:

$$AD(C_1, C_2), AD(C_1, C_4), ...$$

Mișcarea corespunzătoare transferării plăcuței x din locația y în locația z este descrisă prin trei liste:

MUTĂ(x, y, z):

lista precondițiilor:	$PE(x, y), LIBER(z), AD(y, z)$;
lista adăugare:	$PE(x, z), LIBER(y)$;
lista ștergere:	$PE(x, y), LIBER(z)$,

care descrie detaliile mutării plăcuței x din locația y în locația z. Problema constă în găsirea unei secvențe aplicabile de atribuiri variabilelor din operatorul de bază *MUTĂ*(x, y, z), care să transforme starea inițială într-o stare care să satisfacă criteriile ce asigură o stare soluție.

În continuare se examinează efectul relaxării problemei pe calea eliminării condițiilor $LIBER(z)$ și $AD(y, z)$ din lista precondițiilor. Jocul rezult · permite deplasarea printr-o singură mișcare a unei plăcuțe de pe poziția curentă în poziția dorită. O problemă dată poate fi ușor rezolvată utilizând următoarea strategie de control. În orice stare, se alege o plăcuță neașezată pe locația dorită. Fie X_1 această plăcuță, Y_1 locația ei curentă și Z_1 locația dorită. Se aplică operatorul *MUTĂ*(X_1,Y_1,Z_1) și se repetă procedura stării următoare până când toate plăcuțele ajung să fie corect plasate, deci se obține soluția.

Este evident că numărul de mutări necesare pentru a rezolva o asemenea problemă este egal exact cu numărul de plăcuțe plasat greșit în configurația inițială. Dacă pentru rezolvarea acestei probleme se folosește un program de rezolvare mecanică (*mechanical problem-solver*) și se numără mutările necesare, se constată că soluția corespunde euristicii h_1.

Să presupunem acum că se elimină doar condiția $LIBER(z)$ pe când $AD(y, z)$ se păstrează. Modelul rezultat permite mutarea oricărei plăcuțe într-o locație adiacentă, independent de faptul că această locație este ocupată. Rezultă euristica h (•): suma distanțelor Manhattan.

https://biblioteca-digitala.ro / https://unibuc.ro

O altă schimbare urmează în mod natural şi constă în reţinerea condiţiei *LIBER*(z) şi eliminarea lui *AD*(y, z). Modelul obţinut permite mutarea oricărei plăcuţe pe locaţia liberă, chiar dacă locaţiile participante nu sunt vecine. Problema obţinerii configuraţiei soluţie este echivalentă cu cea a sortării prin *interschimbare* a unei liste de elemente, adică schimbarea simultană a locaţiilor a două elemente, fiecare pas trebuind să schimbe un element marcat *blank* cu un alt element. Soluţia optimală a problemei sortării prin interschimbare se poate obţine utilizând următorul algoritm de tip *Greedy*:

Algortimul 1.12. (*Jocul 8*)

1. [Mutare] Dacă locaţia curentă vidă *y* trebuie ocupată de plăcuţa *x*, mută pe *x* în *y*. Altfel, adică dacă *y* trebuie să fie vidă în starea soluţie, mută în *y* orice plăcuţă necorect plasată.
2. [Ciclare] Dacă mai sunt plăcuţe necorect plasate repetă 1, altfel STOP. ■

Euristica h_3 obţinută în modul descris mai sus este apropiată, ca eficienţă, de euristica h_1. Acest rezultat ilustrează faptul că metoda eliminării sistematice a precondiţiilor este capabilă să genereze euristici netriviale.

În general, problemele rezultate prin eliminarea precondiţiilor nu permit obţinerea, întotdeauna, a unor soluţii mai simple şi uneori pot deveni mai dificile decât problema dată. Prin urmare, căutarea unui model în spaţiul eliminărilor trebuie abordată cu precauţie.

1.2.3.2.2. *Observaţii şi concluzii*

Am prezentat o schemă naturală pentru a elabora euristici pentru probleme combinatoriale. Mai întâi, domeniul problemei este formulat în termeni de trei liste de operatori ce descriu transformarea stărilor din domeniu şi condiţiile ce definesc stările soluţie. Apoi, precondiţiile ce limitează aplicabilitatea operatorilor sunt parţial eliminate, generând un meta-spaţiu de căutare a modelului relaxat obţinut în modul descris. Căutarea începe ori pe modelul iniţial, în care operanzii precondiţiilor sunt eliminaţi câte unul, ori plecând de la un model trivial ce nu conţine precondiţii şi căruia i se adaugă succesiv restricţii. Modelul generat de un număr mai mic de restricţii are proprietatea de *decompozabilitate*, în sensul că operatorii ce descriu evoluţia modelului devin mai independenţi între ei, fapt ce permite rezolvarea problemei studiate prin strategii de tip *Greedy*, fără *backtracking*.

Cu toate că efortul investit în căutarea unui model relaxat pare a fi dificil, el este de obicei răsplătit de eficienţa algoritmului elaborat.

https://biblioteca-digitala.ro / https://unibuc.ro

1.2.4. Euristici bazate pe probabilitate

Metodele probabiliste specifică o mulțime de configurații ale unui mediu împreună cu gradul lor de probabilitate. În problema comisvoiajorului, de exemplu, tabelul distanțelor dintre localități specifică o configurație a mediului pe care trebuie să îl parcurgă comisvoiajorul. Dacă, în loc să se specifice distanțele, sunt furnizate marginile inferioare și superioare între care se află fiecare distanță, atunci acest tabel de date definește o *mulțime* de probleme posibile. În sfârșit, dacă se specifică *distribuțiile* distanțelor dintre localități, devine posibilă determinarea faptului dacă o configurație este mai credibilă decât alta și cu cât mai mult credibilă. Cu alte cuvinte, distribuția cuantifică probabilitatea cu care se realizează configurațiile.

Este de remarcat faptul că oamenii își însușesc și invocă considerații probabiliste drept parte a codificării mintale a experiențelor lor. Astfel, în mod normal, un eveniment este apreciat ca fiind probabil dacă posedă particularități considerate *prototipice* unei populații largi.

Deoarece testul fundamental pentru succesul metodelor euristice constă în faptul că ele funcționează bine *de cele mai multe ori* și din cauză că teoria probabilităților este principalul formalism de cuantificare a noțiunii *de cele mai multe ori*, rezultă că modelele probabiliste pot furniza un fundament formal pentru evaluarea cantitativă a metodelor euristice. Mai mult, este de asemenea natural ca modelele probabiliste, atunci când sunt disponibile, să fie consultate în procesul elaborării metodelor euristice, ori în selectarea parametrilor care definesc aceste metode, astfel încât să se poată garanta că numai unei fracțiuni mici din variantele problemei studiate nu i se poate asigura o tratare adecvată.

1.2.4.1. Euristici bazate pe cel mai plauzibil rezultat

Metodele euristice sunt adesea bazate pe estimări ale valorilor necunoscute ale unor variable. De exemplu, algoritmul A^* este ghidat de funcția h care estimează valoarea necunoscută a lui h^*, costul minimal de completare a unui drum soluție. Dacă domeniul problemei este caracterizat de un model probabilistic, uneori este posibil să se calculeze cea mai probabilă valoare a unei variabile necunoscute și să se utilizeze această valoare în construirea funcției euristice.

Fie, de exemplu, problema găsirii celui mai ieftin drum într-un graf în care toate costurile arcelor sunt extrase independent dintr-o funcție comună de distribuție cu valoarea medie μ. Dacă N este numărul de arce rămase între nodul n și nodul soluție, atunci pentru un N mare legea numerelor mari afirmă că pentru costul unui

https://biblioteca-digitala.ro / https://unibuc.ro

drum soluţie se poate lua o valoare în vecinătatea lui μN. Prin urmare, dacă este sigur că există doar un singur drum între n şi nodul soluţie, atunci valoarea μN poate fi acceptată drept o estimare a lui h^* şi în A^* se poate utiliza funcţia $f = g + \mu N$.

Prin natura lor, estimările bazate pe probabilitate nu sunt garantate ca fiind admisibile. Uneori aceste estimări pot depăşi cu mult costul h^* şi pot cauza încheierea prematură a lui A^* cu o soluţie suboptimală. Pe de altă parte, dacă nu se caută întotdeauna o soluţie optimală exactă şi se acceptă găsirea cât mai frecvent a unei soluţii *bune*, euristicile bazate pe probabilitate sunt perfect acceptabile.

1.2.4.2. Euristici bazate pe eşantionare

Eşantionarea este poate cea mai veche şi cea mai des practicată metodă euristică. Ea constă în deducerea unei proprietăţi a unei mulţimi mari de elemente din proprietăţile unei submulţimi de elemente, alese la întâmplare.

Fie, de exemplu, o bază de date imensă conţinând N cifre binare pentru care trebuie calculată *proporţia* cifrelor 1. Evident că răspunsul exact nu poate fi identificat decât dacă se inspectează toate cele N cifre, adică parcurgând N paşi. Dacă însă se admite o eroare mică, este suficient un număr de paşi funcţie de eroarea acceptată. Este suficient să se aleagă la întâmplare n cifre şi să se calculeze proporţia de cifre egale cu 1. Pentru valori mari ale lui N, celebra teoremă a lui Bernoulli garantează că dacă proporţia exactă este π şi proporţia eşantionării este γ, atunci abaterea dintre ele este mărginită de următoarea relaţie:

$$P(|\pi - \gamma| \geq \varepsilon) \leq 2\, e^{-n\varepsilon^2/2}.$$

Rezultă că numărul de observaţii necesar pentru a garanta o probabilitate η suficient de mare astfel încât eroarea $|\pi - \gamma|$ să fie mai mică decât ε este dată de:

$$n = 2 / \varepsilon^2 \log (2 / (1 - \eta)).$$

Acest număr depinde numai de ε şi η, nu şi de N, astfel încât rezultă o reducere a efortului de calcul de la o valoare liniară, egală cu N în cazul parcurgerii integrale a mulţimii date de cifre, la o valoare constantă.

Mai mult, dacă N creşte indefinit, se poate alege drept mărime a eşantionării orice funcţie divergentă de N, $n = g(N)$, ceea ce garantează că abaterea dintre γ şi π tinde către zero cu o probabilitate apropiată de 1, adică:

$$P(|\pi - \gamma| < \varepsilon) \to 1, \text{ pentru } \forall \varepsilon > 0.$$

În felul acesta, complexitatea se reduce de la N la $g(N)$, unde funcţia $g(N)$ poate creşte foarte încet, de exemplu ca $\log N$, $\log \log N$ sau chiar mai încet.

https://biblioteca-digitala.ro / https://unibuc.ro

1.2.5. Modele abstracte pentru analiza cantitativă a performanţei

1.2.5.1. Analiza matematică a performanţei

Utilizarea unor modele abstracte în analiza performanţei algoritmilor constituie o abordare naturală a acestei probleme în cadrul căreia se folosesc structuri simplificate, modele probabilistic abordabile, jocuri sintetice şi alte tipuri de modele. Aplicabilitatea acestor investigaţii este limitată, evident, la acele probleme care au o structură similară cu modelele matematice elaborate, dar, ca şi în cazul cercetărilor de laborator, ele sunt utile în identificarea factorilor care au o influenţă decisivă asupra analizei problemei studiate.

Modelele probabilistice sunt eficient aplicabile în studiul problemelor practice în care se urmăreşte optimizarea unei măsuri a performanţei constând într-o combinaţie a calităţii soluţiei şi efortul de căutare, *mediat* pe toate problemele *frecvent* întâlnite. Teoria probabilităţilor constituie un limbaj foarte potrivit pentru formalizarea descrierii acelor aspecte ale performanţei ce urmează a fi ameliorate. În acest scop, se pleacă de la un model abstract simplificat, conţinând un număr minim de parametri necesari pentru reprezentarea particularităţilor mai deosebite ale domeniului problemei, după care se analizează performanţa diferiţilor algoritmi de căutare pe acest model ca funcţie de parametrii modelului. Dacă drept rezultat al acestei analize, apare evident că valoarea unui anumit parametru are o influenţă mare asupra performanţelor metodei, acest parametru devine focarul atenţiei în eforturile empirice de caracterizare a domeniului problemei. Alternativ, dacă o metodă dată de căutare se dovedeşte a avea o performanţă înaltă pentru un număr mare de parametri, această metodă devine un prim candidat în vederea testării pe diferite variante ale problemei.

1.2.5.2. Exemplul 1: Găsirea unui cel mai scurt drum într-o latice regulată

Punerea problemei. Fie graful latice infinit din figura 1.18., în care fiecare nod este identificat cu perechea de coordonate întregi (x, y) şi fiecare arc are lungimea unitate. Trebuie găsit un cel mai scurt drum între nodul start $s = (0, 0)$ şi nodul soluţie $\gamma = (m, n)$ folosind distanţa aeriană:

$$h(x, y) = \sqrt{(x-m)^2 + (y-m)^2},$$

drept euristică ghid. Se cere estimarea, pentru valori mari m şi n, a numărului Z de noduri expandate de A^*, folosind următoarele funcţii de evaluare:

https://biblioteca-digitala.ro / https://unibuc.ro

a) $f = g$,

b) $f = g + h$,

c) $f = h$. ■

Aplicaţii posibile: organizarea memoriilor bidimensionale.

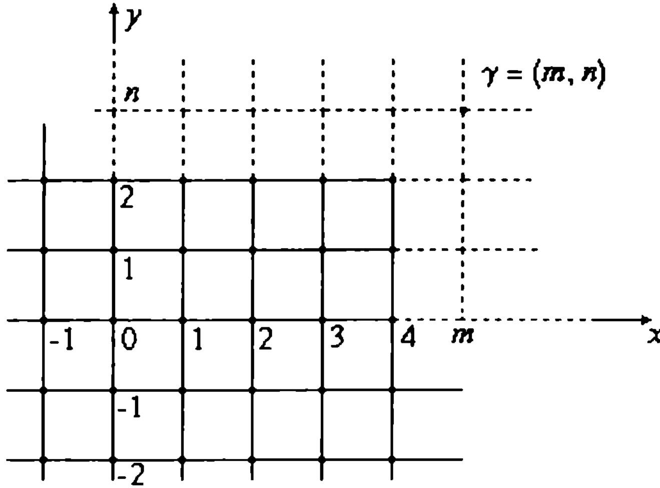


Fig. 1.18.

Soluţia. Funcţia h este o euristică admisibilă şi consistentă, deoarece este o geodezică în modelul relaxat în care orice deplasare este permisă în orice direcţie. Prin urmare în toate nodurile expandate $g=g^*$. În plus, expresiile exacte pentru g^* şi h^* sunt:

$$g^*(x, y) = |x| + |y|,$$

$$h^*(x, y) = |m - x| + |n - y|.$$

Prin urmare, presupunând că $m \geq 0$, $n \geq 0$, rezultă:

$$f^*(s) = f^*(0, 0) = m + n.$$

Cazul a): $f = g$.

Pentru $f = g$, algoritmul A^* devine un algoritm de căutare în lăţime, în care fiecare nod ce satisface inegalitatea:

$$g^*(x, y) < f^*(s),$$

sau:

$$|x| + |y| < m + n,$$

va fi expandat. Mulţimea acestor noduri este circumscrisă de conturul pătrat din figura 1.19., a cărei arie are expresia:

$$I_a = 2(m + n)^2.$$

https://biblioteca-digitala.ro / https://unibuc.ro

Deoarece densitatea nodurilor este unitară şi frontiera poate conţine cel mult $O(m + n)$ noduri, numărul de noduri Z_a expandate fără informaţia euristică este:

$$Z_a = 2(m + n)^2 + O(m + n). \quad (1.7)$$

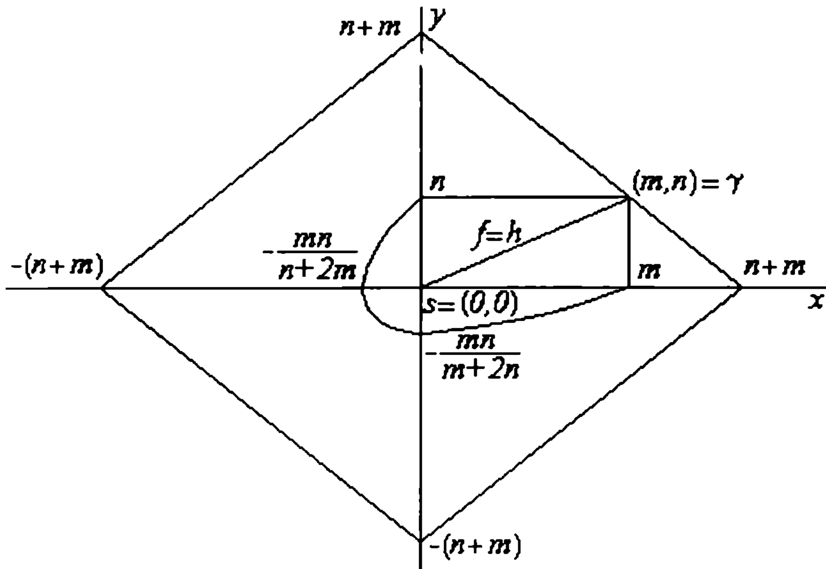


Fig. 1.19.

Cazul b): $f = g + h$.

Mulţimea nodurilor expandate este inclusă în frontiera satisfăcând ecuaţia:

$$g(x, y) + h(x, y) = m + n,$$

sau:

$$|x| + |y| + \sqrt{(x-m)^2+(y-n)^2} = m + n. \quad (1.8)$$

Forma frontierei este reprezentată de conturul interior din figura 1.19. Aria delimitată de acest contur poate fi determinată ca sumă a următoarelor patru componente:

1. $x \geq 0, y \geq 0$. Ecuaţia (1.8) devine:

$$2\,(m - x)\,(n - y) = 0,$$

care circumscrie dreptunghiul $0 \leq x \leq m$, $0 \leq y \leq n$ de arie:

$$J_1 = m \bullet n.$$

2. $x \geq 0, y < 0$. Ecuaţia (1.8) devine:

$$4n\,y + 2\,(m - x)\,(n + y) = 0,$$

sau:

$$y = 2n^2 / (m + 2n - x) - n,$$

https://biblioteca-digitala.ro / https://unibuc.ro

care include aria:

$$I_2 = - \int_0^m [2n^2 / (m + 2n - x) - n]\, dx = m\, n - 2n^2 \ln(1 + m / (2n)).$$

3. $x < 0$, $y \geq 0$. Acest caz este simetric cu cel precedent, valorile lui m şi n schimbându-se între ele. Rezultă că:

$$I_3 = m\, n - 2m^2 \ln(1 + n / (2m)).$$

4. $x < 0, y < 0$. În acest caz $|x| = - x$, $|y| = - y$ şi ecuaţia (1.8) devine:

$$4m\, x + 4n\, y + 2(m + x)\, (n + y) = 0,$$

sau:

$$y = 2(m + n)^2 / (m + 2n + x) - (n + 2m).$$

Rezultă că:

$$I_4 = - \int_{\frac{-mn}{n+2m}}^{0} [2(m + n)^2 / (m + 2n + x) - (n + 2m)]\, dx =$$
$$= m\, n - 2(m + n)^2 \ln(1 + m\, n / (2(m+n)^2)).$$

Însumând cele patru arii se obţine:

$$Z_b = 4mn - 2n^2 \ln(1+m/(2n)) - 2m^2 \ln(1+n/(2m)) - 2(m+n)^2 \ln(1+mn/(2(m+n)^2)). \quad (1.9)$$

Pentru cazul special $m = n$ din (1.9) rezultă că:

$$Z_b \approx 1.436\, m^2,$$

ceea ce reprezintă 18% din numărul de noduri expandate prin căutarea în lă; ıe (vezi (1.7)). Pentru $n = 0$, nodurile sunt situate pe axa x şi numărul lor este mărginit de m. Figura 1.20. ilustrează performanţa relativă Z_b / Z_a ca funcţie de raportul n / m. Această curbă arată că euristica distanţă aeriană asigură o eficienţă mare pentru $m << n$, respectiv $n << m$, ţinând seama de simetria funcţiilor Z_a şi Z_b în m şi n. Acest rezultat era de aşteptat deoarece cea mai mică eroare relativă dintre distanţa optimală h^* şi distanţa aeriană h este atinsă când punctul soluţie este aproape de axe.

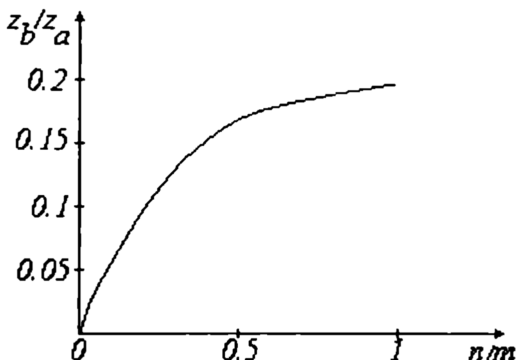


Fig. 1.20.

https://biblioteca-digitala.ro / https://unibuc.ro

Cazul c): $f = h$.

Structura particulară a grafului problemei date arată că în acest caz se poate obţine un drum optimal, expandând numai $m + n$ noduri situate pe acest drum optimal. Motivul acestei eficienţe constă în faptul că orice nod are un fiu pe drumul optimal către nodul soluţie pentru care valoarea lui h este strict mai mică decât valoarea corespunzătoare nodului tată. Deoarece nodul tată avea o valoare h minimă în *DESCHIS*, rezultă că şi fiul va avea o valoare minimă a lui h şi va fi imediat selectat, fără ca algoritmul să intre în *backtracking*.

O motivare de natură geometrică a afirmaţiei precedente se poate deduce urmărind modul de generare al unui drum optimal pe graful din figura 1.18., în care $m > 0$, $n > 0$. Fie dreptunghiul definit de punctele O, γ, $(0, n)$ şi $(m, 0)$. Orice încercare de a trasa un drum optimal printr-un punct exterior acestui dreptunghi eşuează, deoarece distanţa h a acestui punct la γ este sigur mai mare decât distanţa la γ a nodului său tată, situat pe conturul dreptunghiului. De asemenea, orice drum al cărui ultim arc are un sens invers axei x, respectiv y, nu poate fi optimal. Rezultă că un drum optimal nu poate cuprinde decât arce orientate în sensurile pozitive ale axelor şi nu trece prin noduri din afara dreptunghiului, deci are lungimea $m + n$.

Sumar. Acest exemplu demonstrează mai multe particularităţi ale algoritmului A^* dintre care semnalăm următoarele.

1. Cea mai mare parte a nodurilor expandate se află pe drumuri optimale. În exemplul precedent, toate $m \bullet n$ noduri situate în dreptunghiul $0 \le x \le m$, $0 \le y \le n$ sunt situate pe drumuri optimale şi sunt expandate de A^* chiar dacă funcţia empirică ajunge aproape perfectă. Dacă graful conţine mai multe drumuri aproape optimale, A^* le explorează complet înainte de terminare. Metode de evitare a acestui dezavantaj al algoritmului A^* au fost prezentate în paragraful 1.2.2.3.
2. Luând $f = h$ se poate asigura ocazional o căutare mai eficientă decât cu $f = g + h$. În general însă, în acest caz rezultă soluţii suboptimale şi, mai semnificativ, pot exista forme ale lui h care cresc efortul de căutare sau împiedică încheierea algoritmului A^*. De exemplu, dacă în exemplul precedent se utilizează euristica:

$$h_1 = \begin{cases} \sqrt{(m-x)^2 + (n-y)^2}, & \text{pentru } y \ne y_0 \\ 0, & \text{pentru } y = y_0, y_0 < n, \end{cases}$$

atunci A^* nu se încheie niciodată cu $f = h_1$, deoarece are loc expandarea continuă a nodurilor de pe drumul $y = y_0$.

https://biblioteca-digitala.ro / https://unibuc.ro

1.2.5.3. Exemplul 2: Găsirea unui cel mai scurt drum într-o reţea de oraşe distribuite la întâmplare

Punerea problemei. Fie o hartă de drumuri caracterizată printr-o reţea densă de oraşe, distribuite la întâmplare cu o densitate ρ pe unitatea de suprafaţă. Se cere să se estimeze complexitatea identificării unui cel mai scurt traseu între oraşul start s, situat în originea (0, 0) şi oraşul destinaţie γ situat în punctul $(0, c)$, utilizând algoritmul A^* şi aplicând euristica distanţei aeriene. ■

Aplicaţii posibile: turism, aviaţie, marină.

Soluţia. Fie $k(n_1, n_2)$ distanţa minimă dintre oraşele n_1 şi n_2 şi $d(n_1, n_2)$ distanţa aeriană dintre ele. În particular:

$$d(s, \gamma) = h(s) = c,$$

$$k(s, \gamma) = h^*(s) = a.$$

Raportul c / a măsoară acurateţea estimării distanţei start-destinaţie şi poate fi, în general, o funcţie de ρ şi c. Mărimea problemei depinde de asemenea de ρ şi de c. Căutarea în lăţime expandează, estimativ, toate cele $M = \rho\pi c^2$ noduri situate în cercul de rază c cu centrul în s şi de aceea acest număr este ales drept standard în evaluarea euristicii distanţă aeriană. Mai precis, eficienţa selectării η_h a lui $h(\bullet)$ este definită drept raport între numărul de noduri expandat cu şi fără utilizarea lui $h(\bullet)$, adică:

$$\eta_h = M / Z_h = \rho\pi c^2 / Z_h, \tag{1.10}$$

unde Z_h este numărul de noduri expandate de A^* utilizând pe h. Problema de rezolvat constă în calculul lui η_h ca funcţie de mărimea M şi acurateţea parametrului c / a.

Condiţia necesară pentru ca A^* să expandeze un nod n implică faptul că (vezi teorema 1.27.) numai nodurile satisfăcând condiţia:

$$g^*(n) + h(n) \le f^*(s) = a \tag{1.11}$$

sunt candidaţi pentru expandare. Deoarece $h(n)$ este egală cu distanţa $d(n, \gamma)$ şi $g^*(n)$ este margine superioară pentru $d(s, n)$, orice nod ce satisface (1.11) trebuie să satisfacă de asemenea condiţia:

$$d(s, n) + d(n, \gamma) \le a. \tag{1.12}$$

Inegalitatea (1.12) defineşte o elipsă cu focarele s şi γ şi axele a şi $\sqrt{a^2-c^2}$. Aria acestei elipse este egală cu $\pi a \sqrt{a^2-c^2} / 4$ şi, deoarece fiecare nod din afara elipsei este exclus de la expandare, se poate deduce marginea superioară a numărului de noduri expandate:

$$Z_h \le \rho\, \pi\, a \sqrt{a^2-c^2} / 4. \tag{1.13}$$

https://biblioteca-digitala.ro / https://unibuc.ro

Atunci când acuratețea c a distanței aeriene crește, eroarea a - c scade, excentricitatea elipsei crește și o fracțiune mare de noduri nu este expandată. Raportul dintre ariile elipsei și cercului (vezi figura 1.21.) definește eficiența căutării funcției h. În figura 1.21., cercul R delimitează regiunea în care nodurile sunt până la urmă expandate de căutarea în lățime. Orice nod din afara elipsei E este sigur neexpandat de A^*. Conturul C_1 conține acele noduri care sunt sigur expandate de A^*.

Definind:

$$\alpha = (a / c) - 1,$$

din ecuațiile (1.10) și (1.13) rezultă:

$$\eta_h \geq 4 / [(1 + \alpha)\, \alpha^{1/2}\, (2 + \alpha)^{1/2}] \qquad (1.14)$$

și, presupunând că $\alpha << 1$, rezultă că o fracțiune de cel mult $(\alpha / 8)^{1/2}$ de noduri expandate de căutarea în lățime este de asemenea expandată de A^*.

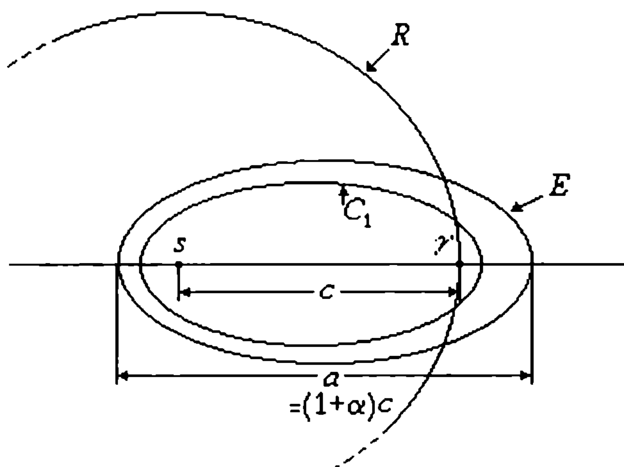


Fig. 1.21.

Deoarece harta drumurilor a fost presupusă omogenă, este de asemenea rezonabil să se presupună că raportul a / c rămâne constant pentru distanțe mari. Prin urmare, marginea inferioară (1.14) rămâne un factor constant indiferent de cât de departe este γ de s.

Pentru a arăta că η_k este de asemenea superior mărginită, este necesară acceptarea unor presupuneri privind regularitatea interconectărilor în harta drumurilor. Mai întâi se presupune că, în afara unei mulțimi de noduri de măsură nulă, estimarea relativă a erorii X este mărginită deasupra lui zero, adică:

$$X(n) = k(s, n) / d(s, n) - 1 \geq \beta > 0.$$

https://biblioteca-digitala.ro / https://unibuc.ro

Cu această presupunere se poate utiliza condiţia suficientă de expandare (vezi teorema 1.35.):

$$g^*(n) + h(n) < a \tag{1.15}$$

şi conchide că toate nodurile ce satisfac condiţia:

$$(1 + \beta)\, d(s, n) + d(n, \gamma) < (1 + \alpha)\, d(s, \gamma) \tag{1.16}$$

vor fi expandate. Aici s-a făcut caz şi de consistenţa lui h, care face ca ecuaţia (1.15) să fie aplicabilă tuturor nodurilor, nu neapărat celor din *DESCHIS*. Inegalitatea (1.16) defineşte conturul C_1 din figura 1.21., ce conţine toate nodurile care sunt sigur expandate.

Mai general, probabilitatea P ca un nod situat în afara lui C_1 să fie expandat de A^* este egală cu probabilitatea ca $X(n)$ să satisfacă inegalitatea:

$$(1 + X(n))\, d(s, n) + d(n, \gamma) < (1 + \alpha)\, d(s, \gamma).$$

Rezultă că, dacă $X(n)$ este caracterizată de o singură funcţie de distribuţie $F_X(x)$, conturul isoprobabilitate ce conectează toate punctele de probabilitate P este definit de ecuaţia:

$$(1 + F_X^{-1}(P))\, d(s, n) + d(n, \gamma) = (1 + \alpha)\, d(s, \gamma).$$

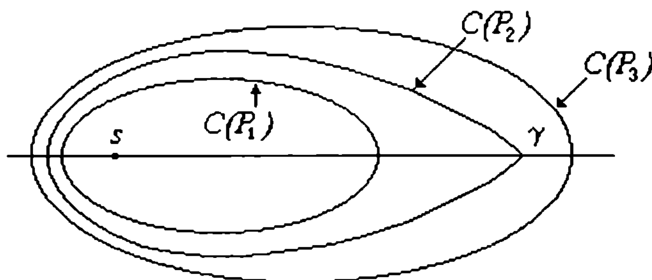


Fig. 1.22.

Unele dintre aceste contururi sunt arătate pe figura 1.22. De notat faptul că *ramurile de căutare* sunt mai concentrate în vecinătatea nodului soluţie, indicând că A^* contribuie cu mai mult efort în jurul nodului start, unde acurateţea lui $h(\bullet)$ relativ la mărimea lui g este mai redusă.

Nodurile din interiorul conturului C_1 (figura 1.21.) sunt sigur expandate şi, deoarece aria mărginită de C_1 este proporţională cu a^2, rezultă că eficienţa selectării η_h obţinută prin utilizarea euristicii distanţă aeriană este cel mult o constantă multiplicativă. Cu alte cuvinte, dacă N este numărul de noduri dealungul unui drum optimal ($N \approx \rho^{1/2} a$) atunci, pentru un ρ dat, complexitatea lui A^* rămâne de ordinul

https://biblioteca-digitala.ro / https://unibuc.ro

$\Theta(N^2)$. Se poate de asemenea argumenta că, în cazul în care densitatea ρ crește, nu există o garanție a creșterii acurateții lui h cât timp nu se formulează ipoteze adiționale privind statisticile ce corelează distanțele cu conexiunile drumurilor.

1.2.5.4. Exemplul 3: Căutarea unui drum optim într-un arbore cu costuri aleatoare

Punerea problemei. Se cere să se găsească un cel mai ieftin drum din rădăcina unui arbore către oricare dintre frunzele sale. Arborele este binar și de înălțime N. Fiecare ramură poate avea costul 1 sau 0 cu probabilitatea p sau 1-p. ■

Aplicații posibile: sisteme distribuite, energetică, telefonie.

Sunt folosite două metode de căutare. Prima este de tip *cost uniform* (vezi paragraful 1.2.2.3.) și nu folosește euristici privind partea neexplorată a arborelui de căutare. A doua constituie o căutare în etape și constă în folosirea periodică a informației în vederea eliminării acelor noduri *rele*, care nu au o performanță în acord cu așteptările.

Enunțul problemei definește o căutare ce urmează a fi executată pe o mulțime de *variante*. În fiecare variantă a problemei, trebuie găsit un cel mai ieftin drum către un nod terminal într-un arbore T, din spațiul de variante $T(N, p)$, conținând arbori binari uniformi de înălțime N cu arce de cost 0 sau 1. Probabilitatea ca un arbore T, având o anumită asignare a costurilor, să facă parte din $T(N, p)$ este egală cu $p^{N_1}(1-p)^{N_0}$ unde N_1 și N_0 sunt respectiv numerele de arce din T de cost 1 și 0 și $N_1 + N_0 = 2^{N+1} - 2$.

Un nod J din arbore are *costul c* dacă c este suma costurilor arcelor de pe drumul din rădăcină la nodul J. Un interes special în cadrul problemei studiate îl au costurile frunzelor (nodurilor terminale) și în special costul celei mai ieftine frunze, care urmează să fie identificată. Fie variabila aleatoare $C(N, p)$ descriind *numărul minim de 1* de pe un drum rădăcină - frunză într-un arbore T din mulțimea $T(N, p)$. După cum se va vedea mai departe, distribuția $C(N, p)$ variază în mod semnificativ cu p și are caracteristici diferite în următoarele trei regiuni: $p < 1/2$, $p = 1/2$ și $p > 1/2$. Mai mult, natura acestei distribuții decide care algoritm este mai potrivit pentru căutare. Din acest motiv aceste trei regiuni sunt analizate separat.

Un nod J' este un *fiu* (α, L) al nodului J dacă au loc următoarele condiții:

1. J' este situat cu L nivele mai jos decât J.
2. Costul drumului de la J la J' nu depășește pe αL.

https://biblioteca-digitala.ro / https://unibuc.ro

O familie de t fii succesivi (α, L) reprezintă un drum în T ce constă din t segmente consecutive, fiecare de cost cel mult αL. Un asemenea drum se numeşte *drum regulat* (α, L) şi familii de asemenea drumuri se numesc *familii regulate* (α, L). Pentru simplificarea notaţiei, un drum încheiat la un nod frunză se numeşte tot *regulat* (α, L), chiar dacă ultimul său segment conţine mai puţin de L noduri şi costul său este cel mult αL. Evident că orice drum rădăcină - frunză care este un drum regulat (α, L) nu poate avea un cost mai mare decât $\alpha (N + L)$.

În continuare este prezentată analiza a doi algoritmi de căutare A_1 şi A_2. A_1 este un algoritm de cost uniform şi este analizat pentru cazurile: $p<1/2$ şi $p=1/2$. A_2 este un algoritm hibrid de căutare locală şi globală în adâncime şi este analizat pentru $p > 1/2$.

Ambii algoritmi încep de la un subarbore conţinând rădăcina lui T şi, ca pas general, expandează un nod de pe frontiera subarborelui. Expandarea unui nod J constă în crearea a doi urmaşi şi a arcelor de la J la aceşti urmaşi.

Algoritmul A_1. La fiecare pas, se expandează cel mai din stânga nod dintre nodurile frontieră de cost minim. Algoritmul se încheie când se încearcă expandarea unei frunze a lui T. Această frunză reprezintă o soluţie optimală.

Algoritmul A_2. Se efectuează o căutare în adâncime în vederea găsirii unui drum regulat (α, L) de la un nod de nivel d la o frunză, unde d, α şi L sunt trei parametri exteriori aleşi. Cu alte cuvinte, A_2 este o strategie de căutare în adâncime care se opreşte la intervale de L nivele în vederea aprecierii progresării căutării. Dacă creşterea costului după ultima apreciere este de cel mult αL, căutarea continuă; în cazul când creşterea costului este mai mare decât αL, nodul curent este eliminat, programul efectuează un *backtracking* la un nivel mai înalt şi căutarea este reluată. Dacă este identificat un drum regulat (α, L), A_2 furnizează acest drum drept soluţie, costul ei fiind cel mult $d + \alpha (N - d + L)$. Dacă are loc o eşuare, căutarea este reluată de la alt nod de nivel d. Dacă toate 2^d noduri de la nivelul d eşuează să devină rădăcină a unui drum regulat (α, L) către o frunză, A_2 se încheie cu un eşec. Probabilitatea unui eşec poate fi totuşi făcută foarte mică pe calea alegerii convenabile a parametrilor d, α şi L.

Schema căutării este arătată în figura 1.23. şi constă din două componente:

1. Căutarea locală în adâncime, cu marginea adâncimii L, a unor fii (α, L).
2. Căutarea globală în adâncime a membrilor familiei (α, L), căutând un drum ce atinge nivelul N.

Analiza probabilistă a algoritmului A_2 arată (vezi mai jos teorema 1.39.) că, pentru $p > 1/2$ şi valori N mari, costul optimal $C(n, p)$ este foarte apropiat de $\alpha^* N$, unde α^* este o constantă dată de p. În cazul când se caută o soluţie aproape optimală,

https://biblioteca-digitala.ro / https://unibuc.ro

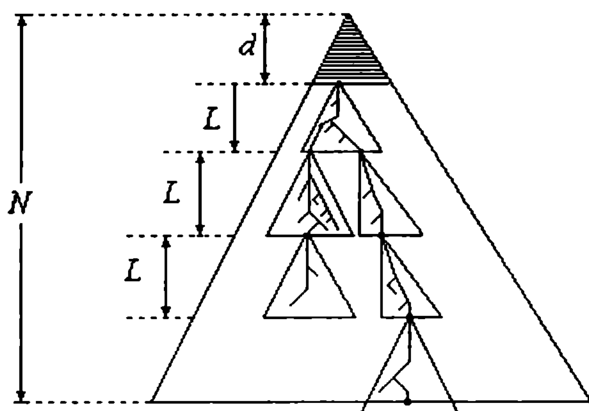


Fig. 1.23.

căutarea poate fi încheiată de îndată ce s-a găsit o frunză cu cost sub αN ($\alpha > \alpha^*$), unde α este ales suficient de aproape de α^*. Din fericire, există mai multe frunze cu costuri între $\alpha^* N$ şi αN, aşa încât căutarea poate fi limitată la o submulţime de asemenea frunze, uşor de identificat. O alegere convenabilă o constituie mulţimea frunzelor accesibile pe drumuri regulate (α, L), dealungul cărora costul creşte cu nu mai mult decât αL, pe fiecare segment succesiv de lungime L. Asemenea drumuri constituie obiective convenabile deoarece violările condiţiei de creştere gradată a costului, spre deosebire de violările condiţiilor impuse costului total al drumului, sunt detectabile printr-o *analiză locală*. Într-adevăr, după cum se va vedea mai departe (vezi teorema 1.42.) algoritmul A_2 are o complexitate temporară liniară în medie şi, dacă parametrii d şi L sunt convenabil aleşi, atunci A_2 identifică o soluţie aproape optimală cu o probabilitate apropiată de unu.

1.2.5.4.1. *Rezultate*

Proprietăţile esenţiale ale algoritmilor A_1 şi A_2 sunt sintetizate în continuare.

Teorema 1.37. [PEA84] Dacă $p < 1/2$, atunci:

$$P[C(N,p) > k] \leq (2p)^{2^{k+1}-2}, \text{ pentru } k = -1, 0, 1, \dots \blacksquare$$

Prin urmare costul optimal este aproape sigur mărginit atunci când $N \to \infty$.

Teorema 1.38. Dacă $p = 1/2$, atunci pentru orice funcţie H monotonă şi mărginită are loc:

$$P[|C(N, 1/2) - \log_2 \log_2 N| > H(N)] \to 0. \blacksquare$$

Aceasta implică faptul următor: costul optimal tinde către $\log_2 \log_2 N$.

https://biblioteca-digitala.ro / https://unibuc.ro

Teorema 1.39. Fie $p > 1/2$ şi $G(\alpha, p) = \left(\frac{p}{\alpha}\right)^{\alpha}\left(\frac{1-p}{1-\alpha}\right)^{1-\alpha}$. Fie α^* mărimea definită de ecuaţia $G(\alpha^*, p) = 1/2$.

Pentru $\alpha < \alpha^*$:

$$P[C(N,p) \leq \alpha N] = O(c_{\alpha}^{-N}) \text{ cu } c_{\alpha} > 1.$$

Pentru $\alpha > \alpha^*$:

$$P[C(N,p) \geq \alpha N] = O(d_{\alpha}^{-N}) \text{ cu } d_{\alpha} > 1. \blacksquare$$

Această teoremă stabileşte proprietăţile: costul optimal creşte proporţional cu N şi factorul de proporţionalitate este apropiat de α^* cu o probabilitate ridicată.

Fie variabilele aleatoare $t_1(N, p)$ şi $t_2(N, p)$ descriind numărul de noduri expandate respectiv de algoritmii A_1 şi A_2 pe un arbore T din mulţimea $T(N, p)$.

Teorema 1.40. Dacă $p < 1/2$, atunci $E[t_1(N, p)] = O(N)$, adică A_1 găseşte o frunză de cost optimal într-un timp mediu liniar. ■

Teorema 1.41. Dacă $p = 1/2$, atunci $E[t_1(N, 1/2)] = O(N^2)$, adică A_1 găseşte o frunză de cost optimal într-un timp mediu pătratic. ■

Teorema 1.42. Dacă $p > 1/2$, atunci orice algoritm ce garantează găsirea unui drum mai ieftin decât $(1 + \varepsilon)\, C(N, p)$ operează în timp exponenţial. Totuşi, pentru orice $\varepsilon > 0$, parametrii d şi L pot fi aleşi astfel încât algoritmul A_2 găseşte o soluţie de cost cel mult $(1 + \varepsilon)\, C(N, p)$ cu o probabilitate apropiată de 1 şi operează în timp mediu liniar, adică $E[t_2(N, p)] = O(N)$. ■

Observaţie. Algoritmul A_1 este o variantă a căutării în lăţime, cu deosebirea că în locul parcurgerii nodurilor de aceeaşi adâncime, căutarea se efectuează pe nivele de noduri de cost egal a drumurilor ce le conectează cu rădăcina.

1.2.6. Complexitatea şi precizia euristicilor admisibile

1.2.6.1. Euristicile interpretate drept surse de informaţie perturbate

1.2.6.1.1. *Modelele simplificate - surse de semnale perturbate*

Metodele euristice se bazează, de obicei, pe modele simplificate ale problemelor studiate. Este deci natural să se accepte că utilizarea unei euristici depinde de proximitatea dintre model şi problema reală studiată. Algoritmul A^* este un model convenabil pentru studiul acestei relaţii în termeni cantitativi, deoarece atât scopul său, minimizarea costului, cât şi euristica sa pot fi exprimate cantitativ. În acest scop, euristicile sunt interpretate drept surse de informaţii perturbate care generează în fiecare

https://biblioteca-digitala.ro / https://unibuc.ro

nod al grafului de căutare informaţia $h(n)$, în locul informaţiei corecte $h^*(n)$. Din acest motiv, diferenţa $h - h^*$ este interpretată drept *eroare* asociată cu funcţia euristică h.

1.2.6.1.2. *Un model probabilistic pentru analiza performanţei unei euristici*

În paragraful 1.2.5.3. a fost prezentată analiza performanţei medii a euristicii distanţă aeriană, presupunând că nodurile sunt distribuite aleator şi uniform în spaţiul euclidian definit de funcţia euristică h. Cu alte cuvinte, se consideră că h şi g definesc o mulţime de coordonate şi se presupune că topologia grafului localităţilor variază aleator, în conformitate cu regularitatea din reţelele tipice de localităţi.

În multe cazuri, nu este accesibilă formularea unei interpretări fizice atât de clare a euristicii h. Drept consecinţă, poate fi mult mai convenabil să se construiască un model probabilistic dintr-un sens opus şi anume: acceptarea topologiei grafului de căutare drept o entitate dată şi tratarea funcţiilor $h(n)$ drept variabile aleatoare distribuite întâmplător în nodurile grafului, dar corelate într-un anumit grad cu distanţele actuale $h^*(n)$.

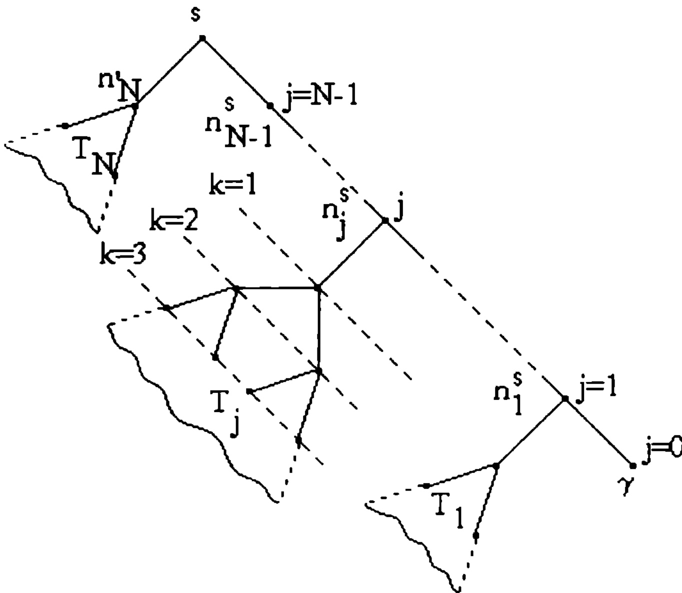


Fig. 1.24.

În acest scop, spaţiul căutărilor este modelat printr-un arbore T nedirecţionat, cu gradul nodurilor egal cu b, având un unic nod start s şi un unic nod soluţie γ, situat la distanţa N de s. Arborele poate fi finit, de adâncime N, sau infinit, fiecare arc având

https://biblioteca-digitala.ro / https://unibuc.ro

un cost simetric unitar. Un arbore de căutare tipic este arătat în figura 1.24., unde, fără a restrânge generalitatea, drumul soluţie $(s, n^s_{N-1}, ..., n^s_j, ..., n^s_1, \gamma)$ este reprezentat în dreapta. Arborii $T_1, ..., T_j, ..., T_N$ sunt subarbori ai lui T, cu rădăcini pe drumul soluţie. Fiecare subarbore T_j este identificat prin rădăcina sa n^s_j de pe drumul soluţie, unde n^s_j este singurul nod din T_j ce are b - 1 urmaşi.

Identificarea nodului γ se efectuează folosind următoarea variantă a algoritmului A^* pentru căutarea pe arbori.

Algoritmul 1.13. (A^* pentru arbori)

1. [Iniţializări] Se marchează s ca aparţinând lui *DESCHIS* şi se calculează $f(s)$.
2. [Alegere nod] Se caută un nod n în *DESCHIS* cu valoare minimală a lui f; în cazul mai multor valori egale ale lui f, alegerea nodului căutat este arbitrară.
3. [Terminare] Dacă n este nodul soluţie γ, atunci STOP.
4. [Modificări şi ciclare] Se marchează n ca aparţinând lui *ÎNCHIS* şi se calculează $f(n'_i)$ pentru fiecare fiu n'_i al lui n. Se marchează fiecare asemenea nod n'_i care nu se află în *ÎNCHIS* sau pentru care $f(n'_i)$ este mai bună ca aparţinând lui *DESCHIS*. Se trece la pasul 2. ■

Algoritmul A^* caută nodul soluţie fără a şti locaţia sa şi nici distanţa N, dar este ghidat de funcţia euristică $h(n)$ care estimează distanţa oricărui nod la nodul soluţie. Deoarece toate arcele se presupun de cost unitar, funcţia f este definită de:

$$f(n) = g(n) + h(n),$$

unde $g(n)$ este adâncimea lui n şi $h(n)$ este estimarea euristică a lui $h^*(n)$, adică numărul de arce ce separă pe n de γ. Se mai presupune că funcţia $h(n)$ este admisibilă, adică $h(n) \leq h^*(n)$, pentru orice n, fapt ce garantează că nodurile de adâncime mai mare decât N nu vor fi expandate.

Modelul probabilistic descris presupune că fiecare $h(n)$ poate fi tratată ca o variabilă aleatoare cu valori în $[0, h^*(n)]$ şi caracterizată prin funcţia de distribuţie:

$$F_{h(n)}(x) = P[h(n) \leq x].$$

Fiind dată o caracterizare a funcţiilor de distribuţie $F_{h(n)}(x)$, problema de rezolvat constă în cercetarea modului în care aceste funcţii influenţează relaţia dintre $E(Z)$, numărul mediu de noduri expandate de A^*, şi lungimea N a drumului soluţie, tratată ca parametru ce defineşte dificultatea problemei.

Procesul de expandare este structurat sub forma arborelui din figura 1.25. Numărul de succesori dintr-o generaţie d + 1 > 1 este egal cu suma descendenţilor tuturor părinţilor de la nivelul d - 1. Prin urmare, mărimea w_{d+1} a generaţiei d + 1 este

https://biblioteca-digitala.ro / https://unibuc.ro

suma a w_d variabile aleatoare $X_{i,d}$, $1 \leq i \leq w_d$ reprezentând numărul de fii fertili ai generației $d + 1$, adică:

$$w_{d+1} = X_{1,d} + \ldots + X_{i,d} + \ldots + X_{w_d,d}.$$

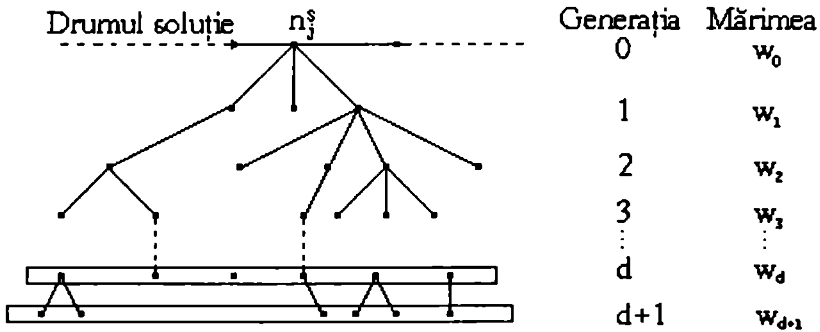


Fig. 1.25.

Presupunând că $X_{i,d}$ și w_d sunt independente, rezultă că media variabilei w_{d+1} este definită de produsul:

$$\overline{w}_{d+1} = E(w_{d+1}) = E(w_d)\, E(X_{i,d}) = \overline{w}_d\, E(X_{i,d}).$$

Dacă se notează cu q_d probabilitatea supraviețuirii unui descendent oarecare al generației d, atunci:

$$E(X_{i,d}) = q_d\, c_d,$$

unde c_d este numărul de descendenți ai generației d ce nu fac parte din drumul soluție. Observând că $c_1 = b - 1$ și $c_d = b$, pentru $d > 1$, rezultă:

$$E(X_{i,d}) = \begin{cases} (b-1)q_1, & \text{pentru } d = 1, \\ bq_d, & \text{pentru } d \geq 2. \end{cases}$$

Ținând seama de expresiile lui $\overline{w}_{d+1}$ și $E(X_{i,d})$ și de faptul că $E(w_0) = \overline{w}_0 = 1$ deoarece generația 0 are cu certitudine un singur descendent, se obține:

$$\overline{w}_1 = (b - 1)\, q_1\, \overline{w}_0 = (b - 1)\, q_1,\ \overline{w}_2 = b\, q_2\, \overline{w}_1,\ \overline{w}_3 = b\, q_3\, \overline{w}_2, \ldots,\ \overline{w}_d = b\, q_d\, \overline{w}_{d-1}.$$

Urmărind șirul precedent se obține succesiv:

$$\overline{w}_d = b\, q_d\, b\, q_{d-1} \ldots b\, q_2\, (b - 1)\, q_1 = (b - 1)\, b^{d-1}\, q_d\, q_{d-1} \ldots q_2\, q_1 = \frac{b-1}{b} b^d \prod_{k=1}^{d} q_k.$$

Numărul mediu al nodurilor expandate de A^* trebuie calculat aplicând analiza precedentă tuturor subarborilor din afara drumului soluție din figura 1.26. Numărul mediu de noduri expandate la adâncimea $d \geq 1$ a unui subarbore T_j este dat de produsul:

$$(b - 1)\, b^{d-1}\, q_{j,d}\, q_{j,d-1} \ldots q_{j,1},$$

unde $q_{j,k}$ reprezintă probabilitatea ca un nod $n_{j,k}$ de adâncime k în T_j să fie expandat o dată intrat în *DESCHIS*, adică:

https://biblioteca-digitala.ro / https://unibuc.ro

$$P[f(n_{j,k}) \le N] \ge q_{j,k} \ge P[f(n_{j,k}) \le N - 1]. \qquad (1.17)$$

Însumând produsele de mai sus pentru toate nivelele d şi toţi subarborii respectivi şi adăugând şi cele N noduri expandate dealungul drumului soluţie se obţine expresia numărului mediu de noduri expandate:

$$E(Z) = N + \frac{b-1}{b} \sum_{j=1}^{N} \sum_{d=1}^{\infty} b^d \prod_{k=1}^{d} q_{j,k}, \qquad (1.18)$$

unde $q_{j,k}$ este mărginit conform relaţiei (1.17). Ţinând seamă de presupunerea că funcţia $h(n)$ este admisibilă, deci că nodurile de adâncime mai mare decât N nu sunt expandate, rezultă că adâncimea maximă a inspectării subarborelui T_j (vezi figura 1.24.) nu depăşeşte adâncimea sa care este egală cu j, deci sumarea după d în (1.18) poate fi efectuată în intervalul $1 \le d \le j$. Rezultă o formă alternativă a lui $E(Z)$ şi anume:

$$E(Z) = N + \frac{b-1}{b} \sum_{j=1}^{N} \sum_{d=1}^{j} b^d \prod_{k=1}^{d} q_{j,k}.$$

1.2.6.2. Dominanţa stohastică pentru euristici admisibile aleatoare

Problema deciderii faptului că o euristică este mai bună decât alta se pune destul de frecvent. Este evident că, dacă o euristică asigură o estimare mai precisă a lui h^*, ea este preferată. Aceasta este ideea teoremei 1.30. care stabileşte că, dacă pentru fiecare nod nonsoluţie al grafului de căutare $h_1(n) < h_2(n)$ şi ambele euristici sunt admisibile, atunci orice nod expandat de A_2^* este expandat de asemenea şi de A_1^*, deci euristica $h_2(n)$ este preferabilă. Totuşi sunt rare cazurile când se dispune de o cunoaştere *a priori* suficientă, ceea ce garantează că $h_1(n) < h_2(n)$ are loc pentru orice nod al arborelui de căutare. Chiar atunci când acurateţea mai bună a lui h_2 este rezultatul utilizării unor proceduri de calcul mai sofisticate decât ale lui h_1, îmbunătăţirea este rar garantată a avea loc în fiecare nod al spaţiului problemei. În general, atunci când $h(n)$ este mai exactă pentru unele noduri, ea poate deveni mai puţin exactă pentru altele. Este natural atunci să se pună problema identificării condiţiilor în care inegalitatea $h_1 < h_2$ este rezonabil probabilă, dar nesatisfăcută ocazional. Formalizarea acestei probleme este prezentată în continuare.

Definiţia 1.10. Fiind date două variabile aleatoare X_1 şi X_2, se zice că X_2 este *statistic mai mare* decât X_1, proprietate notată $X_1 \tilde{<} X_2$, dacă şi numai dacă:

$$P(X_2 > x) \ge P(X_1 > x), \text{ pentru orice } x \in R,$$

sau, echivalent:

$$F_{X_1}(x) \ge F_{X_2}(x), \text{ pentru orice } x \in R. \blacksquare$$

Definiţia 1.11. Fie A_1^* şi A_2^* doi algoritmi ce folosesc respectiv funcţiile h_1 şi h_2. A_2^* este *mai mult informat stohastic* decât A_1^* dacă şi numai dacă $h_1(n) \tilde{<} h_2(n)$, pentru

https://biblioteca-digitala.ro / https://unibuc.ro

orice $n \in T$. Similar, A_2^* se consideră a fi *stohastic mai eficient* decât A_1^* dacă şi numai dacă $Z_2 \tilde{<} Z_1$, unde Z_1 şi Z_2 sunt numerele de noduri expandate de A_1^*, respectiv, A_2^*. ■

Examinarea relaţiilor (1.17) şi (1.18) arată că, dacă pentru fiecare nod n, $h_2(n)$ este mai mare decât $h_1(n)$, atunci h_2 introduce o valoare $E(Z)$ mai mică. Aceasta se datoreşte faptului că fiecare $q_{j,k}$ în cazul lui h_2 este mai mic sau cel mult egal cu factorul corespunzător din h_1. Totuşi, se poate demonstra o proprietate mai tare şi anume: $h_1(n) \tilde{<} h_2(n)$ implică nu numai preferinţa în medie ci şi *preferinţa stohastică.*

Teorema 1.43. Pentru orice distribuţie de erori, dacă A_2^* este stohastic mai informat decât A_1^*, atunci A_2^* este stohastic mai eficient decât A_1^*.

Demonstraţie. Fiind dată o mulţime de variabile aleatoare independente, ordinea $\tilde{<}$ este conservată de adunarea pe această mulţime. Cu alte cuvinte, dacă X_1, Y_1, X_2 şi Y_2 sunt mutual independente, atunci $X_1 \tilde{<} X_2$ şi $Y_1 \tilde{<} Y_2$ implică $X_1+Y_1 \tilde{<} X_2+Y_2$. Fie $Z_1(n)$ şi $Z_2(n)$ numerele de noduri expandate de A_1^*, respectiv, A_2^* în subarborele de rădăcină n. Se poate demonstra că $Z_2(s) \tilde{<} Z_1(s)$. Echivalent, deoarece numerele de noduri expandate în arborii T_j (vezi figura 1.24.) sunt independente, folosind proprietatea de conservare a adunării pentru variabile aleatoare independente, este suficient să se arate că $Z_2(n'_j) \tilde{<} Z_1(n'_j)$ pentru un arbore arbitrar T_j. Fie n orice nod la adâncimea d a arborelui; este suficient să se arate că $Z_2(n) \tilde{<} Z_1(n)$, pentru orice asemenea n. Demonstraţia se efectuează pe calea inducţiei după d, de jos în sus.

Pentru valori suficient de mari ale lui d afirmaţia este adevărată deoarece $Z_2(n)=Z_1(n)=0$, pentru toate nodurile n de la nivelul d. Presupunând că $Z_2(n) \tilde{<} Z_1(n)$, pentru toate nodurile n de la nivelul d, rămâne de arătat că relaţia precedentă are loc şi pentru toate nodurile n de la nivelul d - 1. Fie n_1, ..., n_k, ..., n_b descendenţii direcţi ai nodului n la nivelul d-1, conform figurii 1.26. Pentru toţi întregii $x \geq 0$ are loc relaţia:

$$P(Z(n) \leq x) = \begin{cases} 1, & \text{dacă } A^* \text{ nu expandează pe } n \\ P\left(\sum_k Z(n_k) \leq x-1\right), & \text{dacă } A^* \text{ expandează pe } n, \end{cases}$$

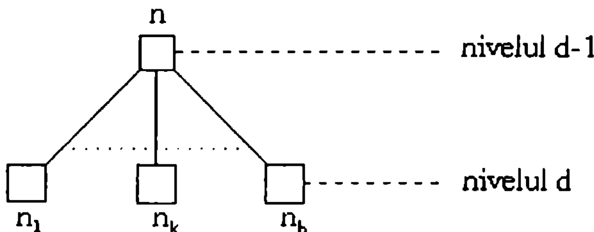


Fig. 1.26.

sau:

$$F_{Z(n)}(x) = P(Z(n) \leq x) = [1 - P(n\ exp)] + P(n\ exp)\ P[(\sum_k Z(n_k) \leq x\text{-}1) \mid n\ exp], \quad (1.19)$$

unde *exp* este prescurtarea lui expandat.

Deoarece $h_1(n) \lesssim h_2(n)$, pentru orice n şi drumul de la s la n este unic, rezultă că:

$$P(A_1^* \text{ expandează pe } n) \geq P(A_2^* \text{ expandează pe } n). \quad (1.20)$$

Într-adevăr, în acest caz $f_1(n) \lesssim f_2(n)$ şi atunci:

$$P(f_1(n) \leq N) \geq P(f_2(n) \leq N).$$

În virtutea inducţiei:

$$Z_2(n_k) \lesssim Z_1(n_k), \text{ pentru toţi } k,$$

şi deoarece $Z(n_1), ..., Z(n_k), ..., Z(n_b)$ sunt condiţional independente:

$$\sum_k Z_2(n_k) \lesssim \sum_k Z_1(n_k),$$

sau:

$$P[\sum_k Z_2(n_k) \leq x - 1] \leq P[\sum_k Z_1(n_k) \leq x - 1]. \quad (1.21)$$

Ţinând seama de (1.20) şi (1.21) în (1.19) se poate arăta că $F_{Z_1(n)}(x) \leq F_{Z_2(n)}(x)$ sau că:

$$\left.\begin{matrix} u_1 \geq u_2 \\ v_1 \leq v_2 \end{matrix}\right\} \Rightarrow 1 - u_1 + u_1 v_1 \leq 1 - u_2 + u_2 v_2, \quad (1.22)$$

unde: $u_j = P(A_j^* \text{ expandează pe } n), \quad j = 1, 2,$

$v_j = P[\sum_k Z_j(n_k) \leq x - 1] \quad j = 1, 2.$

Implicaţia (1.22) se poate reduce la:

$$\left.\begin{matrix} u_1 \geq u_2 \\ v_1 \leq v_2 \end{matrix}\right\} \Rightarrow (u_1 - u_2)(1 - v_1) + (v_2 - v_1) u_2 \geq 0,$$

care este uşor de verificat ţinând seama că u_2 şi v_1 sunt probabilităţi şi, prin urmare, $v_1 \leq 1$ şi $u_2 \geq 0$.

Rezultă că $F_{Z_1(n)}(x) \leq F_{Z_2(n)}(x)$, pentru toate nodurile şi nivelele unui subarbore oarecare al lui T. În concluzie, $Z_1(s) \lesssim Z_2(s)$, fapt ce demonstrează teorema. ◄

1.3. Sinteza algoritmilor euristici

În acest paragraf vom descrie principalele tipuri de abordare ale unor probleme care, pentru soluţionare, conduc la elaborarea unor algoritmi euristici.

1.3.1. Clasificarea metodelor euristice

Metodele euristice pot fi clasificate, în general, în două clase: metode de identificare a unei prime soluţii, respectiv, metode iterative, de îmbunătăţire pas cu pas a unei soluţii cunoscute. O clasificare *în mare* a metodelor euristice poate fi:

https://biblioteca-digitala.ro / https://unibuc.ro

1. Metode de identificare a unei prime soluţii:
   1.1. cu grad de libertate descrescător:
      1.1.1. şi procedee de alegere prioritară;
      1.1.2. şi procedee de relaxare a alegerii;
   1.2. cu grad de libertate crescător;
2. Metode iterative:
   2.1. cu alegere necondiționată;
   2.2. cu alegere condiţionată.

Metodele de identificare a unei prime soluţii utilizează arborii de decizie, dar nu asigură faptul că mulţimea soluţiilor identificate conţine eventual şi soluţia optimală. Parcurgerea metodei de identificare este reprezentabilă printr-un arbore, în cadrul căruia nodurile reprezentând mulţimi de soluţii sunt ramificate succesiv. În multe metode de identificare a unei prime soluţii, numărul de ramificaţii ale nodurilor arborelui scade succesiv. Aceste metode se numesc cu *grad de libertate descrescător.*

Cele mai simple metode de identificare cu grad de libertate descrescător utilizează reguli de prioritate pe baza cărora are loc selecționarea ramurilor ce urmează a fi parcurse. Asemenea metode sunt foarte rapide, dar au desavantajul că adesea generează parcurgeri ce conduc la soluţii puţin eficiente.

Soluţii mai bune se pot obţine frecvent dacă există posibilităţi de relaxare a problemei date care să permită reducerea complexităţii arborelui de identificare.

În cadrul metodelor de identificare cu grad de libertate crescător, numărul de ramificaţii creşte o dată cu adâncimea arborelui de decizie. Creşterea numărului de ramificări permite, după parcurgerea lor, corectarea deciziilor precedente. Drept consecinţă metodele cu grad de libertate crescător asigură frecvent obţinerea de soluţii esenţial mai bune decât cele furnizate de metodele cu grad de libertate descrescător.

Metoda de iteraţie cu alegere necondiţionată constă în continuarea succesivă a căutării soluţiilor, fără utilizarea vreunui criteriu de apreciere a eficienţei căutării. Caracteristic pentru această metodă este numărul mare de soluţii identificate cu ajutorul unei proceduri eficiente în ceea ce priveşte timpul de calcul.

În cadrul metodei cu alegere condiţionată, sunt utilizate teste (sau filtre) care evită generarea de noi soluţii care nu pot fi mai eficiente decât o soluţie precedentă.

1.3.1.1. Morfologia metodelor de identificare a unei prime soluții

În procesul de identificare a unei prime soluţii participă o mulţime de decizii ce pot fi împărţite în următoarele patru grupe:

https://biblioteca-digitala.ro / https://unibuc.ro

- Decizia privind ce submulţimi de soluţii urmează să fie scindate primele si decizia privind ordinea în care urmează să fie parcurse nodurile arborelui de decizie.
- Decizia privind modul în care submulţimea de soluţii urmează a fi scindată şi identificarea ramificării, adică numărul de noduri următoare şi conţinutul lor.
- Decizia privind modul în care se evaluează un nod unic din submulţimea de soluţii, adică felul în care se stabileşte calitatea nodului unic.
- Decizia, luată pe baza evaluării, privind submulţimile de soluţii care ar urma să fie examinate ce pot fi ignorate, deci nodurile aferente eliminate; de asemenea alegerea acelor submulţimi de soluţii ce pot participa la o următoare scindare.

1.3.1.2. Morfologia metodelor iterative

Destul de frecvent, soluţiile identificate cu metoda precedentă nu satisfac condiţiile impuse, adică se abat prea mult de la soluţiile căutate. Dacă nu se poate ori nu se doreşte ameliorarea metodei anterioare, rămâne varianta folosirii metodelor iterative. Cu ajutorul lor se încearcă îmbunătăţirea unei soluţii anterioare, fără garantarea obţinerii soluţiei dorite, eventual optimale.

Esenţa metodelor iterative constă în căutarea unei soluţii mai bune într-o vecinătate (definită) a unei soluţii cunoscute. Deciziile din cadrul metodelor iterative frecvent întâlnite sunt cele relative la definiţia vecinătăţii soluţiilor sau la organizarea metodelor de căutare.

În lucrarea [MÜL76] sunt prezentate pe larg cele două metodologii şi multiplele proceduri folosite în cadrul metodelor euristice, fapt ce permite identificarea multiplelor variante ale algoritmilor euristici de rezolvare a unei probleme.

1.3.2. Metode exacte pentru sinteza algoritmilor

În acest paragraf, vom prezenta metode *clasice* pentru generarea algoritmilor, cu specificarea caracteristicilor algoritmilor obţinuţi. Aceste metode sunt utile în elaborarea algoritmilor euristici atât în determinarea unor metode de determinare a soluţiei unei subprobleme cât şi în determinarea unor soluţii aproximative, plecare, pentru relaxările problemelor pentru care se caută soluţii euristice.

1.3.2.1. Metoda Greedy

Această metodă se aplică, în general, la probleme pentru care se caută determinarea unei submulţimi a unei mulţimi cu anumite caracteristici. Dintre diferitele submulţimi ce îndeplinesc anumite condiţii, numite *soluţii posibile*, se alege una singură, numită *soluţie optimală*. Metoda Greedy se aplică în cazul în care orice

https://biblioteca-digitala.ro / https://unibuc.ro

submulțime a unei soluții posibile este o soluție posibilă. În particular, se consideră că mulțimea vidă este o soluție posibilă. Presupunând că mulțimea elementelor pentru care se aplică metoda Greedy este $A = \{a_1, a_2, ..., a_n\}$ și că rezultatul este mulțimea B, metoda corespunde la algoritmul următor:

Algoritmul 1.14. ***GREEDY(A, n, B)***

1. [Inițializare] Se face $B = \varnothing$.
2. [Terminare] Dacă $A = \varnothing$, atunci furnizează B și STOP.
3. [Alegere] Se alege un element $x \in A$ și se face $A = A - \{x\}$.
4. [Adăugare] Dacă sunt verificate condițiile pentru a fi soluție, se face $B = B \cup \{x\}$.
5. [Ciclare] Se merge la pasul 2. ■

În această metodă sunt de rezolvat două probleme și anume problema determinării unor condiții pe care trebuie să le îndeplinească mulțimea B pentru a fi siguri că, prin completarea ei, se ajunge la soluția optimă și definirea unor criterii de alegere pentru determinarea unei soluții eficiente.

Exemple de aplicare a metodei Greedy: metoda simplex din programarea liniară, problema rucsacului, interclasarea mulțimilor ordonate de elemente, memorarea fișierelor pe benzi magnetice, programarea activităților independente etc.

1.3.2.2. Metode arborescente

Aplicarea metodei arborescente se face pentru probleme de determinare a elementelor $x \in S$, unde S este o mulțime de elemente posibile, care minimizează o funcție $f(x)$. Se consideră o limită superioară F_1 a valorilor $f(x)$ care restrânge mulțimea elementelor soluție. Vom nota cu X mulțimea soluțiilor, adică $X=\{x \in S \mid f(x)<F_1\}$, cu f^* valoarea optimală dată de expresia $f^* = \min_{x \in X} f(x)$ și cu X^* mulțimea soluțiilor optimale, $X^*=\{x \in X \mid f(x)=f^*\}$. Dacă nu există soluții, deci $X = \varnothing$, atunci se consideră $f^* = +\infty$. Vom nota cu $\hat{f}$ o evaluare prin exces pentru f^*. Inițial se poate lua $\hat{f} = F_1$.

Numim *arborescență validă pentru* $\hat{f}$ o structură arborescentă care are rădăcina $R = X$, unde X este mulțimea soluțiilor, nodurile sunt submulțimi ale lui X, succesorii direcți $S_1, S_2, ..., S_p$ ai unui nod S sunt astfel încât $S = S_1 \cup S_2 \dots \cup S_p$, unele noduri sunt marcate cu "*eliminat*" și dacă se elimină un nod, atunci se elimină toți descendenții acelui nod, dacă există o soluție optimală și dacă $\hat{f} > f^*$, atunci cel puțin o soluție optimală nu aparține nodurilor eliminate și pentru fiecare nod S se determină o evaluare $g(S)$ prin lipsă față de elementele lui S care verifică relația $g(S) \le f(x)$ pentru orice $x \in S$.

https://biblioteca-digitala.ro / https://unibuc.ro

Restricţia unei arborescenţe valide la nodurile neeliminate este tot o arborescenţă validă care se numeşte *arborescenţa curentă*.

Se poate demonstra următoarea proprietate: dacă există o soluţie optimală şi dacă $\hat{f} > f^*$, atunci soluţia optimală va aparţine nodurilor terminale ale arborescenţei curente.

Pe mulţimea arborescenţelor valide definim următoarele operaţii:

- *Trunchiere* (*eliminare*): Se pleacă de la o arborescenţă A_1 validă pentru $\hat{f}$ şi, prin eliminarea unui nod şi a descendenţilor lui, se determină o arborescenţă A_2. Trunchierea este definită numai dacă arborescenţa rezultată este validă pentru $\hat{f}$.
- *Separare*: Se pleacă de la o arborescenţă A_1 şi se obţine o arborescenţă A_2 prin introducerea la un nod terminal S a succesorilor direcţi $S_1, S_2, ..., S_p$ astfel încât $S = S_1 \cup S_2 \cup ... \cup S_p$ şi se face $g(S_1) = g(S_2) = ... = g(S_p) = g(S)$. Se demonstrează că, dacă A_1 este o arborescenţă validă pentru $\hat{f}$, atunci şi A_2 este validă pentru $\hat{f}$.
- *Evaluare*: Se calculează pentru un nod S din arborele curent o evaluare prin lipsă $h(S)$ şi dacă $h(S) < g(S)$, atunci se face $g(S) = h(S)$.
- *Ajustare*: La o arborescenţă validă pentru $\hat{f}$, dacă se găseşte o soluţie de valoare $\hat{g} < \hat{f}$, atunci se face $\hat{f} = \hat{g}$. Se demonstrează că arborescenţa este validă pentru $\hat{g}$.

Metoda arborescentă pentru o problemă P este algoritmul prin care se pleacă de la arborescenţa ce conţine un singur nod, rădăcina, cu estimarea $\hat{f} = F_1$ şi aplicând operaţii de trunchiere, separare, evaluare şi ajustare se obţine un şir de arborescenţe valide. Algoritmul se termină când toate nodurile sunt eliminate.

Dacă la terminarea aplicării metodei arborescente se obţine $\hat{f} = F_1$, atunci problema $\min f(x)$ pentru $x \in S$ şi $f(x) < F_1$ nu are soluţie. Altfel, ultima valoare $\hat{f}$ este egală cu valoarea f^* a unei soluţii optimale.

Pentru a obţine o *evaluare prin lipsă*, se ţine seamă de următoarele proprietăţi:

Teorema 1.44. Dacă A este o arborescenţă validă pentru $\hat{f}$ şi dacă un nod S este astfel încât $g(S) \geq \hat{f}$, atunci arborescenţa obţinută plecând de la A prin eliminarea lui S şi a descendenţilor săi este o arborescenţă validă pentru $\hat{f}$. ■

Teorema 1.45. Dacă A este o arborescenţă validă pentru $\hat{f}$ şi dacă se demonstrează că soluţia optimală a unui nod S are o valoare superioară lui $\hat{f}$ sau că S este vidă, atunci arborescenţa obţinută eliminând pe S şi descendenţii săi este tot validă pentru $\hat{f}$. ■

Se poate obţine o *evaluare prin exces* dacă se obţine o valoare iniţială aplicând o *euristică* şi apoi se ameliorează această valoare prin tehnici de vecinătăţi.

În luarea deciziilor de trunchiere şi eliminare se aplică criterii date de proprietăţile următoare numite *proprietăţi de dominanţă*.

https://biblioteca-digitala.ro / https://unibuc.ro

Teorema 1.46. Fie S un nod terminal şi $S' \subset S$; dacă S' conţine o soluţie optimală de fiecare dată când S conţine o asemenea soluţie, atunci se aplică o separare a lui S în S' şi $S - S'$ şi apoi se poate elimina $S - S'$. ■

Teorema 1.47. Dacă S_1 şi S_2 sunt două noduri neeliminate şi dacă are loc relaţia $\min_{x \in S_1} f(x) \leq \min_{x \in S_2} f(x)$, atunci se poate elimina S_2. ■

Pentru separare se pot utiliza două tipuri de strategii:

- *strategia de separare în adâncime* în care se consideră un nivel curent de pe care se selectează un nod din care se trece la un nod de pe nivelul imediat următor care nu a fost evaluat; în cazul când nu mai există astfel de noduri, se revine la nodul din nivelul anterior şi se explorează o nouă variantă; această metodă utilizează pentru gestionare o stivă;
- *strategia de separare în lăţime* în care se consideră pentru un nod curent toţi succesorii lui direcţi, parcurgerea arborescenţei făcându-se nivel cu nivel; această metodă utilizează pentru gestionare o coadă.

La nivelul fiecărui nod S al arborescenţei se poate defini problema P' *derivată* din P care se enunţă astfel: să se minimizeze valoarea $f(x)$ pentru $x \in S$ şi $f(x) < \hat{f}$. Această problemă o mai numim *problemă locală* pentru nodul S. Problema locală corespunzătoare rădăcinii coincide cu problema P.

Dacă problema locală P' corespunzătoare lui S nu are soluţie, atunci se elimină S. Dacă soluţia optimală a problemei locale este $\hat{g}$, atunci se elimină S cu eventuale operaţii de ajustare. Dacă problema locală P' nu se poate rezolva, atunci se aplică o separare a lui S.

O condiţie suficientă pentru convergenţa metodei arborescente este existenţa unei funcţii h cu valori numere naturale astfel încât $h(X)$ să fie un număr finit şi dacă S_1 este un succesor direct al lui S, atunci $0 < h(S_1) < h(S)$ şi oricare nod are un număr de cel mult k succesori direcţi pentru un k dat.

1.3.2.3. Metoda backtracking

Această metodă de elaborare a unui algoritm se aplică problemelor în care soluţia se poate pune sub forma unui vector, fecare element al vectorului putând să ia o mulţime finită de valori. Între elementele din vectorul soluţie se stabilesc relaţii numite *condiţii interne*, prin care se poate stabili dacă o anumită combinaţie poate fi acceptată drept soluţie sau nu. O problemă poate să admită, eventual, mai multe *soluţii rezultat*. În metoda backtracking se încearcă evitarea generării tuturor combinaţiilor posibile pentru a determina soluţiile rezultat. Pentru a determina o soluţie $(x_1, x_2, ..., x_n)$, presupunând că au fost determinate valori pentru $x_1, x_2, ..., x_{k-1}$,

https://biblioteca-digitala.ro / https://unibuc.ro

se atribuie o valoare pentru x_k şi sunt verificate condiţii prin care se determină că s-a obţinut o soluţie, în cazul $k = n$, sau se poate completa cu valorile $x_{k+1}, ..., x_n$ pentru a obţine o soluţie. Aceste condiţii se numesc, în cazul când nu s-a obţinut deja o soluţie, *condiţii de contiuare*. Dacă valoarea dată lui x_k nu este acceptabilă, se încearcă o altă valoare şi, dacă nu mai sunt valori posibile pentru x_k, se revine la x_{k-1} şi se încearcă o nouă valoare. Algoritmul se termină când nu se mai poate atribui nici o valoare posibilă lui x_1. Metoda backtracking se poate descrie sub forma următoare:

Algoritmul 1.15. *BACKTRACK*(x, n)

1. [Iniţializare] Se face $k = 1$.
2. [Verificare soluţie] Dacă $k > n$, atunci se prelucrează soluţia ($x_1, x_2, ..., x_n$) şi se merge la pasul 5.
3. [Testare continuare] Dacă nu mai sunt valori posibile neexplorate pentru x_k, atunci se merge la pasul 5.
4. [Atribuire şi continuare] Se atribuie o valoare posibilă pentru x_k, apoi se face $k = k + 1$ şi se merge la pasul 2.
5. [Revenire şi continuare] Se face $k = k - 1$. Dacă $k > 0$, atunci se trece la pasul 3, altfel STOP. ■

Prin prelucrarea unei soluţii, la pasul 2, se înţelege mulţimea de operaţii care se fac pentru o soluţie găsită cum ar fi tipărirea, numărarea, memorarea, calculul valorii unei funcţii particulare etc. Prin valori posibile neexplorate, la pasul 3, se înţeleg acele valori ale lui x_k, ce nu au fost încă folosite cu combinaţia curentă a elementelor $x_1, ..., x_{k-1}$ şi pentru care sunt îndeplinite condiţiile interne.

Exemple de aplicare a metodei backtracking: determinarea unor submulţimi de elemente de sumă dată, partiţionarea unui întreg, problema celor 8 dame, colorarea grafurilor, generarea submulţimilor unei mulţimi etc.

1.3.2.4. Metoda branch and bound

Această metodă este asemănătoare cu metoda backtracking. Deosebirea constă în faptul că, în această metodă sunt limitate valorile posibil de atribuit elementelor pe baza informaţiilor obţinute anterior luării deciziei. Astfel mulţimea elementelor explorate pentru determinarea soluţiilor devine mai mică şi eficacitatea algoritmului creşte. Determinarea dacă pentru o valoare a unui element este necesară continuarea completării ei sau nu pentru a ajunge la soluţie se face prin calculul valorii unei funcţii, numită *criteriu de evaluare*, şi verificarea apartenenţei acestei valori la o mulţime posibilă de valori pentru soluţie.

Exemple de aplicare a metodei branch and bound: determinarea drumurilor minime ale unui graf, jocul "n^2 - 1" (n = 3, 4, ...).

https://biblioteca-digitala.ro / https://unibuc.ro

1.3.2.5. Metoda divide et impera

Metoda divide et impera constă în împărţirea unei probleme în două sau mai multe subprobleme de dimensiuni mai mici, care la rândul lor sunt divizate eventual în alte subprobleme, procesul continuâd până în momentul în care se ajunge la o mulţime de subprobleme suficient de mici pentru a obţine soluţiile corespunzătoare lor şi apoi se asamblează soluţiile subproblemelor pentru a obţine o soluţie pentru problema iniţială urmând calea inversă descompunerilor.

Exemple de aplicare a metodei divide et impera: sortarea prin interclasare, căutarea în fişiere, căutarea în arbori de sortare, problema turnurilor din Hanoi etc.

1.3.2.6. Metoda programării dinamice

Metoda de programare dinamică a fost concepută de Bellman, fiind publicată în anul 1957. Această metodă a fost aplicată pentru rezolvarea unor probleme privind procesele industriale, gestionarea stocurilor, probleme de investiţii, planificări economice şi alte probleme de optimizări combinatoriale.

Metoda programării dinamice se aplică problemelor în care soluţia se obţine ca rezultat al unui şir de decizii, fiecare decizie depinzând de deciziile luate anterior, în fiecare moment de decizie pot fi explorate mai multe variante de decizie posibile. În luarea deciziilor se ţine seama de *principiul optimalităţii* care stabileşte că orice subşir de decizii al unui şir optimal de decizii este optimal. În linii mari, metoda programării dinamice constituie determinarea unui şir de stări $s_0, s_1, ..., s_n$, unde s_0 este starea iniţială şi s_n este starea finală, din care rezultă soluţia optimală, şi o mulţime de decizii $d_1, d_2, ..., d_n$. Fiecare decizie d_i determină modul de tranziţie din starea s_{i-1} în starea s_i. Deciziile pot fi determinate în ordinea $d_n, ..., d_1$ (*metoda înainte*) sau în ordinea $d_1, ..., d_n$ (*metoda înapoi*). Vom preciza în cele ce urmează acest proces.

Metoda programării dinamice constă în determinarea unui şir de decizii, numit *politică*, prin care se asigură evoluţia unui proces. Determinarea unei decizii a unei probleme de dimensiune N se face în funcţie de deciziile ce sunt luate pentru probleme de dimensiuni $N - 1, N - 2, ..., 2, 1$. De fiecare dată se consideră cea mai bună decizie, aceasta asigurând o soluţie optimă. Deciziile sunt luate secvenţial. Definim formal acest proces de decizie în continuare.

Numim *proces de decizie secvenţială* o tripletă de forma (X, U, g), unde X este spaţiul stărilor, U este o mulţime numită *spaţiul deciziilor* şi

$$g: X \times U \times \mathrm{N} \to X \cup \{\varepsilon\}$$

este *funcţia de tranziţie*. Vom nota cu $\Delta(x, k) = \{u \mid g(x, u, k) \neq \varepsilon\}$ mulţimea deciziilor posibile la momentul k pentru starea x. *Traiectoria unei particule* se defineşte în felul

https://biblioteca-digitala.ro / https://unibuc.ro

următor: se consideră o stare iniţială $x(0)$ şi dacă se cunoaşte starea $x(k)$ la momentul k, se consideră o decizie $u \in \Delta(x(k), k)$ şi se obţine $x(k + 1) = g(x(k), u, k)$ starea la momentul $k + 1$. Se presupune că la fiecare etapă există un număr finit de decizii posibile care nu depăşeşte o valoare N. Aceasta se numeşte *ipoteza Markov* şi un sistem care verifică această ipoteză se numeşte *orizont de procese de decizie secvenţială* şi se notează (X, U, g, N).

Mulţimea stărilor accesibile la momentul k, notată X_k, se defineşte astfel: $x \in X_k$ dacă şi numai dacă există un şir de decizii $(u(0),...,u(k\text{-}1))$ astfel încât $u(0) \in \Delta(x(0),0)$, $x(1) = g(x(0),u(0),0)$, $u(1) \in \Delta(x(1),1)$, $x(2) = g(x(1),u(1),1)$, ..., $u(k\text{-}1) \in \Delta(x(k\text{-}1),k\text{-}1)$, $x(k) = g(x(k\text{-}1),u(k\text{-}1),k\text{-}1)$.

Se numeşte *subpolitică de ordin* $n \leq N$ a orizontului de procese de decizie secvenţială (X,U,g,N) pentru starea x a lui X_{N-n} perechea $(x,(d_0,d_1,...,d_{n-1}))$, cu $d_0 \in \Delta(x,N\text{-}n)$, $d_1 \in \Delta(x,N\text{-}n\text{+}1)$, cu $x_1=g(x,d_0,N\text{-}n)$, $d_2 \in \Delta(x,N\text{-}n\text{+}2)$, cu $x_2=g(x_1,d_1,N\text{-}n\text{+}1)$, ..., $d_{n-1} \in \Delta(x_{n-1},N\text{-}1)$, cu $x_{n-1}=g(x_{n-2},d_{n-2},N\text{-}2)$. Numim *politică* o subpolitică de ordin N.

Pentru compararea politicilor se consideră un *criteriu* care este o funcţie $J(x,(d_0,d_1,...,d_{n-1}))$. Vom spune că $(x,(d_0^*,...,d_{n-1}^*))$ este *subpolitică optimală* dacă pentru orice subpolitică $(x,(d_0,d_1,...,d_{n-1}))$ are loc inegalitatea:

$$J(x, (d_0^*, ..., d_{n-1}^*)) \leq J(x, (d_0, d_1, ..., d_{n-1})).$$

Un criteriu J se numeşte *criteriu separabil* dacă există o funcţie $r_k(x,u)$, pentru $x \in X_k$ şi $u \in \Delta(x,k)$, $0 \leq k \leq N\text{-}1$, numită *funcţie de cost imediat*, funcţiile R_n de n variabile reale, $1 \leq n \leq N$ şi funcţii φ_n de două variabile reale, monoton nedescrescătoare în raport cu a doua variabilă, $2 \leq n \leq N$, astfel încât pentru orice n, $1 \leq n \leq N\text{-}1$ şi orice subpolitică de ordin n $(a,(d_0,d_1,...,d_{n-1}))$ au loc relaţiile $J(a,(d_0,d_1,...,d_{n-1}))=R_n(\rho_0,...,\rho_{n-1})=\varphi_n(\rho_0, R_{n-1}(\rho_1,...,\rho_{n-1}))$, cu $\rho_0=r_{N-n}(a,d_0)$, $\rho_1=r_{N-n+1}(x_1,d_1)$, $\rho_2=r_{N-n+2}(x_2,d_2)$,...,$\rho_{n-1}=r_{N-1}(x_{n-1}, d_{n-1})$, $x_1=g(a,d_0,N\text{-}n)$, $x_2=g(x_1,d_1,N\text{-}n\text{+}1)$,..., $x_{N-1}=g(x_{N-2},d_{N-2},N\text{-}2)$.

Exemple de criterii separabile cel mai des utilizate sunt următoarele: $\sum_{i=0}^{N-1} r_i$, $\prod_{i=0}^{N-1} r_i$, cu $r_i \in R^{*+}$, $\max\{r_i \mid i = 0, 1, ..., N - 1\}$ şi $\min\{r_i \mid i = 0, 1, ..., N - 1\}$.

Se numeşte *principiul optimalităţii* proprietatea că orice subpolitică a unei politici optimale este optimală.

Vom nota în continuare cu $\overline{u}(n,x)$ o subpolitică asociată unei stări iniţiale $x \in X_{N-n}$ şi orizont n, $\overline{u}(n,x)=(u(N\text{-}n),u(N\text{-}n\text{+}1),...,u(N\text{-}1))$ şi cu $\overline{u}^*(n,x)$ o subpolitică optimală în raport cu un criteriu J şi având valoarea $J(x,\overline{u}^*(n,x))$. Din definiţiile anterioare rezultă următoarea relaţie:

$$\overline{u}(n,x) = (u(N - n), \overline{u}(n - 1, g(x, u(N - n), N - n))).$$

Algoritmul de programare dinamică poate să fie descris în felul următor.

https://biblioteca-digitala.ro / https://unibuc.ro

Algoritmul 1.16. ***PROGDYN(X, U, g, J)***

1. [Ciclare k] Pentru $k = N - 1, N - 2, ..., 0$ se execută pasul 2, apoi STOP.
2. [Ciclare stare] Pentru orice stare x din X_k se execută pasul 3.
3. [Calcule] Dacă $\Delta(x, k) \neq \varnothing$, atunci se face:

$$f^*(x, k) = \min_{u \in \Delta(x,k)} \left\{ J\left(x, u, \bar{u}^*\left(N - k - 1, g(x, u, k)\right)\right) \right\};\ v^*(x, k) = u,$$

unde $u \in \Delta(x, k)$ este o decizie pentru care se realizează minimul precedent; (dacă există mai multe şi traiectoria optimală trece prin x la nivelul k, atunci există mai multe politici optimale); se face:

$$\bar{u}^*(N - k, x) = \left(v^*(x, k), \bar{u}^*\left(N - k - 1, g\left(x, v^*(x, k), k\right)\right)\right). \blacksquare$$

Exemple de aplicare a metodei programării dinamice: determinarea drumurilor de lungime minimă în grafuri, determinarea arborilor optimi de căutare, programarea activităţilor etc.

1.3.3. Metode generale de construire a unor euristici

Una din metodele cele mai frecvente de elaborare a algoritmilor euristici se poate descrie pe scurt în felul următor. Se descompune procesul de căutare a soluţiilor în mai multe etape succesive pentru care sunt determinate soluţii pe care le numim *soluţii locale*. Pe baza soluţiilor locale se determină soluţiile problemei iniţiale. Prin exemple, se poate vedea că nu întotdeauna se obţin soluţiile optimale ale problemei iniţiale. Dacă procedeul rezultat produce întotdeauna soluţia optimală, se obţine un *algoritm exact*. Altfel, dacă se găseşte un contraexemplu sau nu s-a găsit o demonstraţie prin care se arată optimalitatea algoritmului propus, se obţine un *algoritm euristic*.

În elaborarea algoritmului euristic, sunt detectate mai întâi toate condiţiile pe care trebuie să le îndeplinească o soluţie optimală, după care, în funcţie de anumite criterii (facilitatea de verificare, necesitatea verificării pentru soluţii posibile, posibilitatea unor relaxări etc.), se determină o mulţime de clase de condiţii.

1.3.3.1. Euristici folosind metoda Greedy

Cea mai mare parte a algoritmilor euristici se poate încadra în această clasă. Prima idee care este exploatată în soluţionarea unei probleme este determinarea unor etape în care *câştigul* să fie cât mai mare posibil. Intuitiv aceasta duce la obţinerea unei soluţii acceptabile într-un număr mai mic de paşi. Riscurile sunt de două feluri: ori o traiectorie care să aibă câştiguri mari la început dar mult mai mici decât alte traiectorii de la un moment dat, ori să fie o traiectorie falsă, prin care să nu ne putem

https://biblioteca-digitala.ro / https://unibuc.ro

apropia oricât de mult de o soluţie optimală. În primul caz, riscul este de a obţine un timp mai mare de lucru, dar în al doilea caz riscul este mult mai mare deoarece cu toate eforturile făcute nu se obţine o soluţie satisfăcătoare. De o importanţă deosebită este alegerea funcţiei de calcul a câştigului.

Exemple de algoritmi euristici bazaţi pe metoda Greedy: metoda gradientului, metoda tangentei în determinarea soluţiilor unor ecuaţii, metoda calculului adresei în căutarea în mulţimi ordonate, metoda de soluţionare a problemei rucsacului (vezi algoritmul 1.4.), metoda de soluţionare a problemei comisvoiajorului (vezi algoritmul 1.2.), *LPT*-planificarea (vezi definiţia 1.1.) etc.

1.3.3.2. Euristici folosind metoda backtracking

Acest tip de euristici se folosesc în cazul în care determinarea aproximării soluţiei pentru o problemă dată se determină dintr-o mulţime finită de elemente ce trebuiesc explorate în totalitatea lor. Prin anumite criterii aplicate spaţiului pe care se lucrează sau prin localizarea unor stări iniţiale se poate face această reducere a posibilităţilor de variaţie a variabilelor. Procesul de explorare este analog metodei exacte de tip backtracking.

Exemple de algoritmi euristici bazaţi pe metoda backtracking: metoda de soluţionare a problemei de programare liniară întreagă, metoda înjumătăţirii intervalului şi metoda coardei pentru determinarea soluţiei unei ecuaţii etc.

1.3.3.3. Euristici folosind metoda branch and bound

În algoritmii branch and bound o *problemă parţială*, referită şi ca *nod*, P_i este descompusă într-o mulţime $S(P_i)$ cu $|S(P_i)| < \infty$. O problemă $P_j \in S(P_i)$ se numeşte *fiu* al lui P_i. Fie o problemă de *minimizare* în care $f(P_i)$ este valoarea optimală a lui P_i. De obicei $f(P_i)$ nu se cunoaşte a priori, dar pentru fiecare P_i se calculează o margine inferioară $g(P_i)$. În afară de proprietatea $g(P_i) \leq f(P_i)$ se mai presupune că:

$$g(P_i) \leq g(P_j), \text{ pentru } P_j \in S(P_i).$$

La calculul lui $g(P_i)$ se poate uneori conchide că $g(P_i) = f(P_i)$. Mulţimea acestor P_i se notează cu G şi descompunerea ei nu mai trebuie continuată, conform cu *testul G* în algoritmul prezentat mai departe. Dacă z este valoarea minimă a lui $f(P_i)$, pentru toate $P_i \in G$ testate, atunci P_i poate fi terminată dacă $g(P_i) \geq z$, deoarece P_i nu poate furniza o valoare mai mică decât z. Acest test se numeşte *testul marginii inferioare*. Dacă se cunoaşte o *relaţie de dominanţă D*, atunci în algoritm se poate introduce şi un *test de dominanţă D* de forma:

$$P_k \, D \, P_i \text{ implică } f(P_k) \leq f(P_i),$$

https://biblioteca-digitala.ro / https://unibuc.ro

unde D are semnificaţia unei relaţii de ordine parţială. Problema P_i poate fi în acest caz încheiată, deoarece P_k va furniza o soluţie cel puţin la fel de bună ca orice soluţie furnizată de P_i. Relaţia de dominanţă D este *consistentă cu g* dacă:

$$P_k \; D \; P_i \text{ implică } g(P_k) \leq g(P_i).$$

O problemă parţială generată într-un calcul Branch and Bound dar netestată încă se numeşte *activă*. Mulţimea curentă a nodurilor active se notează cu A şi mulţimea problemelor parţial generate se notează cu H. O *funcţie de căutare s* care selectează o problemă parţială $P_i = s(A)$ din A determină ordinea testării problemelor parţiale generate în timpul calculului.

O funcţie $s = s_h$ se numeşte *funcţie de căutare euristică* bazată pe *funcţia euristică h* şi care selectează o problemă parţială din A cu cele mai mici valori ale lui h. *Adâncimea* problemei P_i reprezintă numărul de descompuneri necesar generării problemei P_i din problema iniţială P_0. O funcţie $s = \bar{s}_k$ se numeşte *funcţie de căutare în adâncime* bazată pe h, care selectează problema cu cea mai mică valoare a lui h dintre problemele având adâncimea maximă. Funcţia $s = s_g$ se numeşte *funcţie de căutare cu cea mai bună margine*, adică marginea inferioară g este utilizată în locul lui h din s_h. Se poate folosi de asemenea funcţia de căutare în adâncime bazată pe g şi [illegible]tă $s=\bar{s}_g$.

În continuare este prezentat un algoritm BB pentru rezolvarea unei probleme de minimizare P_0.

Algoritmul 1.17. (*BB*)

1. [Iniţializări] Se face $H = \{P_0\}$, $A = \{P_0\}$ şi $z = \infty$.
2. [Căutare] Dacă $A = \emptyset$, atunci se trece la pasul 9; altfel se face $P_i = s(A)$ şi se trece la pasul 3.
3. [Testul G] Dacă $P_i \in G$, atunci se trece la pasul 7; altfel se trece la pasul 4.
4. [Testul marginii inferioare] Dacă $g(P_i) \geq z$, atunci se trece la pasul 8; altfel se trece la pasul 5.
5. [Testul de dominanţă] Dacă o anumită P_k $(\neq P_i) \in H$ satisface $P_k \; D \; P_i$, atunci se trece la pasul 8; altfel se trece la pasul 6.
6. [Ramificare] Se generează $S(P_i)$ descompunând pe P_i, se face $A = A \cup S(P_i) - \{P_i\}$ şi $H = H \cup S(P_i)$, apoi se trece la pasul 2.
7. [Îmbunătăţire] Se face $z = \min[z, f(P_i)]$.
8. [Încheie P_i] Se face $A = A - \{P_i\}$ şi se trece la pasul 2.
9. [Terminare] Se face $z = f(P_0)$ (dacă $z = \infty$, P_0 nu are soluţie) şi STOP. ■

Performanţa algoritmului BB este uneori măsurată cu următoarele valori:

- T: numărul de probleme parţiale descompuse în pasul 6 înainte ca algoritmul să se oprească la pasul 9.

https://biblioteca-digitala.ro / https://unibuc.ro

- *M*: mărimea maximă a mulţimii *A* atinsă în timpul parcurgerii algoritmului.

În continuare, sunt discutate trei tipuri de modificări care asigură accelerarea calculelor şi/sau reducerea spaţiului de memorare.

1. Metoda *ε-reducerii* (*ε-allowance method*) constă în înlocuirea testului din pasul 4 cu $g(P_i) + \varepsilon(z) \geq z$, unde $\varepsilon(z) \geq 0$. Funcţiile de reducere utilizate în practică sunt $\varepsilon(z) = \varepsilon$ (constantă) şi $\varepsilon(z) = \gamma z$ cu $\gamma \geq 0$.
2. Metoda *T-tăieturii* (*T-cut method*). Această metodă constă în întreruperea calculelor de îndată ce numărul de probleme parţiale descompuse în pasul 6 atinge o margine specificată T_0. Valoarea z_F a variabilei z în acest moment reprezintă valoarea soluţiei suboptimale. Pentru aplicarea acestei metode este suficient să se introducă în algoritmul *BB* un procedeu de contorizare a numărului problemelor parţial descompuse. În acest fel se asigură satisfacerea de către T a condiţiei $T \leq T_0$.
3. Metoda *M-tăieturii* (*M-cut method*). Această metodă constă în eliminarea necondiţionată în pasul 6 a unui număr de $|A = A \cup S(P_i) - \{P_i\}| - M_0$ probleme parţiale, unde M_0 este o *margine a capacităţii* (*capacity bound* M_0). În felul acesta se asigură satisfacerea de către M a condiţiei $M \leq M_0$.

În metodele de tip branch and bound, de o importanţă deosebită este modul de evaluare a unui nod.

Exemple de algoritmi euristici bazaţi pe metoda branch and bound: programarea activităţilor, probleme combinatoriale etc.

1.3.3.4. Euristici folosind metoda divide et impera

Principiul de construire al acestor algoritmi coincide cu cel descris pentru algoritmii exacţi de tip divide et impera. Deosebirea constă numai în eventualele relaxări luate în consideraţie la rezolvarea problemelor considerate simple.

Algoritmii de acest tip sunt foarte utili mai ales în calculul paralel.

Exemple de algoritmi euristici bazaţi pe metoda divide et impera: programarea activităţilor, determinarea extremelor unei funcţii, integrarea prin metoda Monte-Carlo, metode statistice pentru concordanţă şi omogenitate etc.

1.3.3.5. Euristici folosind metoda programării dinamice

Aceşti algoritmi euristici se obţin prin relaxarea condiţiilor de optimalitate. Această relaxare poate fi făcută fie în scopul unor calcule mai simple, fie în scopul satisfacerii principiului optimalităţii, făcând astfel posibilă aplicarea metodei programării dinamice.

https://biblioteca-digitala.ro / https://unibuc.ro

Exemple de algoritmi euristici bazaţi pe metoda programării dinamice: metode de alocare a memoriei (prima potrivire, cea mai bună potrivire, metoda camarazilor), programarea activităţilor etc.

1.4. Algoritmi genetici şi evolutivi

1.4.1. Introducere

Ideea de bază a algoritmilor genetici şi evolutivi constă în modificarea succesivă a unor soluţii iniţiale sau generate anterior folosind, de obicei, doi operatori: *mutaţie* şi *încrucişare*. Dacă noua soluţie este mai bună dacât cele precedente, procesul de modificare continuă. Altfel, dacă nu este posibilă ameliorarea soluţiei sau numărul de modificări a depăşit o anumită limită, algoritmul se încheie. Un algoritm se numeşte *genetic clasic* dacă soluţiile sale sunt reprezentate sub formă de *cromozomi*, adică şiruri de biţi 0 şi 1. Problemele în care soluţiile sunt reprezentate sub formă de cromozomi sunt însă rare. Pentru probleme obişnuite, implementarea conceptelor de *mutaţie* şi *încrucişare* (vezi 1.4.2.4.), aşa cum sunt gândite în cadrul algoritmilor genetici clasici, se poate realiza în două moduri, ilustrate în figurile 1.27. şi 1.28.. Algoritmii realizaţi în modurile descrise în aceste figuri se numesc *algoritmi genetici*, respectiv *evolutivi*. În cele ce urmează adjectivul *clasic* va fi omis.

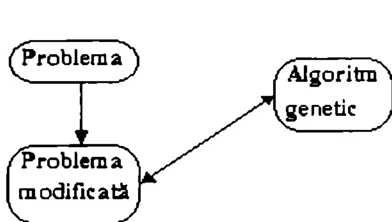


Fig. 1.27.

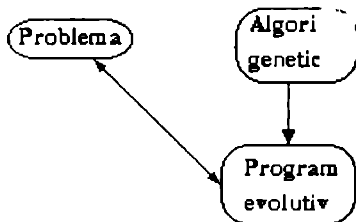


Fig. 1.28.

Un algoritm genetic, ca şi orice program evolutiv, pentru o problemă particulară trebuie să cuprindă următoarele cinci componente:

- o reprezentare genetică a soluţiilor potenţiale ale problemei studiate,
- un mod de creare a populaţiei iniţiale de soluţii potenţiale,
- o funcţie de evaluare, care joacă rol de mediu şi evaluează calitatea soluţiilor,
- operatori genetici ce alterează structura copiilor,
- valorile diferiţilor parametri pe care le utilizează algoritmii genetici cum sunt mărimea populaţiei, probabilităţile de aplicare a operatorilor genetici etc..

https://biblioteca-digitala.ro / https://unibuc.ro

1.4.2. Algoritm genetic pentru optimizarea unei funcţii

În cadrul exemplului ce urmează sunt ilustrate ideile de bază ale algoritmilor genetici, pe un exemplu de identificare a maximului unei funcţii. Funcţia dată este (vezi figura 1.29.):

$$f(x) = x \cdot \sin(10\pi \cdot n) + 1.0,$$

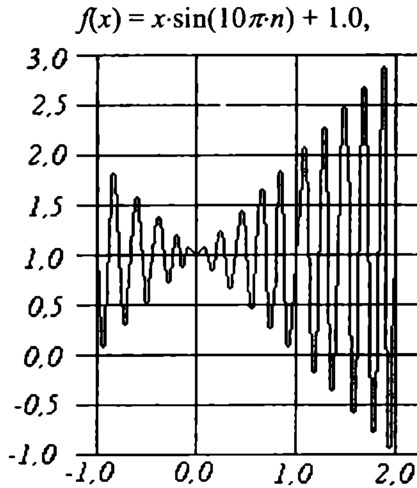


Fig. 1.29.

problema de rezolvat fiind identificarea lui x_0 aparţinând inervalului [-1, 2] pentru care funcţia dată are o valoare maximă, adică $f(x_0) \geq f(x)$ pentru toţi $x \in [-1, 2]$. Anulând derivata funcţiei f se obţine:

$$f'(x) = \sin(10\pi \cdot x) + 10\pi x \cdot \cos(10\pi x) = 0,$$

de unde rezultă:

$$\mathrm{tg}(10\pi \cdot x) = -10\pi x.$$

Ecuaţia precedentă are o infinitate de soluţii şi anume:

$$x_i = (2i - 1) / 20 + \varepsilon_i, \text{ pentru } i = 1, 2, \ldots,$$

$$x_0 = 0,$$

$$x_i = (2i + 1) / 20 - \varepsilon_i, \text{ pentru } i = -1, -2, \ldots,$$

unde ε_i reprezintă secvenţe descrescătoare de numere reale, pentru $i = 1, 2, \ldots$ şi $i = -1, -2, \ldots$, ce tind către zero. Funcţia f are maxime locale pentru x_i, dacă i este un întreg impar, şi minime locale pentru x_i, dacă i este un întreg par. Deoarece domeniul este [-1, 2], rezultă că maximul global se realizează pentru $x_{19}=37/20+\varepsilon_{19}=1.85+\varepsilon_{19}$, unde $f(x_{19})$ este puţin mai mare decât $f(1.85) = 1.85 \cdot \sin(18\pi + \pi/2) + 1.0 = 2.85$.

Componentele necesare pentru elaborarea algoritmului genetic de identificare a soluţiei problemei enunţate sunt prezentate în continuare.

https://biblioteca-digitala.ro / https://unibuc.ro

1.4.2.1. Reprezentarea

Pentru reprezentarea valorilor variabilei x se foloseşte un vector binar x, în presupunerea că precizia soluţiei este de şase cifre după punctul zecimal. Domeniul variabilei x are lungimea 3 şi precizia acceptată implică împărţirea intervalului [-1, 2] în cel puţin $3 \cdot 10^6$ subintervale egale. Rezultă că vectorul asociat variabilei x trebuie să conţină 22 biţi, deoarece:

$$2\,097\,152 = 2^{21} < 3\,000\,000 < 2^{22} = 4\,194\,304.$$

Transformarea şirului binar $<b_{21}\ b_{20} \ldots b_0>$ într-un număr real din intervalul [-1..2] se poate efectua în două etape:

- convertirea şirului binar $<b_{21}\ b_{20} \ldots b_0>$ din baza 2 în baza 10:

$$(<b_{21}\ b_{20} \ldots b_0>)_2 = (\sum_{i=0}^{21} b_i \cdot 2^i)_{10} = x',$$

- identifică numărul real corespunzător:

$$x = -1.0 + x' \cdot 3 / (2^{22} - 1),$$

unde -1.0 este marginea inferioară a domeniului şi 3 este lungimea domeniului. De exemplu, cromozomul:

$$(1000101110110101000111)$$

reprezintă numărul:

$$x' = (1000101110110101000111)_2 = 2\,288\,967,$$

şi

$$x = -1.0 + 2\,288\,967 \cdot 3 / 4\,194\,303 = 0.637197.$$

Evident, cromozomii:

$$(0000000000000000000000) \text{ şi } (1111111111111111111111)$$

reprezintă marginile domeniului adică pe -1.0, respectiv 2.0.

1.4.2.2. Populaţia iniţială

Populaţia iniţială constă din mai mulţi cromozomi, fiecare dintre ei fiind un vector de 22 de biţi. Cei 22 de biţi ai fiecărui cromozom sunt iniţializaţi aleator.

1.4.2.3. Funcţia de evaluare

Funcţia de evaluare *eval* a vectorilor binari v este echivalentă cu funcţia f:

$$eval(v) = f(x),$$

unde cromozomul v reprezintă valoarea reală x. De exemplu, următorii trei cromozomi:

$$v_1 = (1000101110110101000111),$$
$$v_2 = (0000001110000000010000),$$
$$v_3 = (1110000000111111000101),$$

https://biblioteca-digitala.ro / https://unibuc.ro

corespund valorilor x_1 = 0.637197, x_2 = -0.958973 şi, respectiv, x_3 = 1.627888. Funcţia de evaluare generează valorile:

$$eval(v_1) = f(x_1) = 1.586345,$$
$$eval(v_2) = f(x_2) = 0.078878,$$
$$eval(v_3) = f(x_3) = 2.250650.$$

Cromozomul v_3 este evident cel mai bun dintre cei trei cromozomi, deoarece furnizează cea mai mare valoare.

1.4.2.4. Operatori genetici

În faza de modificare a soluţiei deja generate se folosesc doi operatori clasici: *mutaţie* şi *încrucişare*. Operatorul *mutaţie* alterează una sau mai multe *gene* (biţi ai cromozomilor). Presupunând că a cincea genă a cromozomului v_3 a fost aleasă pentru mutaţie şi ţinând seama că ea este 0, rezultă că va fi transformată în 1. Noul cromozom obţinut este:

$$v_3' = (1110100000111111000101).$$

Acest cromozom reprezintă valoarea x_3' = 1.721638 şi $f(x_3')$ = -0.082257, ceea ce arată că această mutaţie particulară produce o descreştere a valorii cromozomului v_3. Dacă însă a 10-a genă a fost selectată pentru mutaţie în cromozomul v_3, atunci:

$$v_3'' = (1110000001111111000101).$$

Valoare corespunzătoare este x_3'' = 1.630818 şi $f(x_3'')$ = 2.343555, ceea ce reprezintă o îmbunătăţire a valorii $f(x_3)$ = 2.250650.

Pentu ilustrarea operatorului de încrucişare se consideră cromozomii v_2 şi v_3, cu presupunerea că punctul de încrucişare a fost ales (aleator) după a cincea genă:

$$v_2 = (00000|01110000000010000),$$
$$v_3 = (11100|00000111111000101).$$

Cei doi descendenţi generaţi sunt:

$$v_2' = (00000|00000111111000101),$$
$$v_3' = (11100|01110000000010000).$$

Evaluările acestor doi descendenţi sunt:

$$f(v_2') = f(-0.998113) = 0.940865,$$
$$f(v_3') = f(1.666028) = 2.459245.$$

Se observă că al doilea descendent are o evaluare mai bună decât cei doi părinţi. Aceste observaţii pot fi utilizate în continuare pentru a obţine soluţii mai bune prin operaţii succesive de mutaţie şi încrucişare din valori deja studiate.

1.4.2.5. Parametri

Pentru problema studiată au fost aleşi următorii parametri: mărimea populaţiei *pop_dim* = 50, probabilitatea încrucişerii p_c = 0.25, probabilitatea mutaţiei p_m = 0.01.

https://biblioteca-digitala.ro / https://unibuc.ro

Numărul generației	Funcția de evaluare
1	1.441942
6	2.250003
8	2.250283
9	2.250284
10	2.250363
12	2.328077
39	2.344251
40	2.345087
51	2.738930
99	2.849246
137	2.850217
145	2.850227

Tabelul 1.1.

1.4.2.6. Rezultate experimentale

În tabelul 1.1. sunt prezentate rezultatele aplicării procedurilor descrise pentru problema studiată, după 150 de generații. Cel mai bun cromozom obținut este:

$$v_{max} = (1111001101000100000101)$$

şi îi corespunde valoarea $x_{max} = 1.850773$. După cum era de aşteptat $x_{max} = 1.85 + \varepsilon$ şi $f(x_{max})$ este puțin mai mare decât 2.85.

1.4.2.7. Procesul de selectare

Pentru selectarea unei noi populații, folosind o distribuție de probabilități bazată pe funcția de evaluare, se foloseşte modelul ruletei. O asemenea ruletă se construieşte în modul următor:

- Se calculează evaluarea $eval(v_i)$ a fiecărui cromozom v_i ($i = 1, ..., pop_dim$).
- Se calculează evaluarea totală a populației:

$$F = \sum_{i=1}^{pop_dim} eval(v_i).$$

- Se calculează probabilitatea p_i a selectării pentru fiecare cromozom v_i, $i = 1, 2, ..., pop_dim$):

$$p_i = eval(v_i / F).$$

- Se calculează probabilitatea cumulativă q_i pentru fiecare cromozom v_i, $i = 1, 2, ..., pop_dim$):

$$q_i = \sum_{j=1}^{i} p_j.$$

https://biblioteca-digitala.ro / https://unibuc.ro

Selecţia se realizează pe calea rotirii ruletei de un număr de *pop_dim* ori. De fiecare dată, se alege un singur cromozom pentru noua populaţie în modul următor:

- Se generează un număr aleator real r din intervalul [0..1].
- Dacă $r < q_1$, atunci se selectează primul cromozom (v_1); altfel se selectează cromozomul v_i, $2 \le i \le pop_dim$, astfel încât $q_{i-1} < r \le q_i$.

Evident că unii cromozomi vor fi selectaţi mai mult decât o singură dată şi alţii nu vor fi selectaţi. Vom nota în continuare cu p_m probabilitatea de mutaţie şi cu p_c probabilitatea de încrucişare.

Pentru aplicarea operatorului *mutaţie* se consideră şirul de biţi generat de toţi cromozomii populaţiei curente. Numărul de biţi ai unei populaţii este egal cu $m \cdot pop_dim$, unde m este numărul (actual) de cromozomi. Pentru toţi biţii populaţiei se procedează în modul următor:

- Se generează un număr aleator r (real) din intervalul [0, 1];
- Dacă $r < p_m$, atunci se schimbă bitul examinat.

Numărul aşteptat al biţilor ce se schimbă este egal cu $p_m \cdot m \cdot pop_dim$.

Pentru operatorul *încrucişare* se procedează astfel:

- Se generează un număr aleator r (real) din intervalul [0, 1];
- Dacă $r < p_c$, atunci se selecteză cromozomul considerat pentru încrucişare.

În continuare, pentru fiecare pereche succesivă de cromozomi selectaţi pentru încrucişare, se generează un număr întreg *pos* din intervalul [1, m - 1]. Acest număr indică poziţia punctului ce defineşte încrucişarea. Doi cromozomi:

$$(b_1\ b_2\ \dots\ b_{pos}\ b_{pos+1}\ \dots\ b_m) \text{ şi}$$
$$(c_1\ c_2\ \dots\ c_{pos}\ c_{pos+1}\ \dots\ c_m)$$

sunt înlocuiţi cu perechea de descendenţi:

$$(b_1\ b_2\ \dots\ b_{pos}\ c_{pos+1}\ \dots\ c_m) \text{ şi}$$
$$(c_1\ c_2\ \dots\ c_{pos}\ b_{pos+1}\ \dots\ b_m).$$

Numărul aşteptat al cromozomilor ce participă la încrucişare este egal cu $p_c \cdot pop_dim$.

Observaţie. Dacă numărul de cromozomi selectaţi pentru încruc are este impar, atunci ori se adaugă aleator încă un cromozom, ori se elimină tot aleator unul dintre cromozomii existenţi.

1.4.3. Algoritm evolutiv pentru rezolvarea problemei liniar-pătratice

Exemplul precedent ilustrează concepţiile de bază ale algoritmilor genetici în cazul în care problema studiată este suficient de simplă pentru a permite codificarea binară a soluţiei. În cazul problemelor întâlnite de obicei, se poate adopta aceeaşi

https://biblioteca-digitala.ro / https://unibuc.ro

strategie, cu prețul creșterii foarte mari a lungimii cromozomilor și măririi timpului de calcul. În acest caz, este convenabilă utilizarea reprezentării cromozomilor în termeni de valori reale, eventual întregi, și adaptarea operatorilor *mutație* și *încrucișare* la acest mod de reprezentare. Algoritmul formulat în acest mod este de tip evolutiv și utilizează de obicei și alte modalități de efectuare a mutației și încrucișării după cum vom arăta în continuare.

Drept exemplu ilustrativ, fie problema determinării minimului:

$$\min q\cdot x_N^2 + \sum_{k=0}^{N-1}\left(s\cdot x_k^2 + r\cdot u_k^2\right), \tag{1.23}$$

cu condițiile:

$$x_{k+1} = a\cdot x_k + b\cdot u_k,\ k = 0, 1, ..., N-1, \tag{1.24}$$

unde x_0 este dat, a, b, q, s, r sunt constante date, $x_k \in$ R are semnificație de stare și $u_k \in$ R reprezintă controlul sistemului.

Valoarea performanței optime a procesului de minimizare a expresiei (1.23) cu condiția (1.24) este:

$$J^* = K_0\, x_0^2, \tag{1.25}$$

unde K_k este soluția ecuației Riccati:

$$K_k = s + r\, a^2\, K_{k+1} / (r + b^2\, K_{k+1}),\ K_N = q.$$

Problema (1.23) este rezolvată în continuare pentru mulțimile de parametrii din tabelul 1.2.

Cazul	N	x_0	s	r	q	a	b
I	45	100	1	1	1	1	1
II	45	100	10	1	1	1	1
III	45	100	1000	1	1	1	1
IV	45	100	1	10	1	1	1
V	45	100	1	1000	1	1	1
VI	45	100	1	1	10	1	1
VII	45	100	1	1	1000	1	1
VIII	45	100	1	1	1	0.01	1
IX	45	100	1	1	1	1	0.01
X	45	100	1	1	1	1	100

Tabelul 1.2.

1.4.3.1. Operatori specializați

Operatorii utilizați sunt diferiți de cei clasici, deoarece ei operează în spațiul valorilor reale. Totuși, datorită asemănărilor intuitive, ei sunt atribuiți claselor standard, adică *mutație* și *încrucișare*. În plus, unii operatori sunt neuniformi, adică acțiunea lor depinde de vârsta populației.

https://biblioteca-digitala.ro / https://unibuc.ro

Grupul mutaţiilor:

- *mutaţia uniformă* este definită în mod similar versiunii clasice: dacă x_i^t=<v_1,v_2, ...,v_n> este un cromozom, atunci fiecare element v_k are aceeaşi şansă de a fi modificat. Rezultatul unei mutaţii unice a operatorului este vectorul <v_1,v_2,..., v_k',...,v_n>, cu $1 \le k \le n$ şi v_k' este o valoare aleatoare din domeniul respectiv, $domeniu_k$.
- *mutaţia neuniformă* este un operator responsabil pentru capacitatea tratării mai fine a problemei. Operatorul de mutaţie neuniformă este definit astfel: dacă vectorul s_v^t = <v_1, v_2, ..., v_m> este un cromozom şi a fost ales elementul v_k pentru mutaţie, domeniul lui v_k fiind [l_k, u_k]), atunci, rezultatul este vectorul s_v^{t+1} = <v_1, v_2, ..., v_k', ..., v_m>, cu $k \in \{1, ..., n\}$ şi:

$$v_k' = \begin{cases} v_k + \Delta(t, u_k - v_k), & \text{dacă valoarea aleatoare este 0,} \\ v_k - \Delta(t, v_k - l_k), & \text{dacă valoarea aleatoare este 1,} \end{cases}$$

unde, $\Delta(t, y)$ furnizează o valoare din [0, y] astfel încât probabilitatea ca $\Delta(t, y)$ să fie aproape de 0 şi creşte o dată cu t. Această proprietate are drept consecinţă faptul că, iniţial, acest operator parcurge spaţiul în mod uniform (când t este mic) şi foarte local la etapele următoare. Un exemplu de funcţie $\Delta(t, y)$ este:

$$\Delta(t, y) = y \cdot (1 - r^{(1-t/T)^b}),$$

unde r este un număr aleator din [0, 1], T este numărul generaţiilor şi b este un parametru al sistemului ce determină gradul de neuniformitate. Valorile lui Δ pentru două momente distincte, sunt descrise de figurile 1.30. şi 1.31.

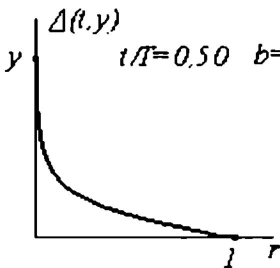


Fig. 1.30.

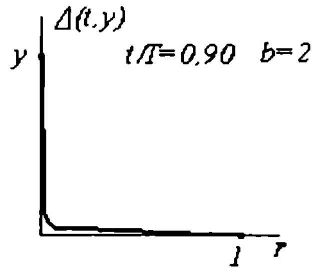


Fig. 1.31.

Grupul încrucişărilor:

- *încrucişarea simplă* este definită în mod obşnuit dar cu condiţia ca punctul de încrucişare să figureze între doi v_i şi v_j consecutivi ai unui cromozom dat x.
- *încrucişarea aritmetică* este definită drept conbinaţie liniară a doi vectori şi anume, dacă s_v^t şi s_w^t urmează să fie încrucişaţi, atunci descendenţii generaţi sunt $s_v^{t+1} = a \cdot s_w^t + (1-a)\, s_v^t$ şi $s_w^{t+1} = a \cdot s_v^t + (1-a)\, s_w^t$. În acest operator, a reprezintă fie o

https://biblioteca-digitala.ro / https://unibuc.ro

constantă (*încrucişare aritmetică uniformă*), fie o variabilă a cărei valoare depinde de vârsta populaţiei (*încrucişare aritmetică neuniformă*).

Rezultatele aplicării algoritmului evolutiv problemei enunţate mai sus sunt prezentate în tabelul 1.3. Pentru toate cele 10 cazuri, mărimea populaţiei a fost egală cu 70 şi au fost generate 40 000 de generaţii. Vectorul $<u_0, ..., u_{69}>$ a fost iniţializat cu valori aleatoare aparţinând domeniului de control. În tabelul 1.3. sunt menţionate atât valorile finale obţinute după 40 000 de generaţii cât şi valorile obţinute după diferite generaţii intermediare. În ultima coloană a tabelului este menţionat factorul cu care trebuie multiplicate valorile din coloanele 1-7 în vederea obţineri valorii soluţiei.

			Generaţii			Factor		
Cazul	1	100	1000	10000	20000	30000	40000	
I	17904.4	3.87385	1.73682	1.61859	1.61817	1.61804	1.61804	10^4
II	13572.3	5.56187	1.35678	1.11451	1.09201	1.09162	1.09161	10^5
III	17024.8	2.89355	1.06954	1.00952	1.00124	1.00102	1.00101	10^7
IV	15082.1	8.74213	4.05532	3.71745	3.70811	3.70162	3.70160	10^4
V	5968.42	12.2782	2.69862	2.85524	2.87645	2.87571	2.87569	10^5
VI	17897.7	5.37447	2.09334	1.61863	1.61837	1.61805	1.61804	10^4
VII	2690258	18.6685	7.23567	1.73564	1.65413	1.61842	1.61804	10^4
VIII	123.942	72.1958	1.95783	1.00009	1.00005	1.00005	1.00005	10^4
IX	7.28165	4.82740	4.39091	4.42524	4.31021	4.31004	4.31004	10^5
X	9971341	148233	16081.0	1.48445	1.00040	1.00010	1.00010	10^4

Tabelul 1.3.

Rezultatele furnizate de soluţia exactă a problemei studiate (vezi (1.25)) sunt date în tabelul 1.4.

Cazul	Soluţia exactă
I	16180.3399
II	109160.7978
III	10009990.0200
IV	37015.6212
V	287569.3725
VI	16180.3399
VII	16180.3399
VIII	10000.5000
IX	431004.0987
X	10000.9999

Tabelul 1.4.

https://biblioteca-digitala.ro / https://unibuc.ro

1.4.4. Forma generală a algoritmilor genetici şi evolutivi

Algoritmii evolutivi pot fi construiţi conform schemei următoare:

Algoritmul 1.18. (***Evolutiv***)

1. [Iniţializări] Se face $t = 0$ şi se iniţializează $P(0)$.
2. [Evaluare] Se evaluează $P(t)$.
3. [Terminare] Dacă sunt îndeplinite condiţiile de terminate, atunci STOP.
4. [Generaţie nouă] Se face $t = t + 1$ şi se selectează $P(t)$ din $P(t - 1)$.
5. [Adaptare] Se adaptează $P(t)$.
6. [Ciclare] Se trece la pasul 2. ■

În acest algoritm am notat prin $P(t)$ populaţia din generaţia t, prin evaluarea din pasul 2 se stabilesc măsuri elementelor populaţiei, selectarea din pasul 4 presupune alegerea elementelor utilizate pentru construirea unei noi generaţii care se determină în pasul 5 prin aplicarea regulilor de derivare. Condiţiile de terminare din pasul 3 stabilesc situaţiile în care calculele nu mai trebuiesc făcute în continuare fie din cauza imposibilităţii îmbunătăţirii soluţiei, fie prin atingerea numărului maxim de generaţii stabilit iniţial.

Un model de algoritm genetic în care populaţiile sunt puse în corespondenţă cu vectori cu m elemente 0 şi 1 este următorul.

Algoritmul 1.19. (***Iterated Hillclimber***)

1. [Iniţializări] Se face $t = 0$.
2. [Terminare] Dacă $t > MAX$, atunci STOP.
3. [Iniţializare iteraţie] Se selectează aleator vectorul curent v_c şi se evaluează v_c.
4. [Explorare] Se generează cei m vecini ai lui v_c obţinuţi prin schimbarea câte unui bit în vectorul v_c şi se alege unul dintre ei v_n care are cea mai mare valoare pentru funcţia obiectiv f.
5. [Progresare] Dacă $f(v_c) < f(v_n)$, atunci se face $v_c = v_n$ şi se trece la pasul 4.
6. [Ciclare] Se face $t = t + 1$ şi se trece la pasul 2. ■

Algoritmul anterior asigură determinarea unui maxim local dar soluţiile furnizate pot să fie foarte depărtate de maximul absolut. O variantă a acestui algoritm care de cele mai multe ori dă o aproximaţie a soluţiei problemei este următoarea.

Algoritmul 1.20. (***Simulated Annealing***)

1. [Iniţializări] Se face $t = 0$, se iniţializează T, se alege la întâmplare un vector curent v_c şi se evaluează v_c.
2. [Terminare] Dacă $T < \varepsilon$, atunci STOP.

https://biblioteca-digitala.ro / https://unibuc.ro

3. [Explorare] Se alege dintre cei m vecini ai lui v_c unul v_n care diferă de v_c printr-un singur bit.
4. [Progresare] Dacă $f(v_c) < f(v_n)$, atunci se face $v_c = v_n$; altfel se generează un număr aleator r din intervalul [0, 1) şi dacă $r < \exp((f(v_n) - f(v_c))/T)$, atunci se face $v_c = v_n$.
5. [Reluare] Dacă v_c poate să fie îmbunătăţită, atunci se trece la pasul 3.
6. [Ciclare] Se face $T = g(T, t)$ şi $t = t + 1$ şi se trece la pasul 2. ■

Funcţia g din pasul 6 are propietatea $g(T, t) < T$, pentru orice T şi t. Variabila T limitează posibilităţile de *întoarcere*, adică posibilitatea de a considera combinaţii mai puţin performante decât cele cunoscute la momentul curent. Întoarcerea devine din ce în ce mai limitată pe măsură ce se consideră o generaţie de ordin mai mare. Când nu mai sunt posibilităţi de întoarcere ($T < \varepsilon$) explorarea se termină.

1.4.5. Observaţii şi comentarii.

1. În unele probleme, datele au un caracter estimativ. În asemenea cazuri este indicat să se găsească o soluţie euristică apropiată de cea optimală, obţinută mai uşor decât una optimală (vezi de exemplu [SAH77]).

2. În [FIS80] sunt citate trei metode de analiză a algoritmilor: testul empiric, comportarea în cel mai rău caz şi analiza probabilistă. Principalul avantaj al testării empirice constă în acurateţea ei. Măcar pentru problemele testate, performanţa algoritmului este cert cunoscută. Pe de altă parte, testarea empirică este costisitoare şi furnizează numai o informaţie de natură statistică a performanţei euristicii.

Analiza comportării în cel mai rău caz are avantajul furnizării unei garanţii privind abaterea maximală a euristicii faţă de soluţia optimală, pentru *orice* problemă din clasa studiată. Această analiză furnizează de asemenea exemple pentru care euristica studiată se comportă în modul cel mai rău. Desavantajul major al analizei în cazul cel mai rău constă în faptul că nu este de obicei predictivă privind performanţa în medie.

Analiza probabilistă corectează deficienţa de mai sus, arătând cum se comportă euristica pentru probleme *tipice*. Limitarea majoră a analizei probabiliste constă în faptul că nu furnizează o funcţie de densitate pentru datele problemei, alese de obicei sub o formă simplă. În afară de aceasta, cele mai multe rezultate probabiliste sunt asimptotice în mărimea problemei şi nu sunt strict aplicabile problemelor de mărime finită.

O observaţie cu caracter general privind marginile în cazul cel mai rău, este că valorile lor sunt prea mari în comparaţie cu performanţa medie sau cu o performanţă considerată ca fiind acceptabilă.

https://biblioteca-digitala.ro / https://unibuc.ro

2 PROBLEME COMBINATORIALE

2.1. Împerechere minimală

Punerea problemei. Fiind dat un număr par N de puncte situate într-un plan, se cere identificarea unei împărţiri în N / 2 perechi a acestor puncte astfel încât suma distanţelor dintre punctele perechilor să fie minimă. ■

Aplicaţii posibile: construirea circuitelor electrice şi electronice, gruparea informaţiilor în baze de date.

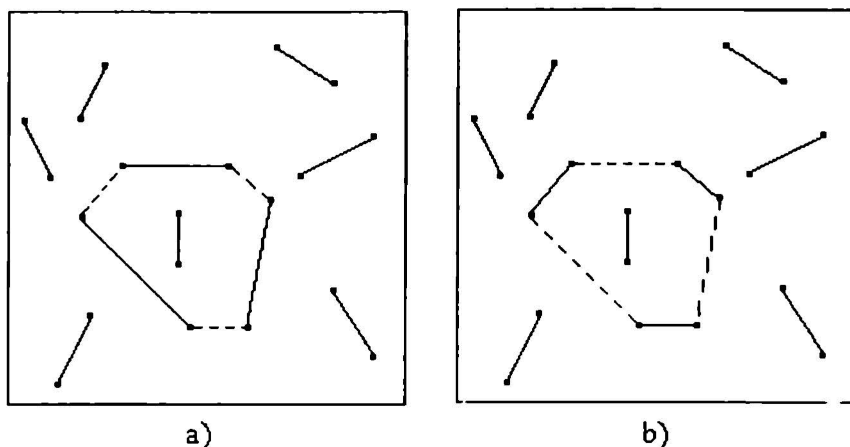


Fig. 2.1.

Euristica propusă se bazează pe următoarea procedură. Se aleg la întâmplare $2n$ puncte ce formează un ciclu de perechi alternativ conectate respectiv neconectate (vezi figura 2.1.a). Apoi se calculează suma lungimilor muchiilor ce conectează, respectiv, nu conectează puncte şi care sunt desenate cu linii pline, respectiv, punctate. Dacă suma lungimilor muchiilor pline este mai mică decât suma lungimilor celor punctate, atunci se acceptă împerecherea construită. Altfel muchiile pline se

https://biblioteca-digitala.ro / https://unibuc.ro

consideră întrerupte şi reciproc (vezi figura 2.1.b), obţinând astfel o valoare iniţială mai bună pentru perechile considerate. Această operaţie se numeşte, în continuare, *rotirea* (engl. *rotation*) unui ciclu de perechi alternative.

Mai precis, alegerea ciclurilor se efectuează astfel. După ce a fost obţinută o configuraţie de conexiuni, se elimină în mod arbitrar o conexiune după care unul dintre cele două puncte se notează cu "-" şi se numeşte *coadă* şi celălalt se notează cu "+" şi se numeşte *cap* (vezi figura 2.2.).

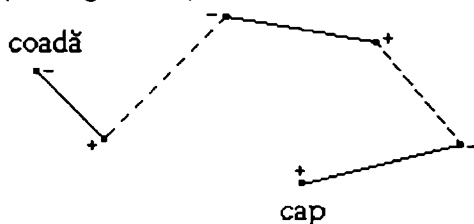


Fig. 2.2.

În continuare se încearcă identificarea unui ciclu corect construit, adică format din muchii alternativ pline şi punctate, pe calea conectării capului cu unul dintre nodurile anterioare, situate pe drumul coadă-cap. Un asemenea ciclu este arătat pe figura 2.3.a). Două exemple de cicluri necorect construite sunt arătate pe figurile 2.3.b) şi 2.3.c). Într-o versiune mai eficientă, construirea ciclului se efectuează pe calea extinderii drumului iniţial identificat în două direcţii, atât dinspre capul cât şi dinspre coada drumului.

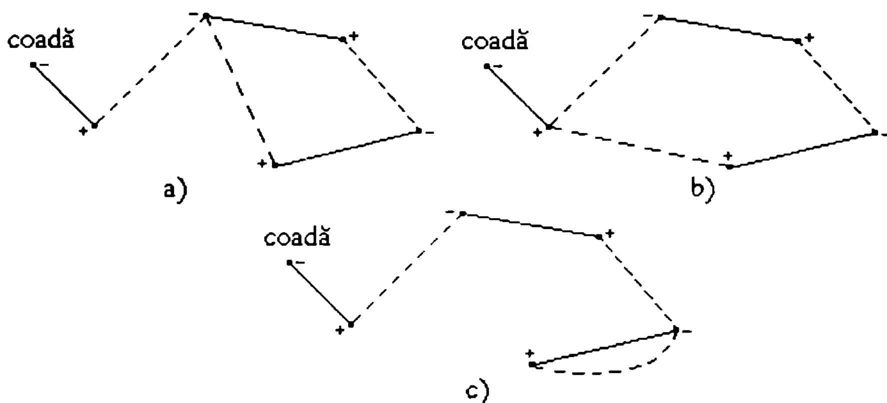


Fig. 2.3.

În cadrul algoritmului există mai multe detalii expuse în continuare.

1. La elaborarea configuraţiei iniţiale se poate adopta o strategie mai mult sau mai puţin *Greedy*, dar s-a constatat că viteza convergenţei depinde foarte puţin de

https://biblioteca-digitala.ro / https://unibuc.ro

acest factor. Din acest motiv, este acceptabil ca mulţimea iniţială a perechilor să fie elaborată pe calea parcurgerii la întâmplare a punctelor (nodurilor) şi identificarea succesivă a celui mai apropiat vecin. Dacă nu există vre-un vecin liber, atunci se alege la întâmplare un nod liber.

2. În testările efectuate, numărul de vecini a fost $K = 17$. Această alegere e motivată de faptul că în acest fel au putut fi analizate latice cu câte 20 000 de noduri.
3. Fiecărui nod din lista vecinilor i se atribuie o pondere care descreşte cu distanţa:

$$p_k = \text{const} \bullet e^{-\mu l_k},$$

unde, l_k este distanţa la vecinul k, μ o constantă pozitivă şi:

$$\sum_{k=1}^{K} p_k = 1.$$

Pentru selectarea nodului pereche au fost utilizate următoarele două metode:

a) Folosind un generator (pseudo-)aleator, se identifică un k cu probabilitatea p_k. Dacă eticheta acestui nod este nepozitivă şi dacă acest nod nu a fost mai înainte legat cu capul actual, atunci el este acceptat. Altfel căutarea este reluată până se identifică un nod acceptabil.

b) Se începe la fel ca mai sus. Dacă însă nodul are eticheta "+" sau este deja conectat cu capul, atunci se caută *Greedy* cel mai apropiat nod acceptabil.

Testările efectuate au demonstrat că metoda b) este mai eficientă (mai rapidă) decât metoda a), dar nu într-un mod esenţial.

4. După cum s-a mai amintit, continuarea căutării poate avea loc la ambele capete sau numai la capul drumului anterior identificat. Rezultatele obţinute arată că prima strategie este ceva mai eficientă.
5. Prima pereche ce este desfăcută (*brocken*) poate fi aleasă aleator, sau astfel încât lungimea muchiei corespunzătoare să fie mai scurtă sau mai lungă decât lungimea medie. Cu toate că s-ar părea că este mai eficientă alegerea unei muchii mai lungi, experienţele au demonstrat că prima pereche poate fi aleasă aleator.
6. După construirea unui ciclu, se poate continua de la aceeaşi pereche iniţială de noduri sau se caută altă pereche. A doua variantă s-a dovedit ceva mai eficientă decât prima.
7. După ce a fost construit un ciclu, se poate trece la rotirea lui dacă această operaţie micşorează lungimea sau la identificarea altui ciclu. A doua variantă poate fi mai convenabilă, deoarece evită consumul de timp necesar identificării optimalităţii.

Algoritmul prezentat s-a dovedit foarte robust deoarece s-a comportat foarte bine în majoritatea testărilor. Folosind o structură de date adecvată programele ce corespund diferitelor variante ale algoritmului pot fi uşor construite şi se dovedesc foarte eficiente.

https://biblioteca-digitala.ro / https://unibuc.ro

2.2. Problema rucsacului

Problema rucsacului a mai fost abordată în în paragrafele 1.1.2.3.3. şi 1.1.2.5.1. În continuare vom da alte metode de rezolvare şi caracteristici ale soluţiilor ei.

2.2.1. Algoritm euristic eficient pentru problema rucsacului

Algoritmul 1.3. prezentat anterior poate avea o comportare arbitrar de neeficientă după cum se vede din exemplul următor.

Exemplul 2.1. Fie un număr întreg $x > 2$ oricât de mare şi o problemă a rucsacului cu capacitatea $M = x$ şi două obiecte ale căror ponderi sunt $w_1 = 1$ şi $w_2 = x$ şi cu profiturile $p_1 = 2$ şi $p_2 = x$. Obiectele sunt deja ordonate necrescător în raport cu densitatea profitului deoarece $p_1 / w_1 = 2$ este mai mare decât $p_2 / w_2 = 1$. Prin aplicarea algoritmului 1.3. se plasează mai întâi primul obiect în rucsac deoarece $w_1 < M$ şi algoritmul se termină cu profitul 2. Al doilea obiect nu se mai poate pune deoarece s-ar depăşi capacitatea rucsacului. Evident că valoarea optimă se obţine prin plasarea celui de al doilea obiect în rucsac, în acest caz obţinând profitul x. Deci algoritmul Greedy furnizează o valoare de $x / 2$ ori mai mică decât cea optimă. ■

Algoritmul 1.3. poate fi modificat uşor pentru a obţine o soluţie mai bună în cazul în care nici un obiect nu are o pondere mai mare decât capacitatea rucsacului. De altfel astfel de obiecte ar putea să fie ignorate pentru că ele nu pot fi puse în rucsac în nici o soluţie. Algoritmul transformat este următorul:

Algoritmul 2.1. *AL(I, p, w, M)*

1. [Profit maxim] Se determină valoarea maximă p_i a profitului dat de un obiect.
2. [Profit dat de L] Se determină valoarea v furnizată de algoritmul 1.3.
3. [Terminare] Se furnizează cea mai mare dintre valorile determinate în paşii 1 şi 2, apoi STOP. ■

Teorema 2.1. Soluţia furnizată de algoritmul 2.1. este cel puţin jumătate din soluţia optimală.

Demonstraţie. Fie *opt* soluţia optimală şi *sol* cea furnizată de algoritm. Fie l cel mai mic întreg astfel încât $\sum_{i=1}^{l} w_i > M$ în ipoteza că rucsacul nu poate conţine toate obiectele. Fie apoi o problemă modificată care foloseşte aceleaşi obiecte dar capacitatea rucsacului are valoarea $M' = \sum_{i=1}^{l} w_i$. În acest caz, algoritmul Greedy furnizează soluţia optimală (vezi [HOR78]) având valoarea $opt' = \sum_{i=1}^{l} p_i$. Este evident că soluţia problemei originale nu poate fi mai mare decât cea a problemei modificate,

https://biblioteca-digitala.ro / https://unibuc.ro

deci $opt \le opt'$. Pe de altă parte $v \ge \sum_{i=1}^{l-1} p_i$, unde v este valoarea din pasul 2, deoarece algoritmul 1.3. va introduce sigur primele l - 1 obiecte în rucsac înainte de a eşua plasarea obiectului l. În plus $p_i \ge p_l$, unde p_i este cea mai mare valoare a profitului determinată în pasul 1 al algoritmului. Ţinând seama de cele menţionate mai sus şi de inegalitatea:

$$\max\{x, y\} \ge (x + y) / 2,$$

rezultă că:

$$sol = \max\{p_i, v\} \ge (p_i + v) / 2 \ge (p_l + \sum_{i=1}^{l-1} p_i) / 2 = \sum_{i=1}^{l} p_i / 2 = opt' / 2 \ge opt / 2,$$

adică algoritmul 2.1. dă o soluţie egală cu cel puţin jumătate din cea optimală. ■

2.2.2. Algoritm aproximativ polinomial pentru problema rucsacului

Algoritmul ce urmează furnizează o soluţie ε-aproximativă pentru problema rucsacului cu n obiecte, unde: $E_n = \{1, 2, ..., n\}$ este mulţimea obiectelor, $w_i \in Z^+$ sunt greutăţile, $p_i \in Z^+$ sunt profiturile obiectelor şi $M \in Z^+$ este capacitatea rucsacului. Se presupune că $w_i \le M$, pentru $i = 1, 2, ..., n$. Acest algoritm este *polinomial aproximativ* în sensul că necesită un timp polinomial în mărimea l a datelor problemei rezolvate.

Algoritmul 2.2. (*Rucsac*)

1. [Iniţializări] $P_{max} = 0$, $k = \lceil 1 / \varepsilon \rceil - 1$.
2. [Ordonare] Se sortează obiectele după densitatea profiturilor astfel încât $p_1 / w_1 \ge p_2 / w_2 \ge ... \ge p_n / w_n$.
3. [Ponderi] Pentru fiecare submulţime $S \subseteq E_n$ cu $|S| \le k$ şi $\sum_{i \in S} w_i \le M$ se face:
$$P = \sum_{i \in S} p_i;\ W = \sum_{i \in S} w_i.$$
4. [Ciclare] Pentru $j = 1, 2, ..., n$: Dacă $j \notin S$ şi $W + w_j \le M$, atunci, se face:
$$P = P + p_j;\ W = W + w_j;\ P_{max} = \max(P_{max}, P).$$
5. [Terminare] Se furnizează P_{max} şi STOP. ■

Algoritmul enumeră toate submulţimile cu k ori mai puţine elemente, după care o procedură greedy este aplicată fiecărei asemenea submulţimi. Se poate arăta că, pentru $k = \lceil 1 / \varepsilon \rceil - 1 \le n$, acest algoritm este polinomial.

2.2.3. Teste de realizabilitate

Punerea problemei. Să se deducă margini inferioare pentru timpul necesar găsirii de soluţii aproximative ale problemei rucsacului pentru algoritmi ce folosesc oracole în vederea efectuării testelor de realizabilitate (*feasability*) şi dominanţă. ■

https://biblioteca-digitala.ro / https://unibuc.ro

În teorema ce urmează se deduce o margine inferioară a programelor ce folosesc teste de realizabilitate dar nu şi teste de dominanţă, şi nu se utilizează ordonarea obiectelor după densitatea profitului. O submulţime $S \subseteq E_n$ este *realizabilă* dacă satisface condiţia, folosită în programul de mai înainte, $\sum_{i \in S} w_i \le M$. În caz contrar, mulţimea S se numeşte *nerealizabilă*. Pentru orice $S \subseteq E_n$, se definesc mărimile $w(S) = \sum_{i \in S} w_i$ şi $p(S) = \sum_{i \in S} p_i$. Cu $R_{w,M}$ se notează un oracol în care sunt memorate $w_1, w_2, ..., w_n$ şi M. *Oracolul de realizabilitate* folosit în continuare este definit astfel:

$$R_{w,M}(S) = \begin{cases} <, \text{ dacă } w(S) < M, \\ =, \text{ dacă } w(S) = M, \\ >, \text{ dacă } w(S) > M. \end{cases}$$

Un asemenea oracol este mai riguros decât oracolele "da-nu" folosite în schemele aproximative. Mai jos se foloseşte notaţia $\gamma = \lceil 1 / \varepsilon \rceil - 1$.

Teorema 2.2. Orice algoritm ε-aproximativ, adică un algoritm care furnizează o soluţie realizabilă cu eroarea ε, $0 < \varepsilon < 1$, pentru problema rucsacului, care are acces la w_i şi M numai prin intermediul unui oracol de realizabilitate, foloseşte cel puţin $\binom{n}{\gamma}$ apelări ale oracolului.

Demonstraţie. Pentru $n \ge 2$ şi $0 < \varepsilon < 1$, fie $m \in Z^+$, astfel încât $1 \le m < 1 / \varepsilon$ şi $|n / 2 - m|$ să fie minimală. Dacă $2 / (n + 2) \le \varepsilon < 1$, adică $n / 2 + 1 \ge 1 / \varepsilon > 1$, atunci $m = \gamma$. Dacă $0 < \varepsilon < 2 / (n + 2)$, atunci $m = \lceil n / 2 \rceil$. În ambele cazuri, $\binom{n}{m} \ge \binom{n}{\gamma}$. Demonstraţia teoremei constă în a arăta că nici un algoritm care are acces numai la w_i şi M, prin intermediul oracolului descris, nu poate garanta faptul că eroarea relativă a soluţiei furnizate este mărginită de ε, dacă acest algoritm foloseşte mai puţin de $\binom{n}{m}$ apeluri la oracol.

Fie A un algoritm ε-aproximativ care apelează oracolul său de $r < \binom{n}{m}$ ori. Fie o problemă a rucsacului cu $p_i = 1$; $w_i = 2$; pentru $i = 1, 2, ..., n$ şi $M = 2m - 1$. Evident că $R_{w,M}$ furnizează răspunsul "<" pentru orice $S \subseteq E_n$ cu $|S| < m$ şi răspunsul ">" pentru orice $S \subseteq E_n$ cu $|S| \ge m$. Pentru intrarea I, algoritmul A apelează pe $R_{w,M}$ pentru mulţimile $S_1, S_2, ..., S_r$, unde $S_i \subseteq E_n$, şi furnizează răspunsul realizabil S_{r+1}. Deoarece $\binom{n}{m} > r$ şi $|S_{r+1}| < m$, pentru că S_{r+1} este realizabil, rezultă că există o submulţime $S_0 \subseteq E_n$ astfel încât $|S_0| = m$ şi $S_0 \ne S_i$, $1 \le i \le r + 1$. Fie acum intrarea I' definită prin:

$$w'_i = \begin{cases} m, & \text{dacă } i \in S_0, \\ m + 1, & \text{dacă } i \notin S_0, \end{cases} \quad M' = w'(S_0) = m^2 \text{ şi } p'_i = p_i, \text{ pentru } i = 1, 2, ..., n.$$

https://biblioteca-digitala.ro / https://unibuc.ro

Pentru fiecare S_i, $1 \leq i \leq r$,

$|S_i| < m$ implică $w'(S_i) \leq (m + 1)\cdot|S_i| \leq (m + 1)(m - 1) = m^2 - 1 \leq M'$

şi, deoarece fiecare S_i, $1 \leq i \leq r$, cu $|S_i| \geq m$, diferă de S_0 măcar cu un obiect,

$|S_i| \geq m$ implică $w'(S_i) \geq (m + 1) + m\cdot(|S_i| - 1) \geq (m + 1) + m(m - 1) = m^2 + 1 > M'$.

Prin urmare, pentru fiecare S_i, $1 \leq i \leq r$, $R_{w',M'}(S_i) = R_{w,M}(S_i)$. Deoarece oracolele $R_{w,M}$ şi $R_{w',M'}$ se comportă la fel pentru fiecare S_i, $1 \leq i \leq r$ şi $p_i = p_i'$, $1 \leq i \leq n$, rezultă că algoritmul A nu poate distinge pe I de I' în r apeluri ale oracolului şi furnizează acelaşi răspuns S_{r+1} pentru ambele probleme. Deoarece S_{r+1} este realizabil pentru I, rezultă că $|S_{r+1}| < m$, astfel încât $p'(S_{r+1}) \leq m - 1$. Atunci eroarea relativă a lui S_{r+1} pentru problema I' este:

$$\frac{p'(S_0) - p'(S_{r+1})}{p'(S_0)} \geq \frac{m-(m-1)}{m} = \frac{1}{m} > \varepsilon,$$

ceea ce contrazice ipoteza că A este un algoritm cu oracole ε-aproximativ. Deoarece m a fost ales astfel încât $\binom{n}{m} \geq \binom{n}{\gamma}$, rezultă că $\binom{n}{\gamma}$ este o margine inferioară pentru numărul de apeluri ale oracolului în vederea garantării unei erori nu mai mari decât ε. ■

2.2.4. Cazul densităţilor ordonate ale profitului

În ipoteza că densităţile profiturilor sunt preordonate, adică ordonate înainte de aplicarea algoritmului, ca în paragraful 2.2., are loc teorema ce urmează.

Teorema 2.3. Orice algoritm ε-aproximativ pentru problema rucsacului, care are acces la w_i şi M numai prin intermediul unui oracol de realizabilitate, foloseşte cel puţin $\binom{n}{\gamma}$ apelări ale oracolului, chiar dacă obiectele la intrare sunt sortate astfel încât $p_i / w_i \geq p_{i+1} / w_{i+1}$, pentru $i = 1, 2, ..., n - 1$.

Demonstraţie. Pentru $n \geq 2$ şi $0 < \varepsilon < 1$ se identifică un $m \in Z^+$ astfel încât $1 \leq m < 1/\varepsilon$ şi $|n / 2 - m|$ este minimală (ca în teorema 2.2.). Să presupunem că A este un algoritm cu oracole ε-aproximativ care apelează oracolul său de realizabilitate de $r < \binom{n}{m}$ ori. Fie x cel mai mic întreg astfel încât:

$$x > \frac{(n-1)(m-1)}{1-m\varepsilon} \geq 0.$$

Fie apoi I problema rucsacului cu $p_i = x + n + i$, $w_i = 2$, pentru $i = 1, 2, ..., n$ şi $M = 2m-1$. Deoarece $p_i > p_{i+1}$ şi $w_i = w_{i+1}$, pentru $1 \leq i < n$, rezultă $p_i / w_i \geq p_{i+1} / w_{i+1}$, pentru $1 \leq i < n$, aşa cum se presupune în enunţul teoremei. Ca şi în teorema 2.2., $R_{w,M}$ furnizează răspunsul "<" pentru orice $S \subseteq E_n$ cu $|S| < m$ şi răspunsul ">" pentru orice

https://biblioteca-digitala.ro / https://unibuc.ro

$S \subseteq E_n$ cu $|S| \geq m$. Algoritmul A apelează pe $R_{w,M}$ pentru mulţimile $S_1, S_2, .. S_r$, unde $S_i \subseteq E_n$, şi furnizează soluţia realizabilă S_{r+1}. Deoarece $\binom{n}{m} > r$ şi $|S_{r+1}| < m$, există o submulţime $S_0 \subseteq E_n$ astfel încât $|S_0| = m$ şi $S_0 \neq S_i$, $1 \leq i \leq r + 1$. Fie acum altă problemă I' cu:

$$w'_i = \begin{cases} x+n-2, & \text{dacă } i \in S_0, \\ x+n-1, & \text{dacă } i \notin S_0, \end{cases} \quad M' = w'(S_0) = m(x+n-2) \text{ şi } p'_i = p_i = x+n-1,\ 1 \leq i \leq n.$$

Pentru orice $1 \leq i \leq r$ au loc relaţiile:

$$\frac{p'_i}{w'_i} \geq \frac{x+n-i}{x+n-1} \text{ şi } \frac{p'_{i+1}}{w'_{i+1}} \geq \frac{x+n-i-1}{x+n-2},$$

de unde, aşa cum s-a cerut,

$$\frac{p'_i w'_{i+1}}{w'_i p'_{i+1}} \geq 1 \text{ şi } \frac{p'_i}{w'_i} \geq \frac{p'_{i+1}}{w'_{i+1}},\ 1 \leq i \leq n.$$

Pentru fiecare S_i, $1 \leq i \leq r$:

$|S_i| < m$ implică $w'(S_i) \leq (x + n - 1) \cdot |S_i| \leq (x + n - 1)(m - 1) = M' - ((x + n) - (m + 1))$.

În continuare, $m > 0$, $\varepsilon > 0$ şi $m < 1 / \varepsilon$ implică $0 < m\varepsilon < 1$, de unde:

$x > \frac{(n-1)(m-1)}{1-m\varepsilon} > (n - 1)(m - 1)$ şi $(x + n) - (m + 1) > n m - 2m \geq 0$, pentru orice $n \geq 2$.

Rezultă că $|S_i| < m$ implică $w'(S_i) < M'$. Deoarece fiecare S_i, $1 \leq i \leq r$, cu $|S_i| \geq m$ diferă de S_0 măcar cu un obiect, $|S_i| \geq m$ şi $S_0 \neq S_i$ implică:

$$w'(S_i) \geq (x + n - 1) + (x + n - 2)(|S_i| - 1) \geq$$
$$\geq (x + n - 1) + (x + n - 2)(m - 1) = m(x + n - 2) + 1 > M'.$$

Prin urmare, pentru fiecare S_i, $1 \leq i \leq r$, $R_{w',M'}(S_i) = R_{w,M}(S_i)$. Deoarece oracolele $R_{w,M}$ şi $R_{w',M'}$ se comportă la fel pentru orice S_i, $1 \leq i \leq r$, şi $p_i = p'_i$, pentru $1 \leq i \leq n$, şi ambele mulţimi de obiecte sunt sortate conform densităţii profiturilor, algoritmul A nu poate distinge problemele I şi I' în r apeluri ale oracolului şi furnizează aceeaşi soluţie S_{r+1}.

Deoarece S_{r+1} este realizabilă pentru I, rezultă că $|S_{r+1}| < m$. Rezultă că o margine slabă (engl. *loose bound*) pentru $p'(S_{r+1})$ este:

$$p'(S_{r+1}) \leq (m - 1) \max_{1 \leq i \leq n} p'_i = (m - 1)(x + n - 1).$$

Faptul că S_0 este o soluţie realizabilă pentru I' şi $|S_0| = m$, implică:

$$p'(S_0) \geq m \min_{1 \leq i \leq n} p'_i = m x.$$

Eroarea relativă a lui S_{r+1} ca soluţie pentru problema I' este, prin urmare, cel puţin egală cu:

$$\frac{p'(S_0) - p'(S_{r+1})}{p'(S_0)} \geq \frac{mx - (m-1)(x+n-1)}{mx} = \frac{1}{m} - \frac{(n-1)(m-1)}{mx} > \frac{1}{m} - \frac{1-m\varepsilon}{m} = \varepsilon.$$

https://biblioteca-digitala.ro / https://unibuc.ro

Acest rezultat contrazice ipoteza că A este un algoritm ε-aproximativ cu oracole şi, prin urmare, marginea inferioară este egală cu $\binom{n}{\gamma}$. ■

2.2.5. Algoritmi genetici pentru problema rucsacului

Vom considera trei tipuri de probleme ale rucsacului şi anume:

1. *Problema necorelată*, în care atât w_i cât şi p_i, pentru $i = 1, 2, ..., n$, iau valori aleatoare din intervalul $[1, v]$, unde v este un număr natural dat.
2. *Problema cu corelaţie slabă*, în care w_i, pentru $i = 1, 2, ..., n$, iau valori aleatoare din intervalul $[1, v]$, unde v este un număr natural dat, şi $p_i = w_i + u_i$, unde u_i este un număr aleator din intervalul $[-r, r]$, cu r număr natural dat, pentru $i = 1, 2,..., n$.
3. *Problema cu corelaţie tare*, în care w_i, pentru $i = 1, 2, ..., n$, iau valori aleatoare din intervalul $[1, v]$, unde v este un număr natural dat, şi $p_i = w_i + r$, unde r este un număr natural dat, pentru $i = 1, 2, ..., n$.

Cu cât corelaţia este mai mare, cu atât este mai mică diferenţa:

$$\max_i p_i / w_i - \min_i p_i / w_i,$$

de unde rezultă că o problemă cu o corelaţie mai mare este, în general, mai dificilă.

În experimentările făcute s-a considerat $v = 10$ şi $r = 5$. Pentru teste au fost folosite câte trei rânduri de date pentru fiecare tip de problemă având un număr de elemente $n = 100$, 250 şi, respectiv, 500. Se pot considera două tipuri de probleme:

- *rucsac de capacitate restrictivă* în care capacitatea rucsacului este $M_1 = 2v$; în acest caz soluţia optimală conţine numai câteva elemente;
- *rucsac de capacitate aleatoare* în care capacitatea rucsacului este $M_2 = \frac{1}{2}\sum_{i=1}^{n} w_i$; în acest caz soluţia optimală conţine circa jumătate dintre elemente.

Au fost concepute trei tipuri de algoritmi: algoritmi $A_p(i)$ bazaţi pe funcţii de penalitate, algoritmi $A_r(i)$ bazaţi pe metode de rectificare şi algoritmi $A_d(i)$ bazaţi pe decodificatori. Vom prezenta în continuare aceste tipuri de algoritmi.

Algoritmii $A_p(i)$. În aceşti algoritmi, soluţiile sunt reprezentate sub forma unor vectori x cu n elemente cu valori 0 şi 1; se selectează al i-lea element pentru a fi pus în rucsac dacă şi numai dacă $x_i = 1$. Evaluarea unui vector x se face după formula:

$$eval(x) = \sum_{i=1}^{n} x_i p_i - pen(x),$$

unde funcţia de penalitate $pen(x)$ este 0 pentru orice soluţie corectă x, adică acelea pentru care $\sum_{i=1}^{n} x_i w_i \le M$, şi fiind strict pozitivă în caz contrar. Se pot folosi mai multe

https://biblioteca-digitala.ro / https://unibuc.ro

strategii pentru calculul penalităţilor. Următoarele trei exemple consideră creşterea penalităţii logaritmic, liniar şi, respectiv pătratic, în raport de măsura depăşirii:

$$A_p(1): pen(x) = \log_2(1 + \rho(\sum_{i=1}^{n} x_i w_i - M)),$$

$$A_p(2): pen(x) = \rho(\sum_{i=1}^{n} x_i w_i - M),$$

$$A_p(3): pen(x) = (\rho(\sum_{i=1}^{n} x_i w_i - M))^2.$$

În formulele anterioare s-a motat $\rho = \max_i p_i / w_i$.

Algoritmii $A_r(i)$. În aceşti algoritmi, soluţiile sunt reprezentate tot sub forma unor vectori x cu n elemente cu valori 0 şi 1; se selectează al i-lea element pentru a fi pus în rucsac dacă şi numai dacă $x_i = 1$. Evaluarea unui vector x se face după formula:

$$eval(x) = \sum_{i=1}^{n} x'_i p_i ,$$

unde x' este versiunea rectificată a vectorului original x. Sunt două probleme de rezolvat în acest caz. În primul rând trebuie să fie stabilită o strategie de rectificare şi în al doilea rând trebuie stabilită probabilitatea cu care vectorul rectificat înlocuieşte vectorul din care provine în populaţie. Această probabilitate poate fi cuprinsă între 0 şi 1. Orvosh şi Davis au stabilit *regula 5%* prin care se stabileşte că cele mai bune rezultate se obţin dacă se consideră probabilitatea de înlocuire de 0.05.

Procesul de rectificare se desfăşoară după următorul algoritm:

Algoritmul 2.3. (*Rectificare*)

1. [Iniţializare] Se face $x' = x$.
2. [Terminare] Dacă $\sum_{i=1}^{n} x'_i w_i \le M$, atunci STOP.
3. [Alegere] Se alege un element i din rucsac, adică un indice cu $x'_i = 1$.
4. [Eliminare] Se elimină elementul i din rucsac, adică se face $x'_i = 0$.
5. [Ciclare] Se merge la pasul 2. ■

Dintre strategiile de alegere din pasul 3 se pot menţiona următoarele două care duc la două tipuri de algoritmi prin rectificare:

- $A_r(1)$ (*rectificare aleatoare*): alegerea se face aleator.
- $A_r(1)$ (*rectificare Greedy*): alegerea se face prin luarea în considerare a elementului cu cel mai mic profit dintre cele din rucsac; dacă sunt mai nulte cu acelaşi profit se alege oricare dintre ele.

Algoritmii $A_d(i)$. Cei mai naturali decodificatori pentru problema rucsacului se bazează pe reprezentarea întreagă. Fiecare cromozom este un vector de n întregi,

https://biblioteca-digitala.ro / https://unibuc.ro

cel de al i-lea element al vectorului fiind un întreg cuprins între 1 şi $n - i + 1$. Reprezentarea aceasta se pune în legătură cu o listă de elemente L. Vectorul se decodifică prin considerarea şi eliminarea din lista curentă a elementului respectiv. De exemplu, începând cu lista $L = (1, 2, 3, 4, 5, 6)$, vectorul $x = (4, 3, 4, 1, 1, 1)$ se decodifică astfel: se consideră 4 obţinând lista $L = (1, 2, 3, 5, 6)$, se consideră 3 obţinând lista $L = (1, 2, 5, 6)$, se consideră 6 obţinând lista $L = (1, 2, 5)$, se consideră 1 obţinând lista $L = (2, 5)$, se consideră 2 obţinând lista $L = (5)$ şi, în final, se consideră 5 obţinând lista $L = ()$. Deci decodificarea vectorului este (4, 3, 6, 1, 2, 5).

Fiecare cromozom poate fi interpretat ca o strategie de introducere a elementelor în rucsac. În plus, aplicarea unei încrucişeri între doi cromozomi fezabili produce tot cromozomi fezabili. Operatorul mutaţie se defineşte similar ca pentru cazul binar: se alege la întâmplare pentru poziţia i orice număr cuprins între 1 şi $n - i + 1$. Algoritmul de decodificare este următorul:

Algoritmul 2.4. (***Decodificare***)

1. [Iniţializare] Se construieşte o listă de elemente L, se face $i = 1$, $W = 0$ şi $P = 0$.
2. [Terminare] Dacă $i > n$, atunci STOP.
3. [Eliminare] Se face $j = L(x_i)$ şi se elimină al x_i-lea element din lista L.
4. [Adăugare] Dacă $W + w_j \leq M$, atunci se face $W = W + w_j$ şi $P = P + p_j$ (se adaugă elementul j la rucsac).
5. [Ciclare] Se face $i = i + 1$ şi se merge la pasul 2. ■

Algoritmii bazaţi pe decodificare diferă doar prin modul de construire a listei L. Pot fi considerate, de exemplu următoarele două strategii:

- $A_d(1)$ (*decodificarea aleatoare*): se consideră o ordine aleatoare a elementelor, de exemplu ordinea de intrare.
- $A_d(1)$ (*decodificarea Greedy*): se pun în lista L în ordinea descrescătoare a profiturilor; dacă sunt mai multe elemente cu acelaşi profit se consideră o ordine oarecare.

S-au făcut experimentări pe populaţii de 100 de cromozomi cu probabilitatea de mutaţie 0.05 şi probabilitatea de încrucişare 0.65. Măsurarea performanţelor a fost făcută prin reţinerea celor mai bune soluţii obţinute din câte 500 de generaţii. S-a observat că pentru un număr mai mare de generaţii se produc, în general, puţine modificări. Rezultatele prezentate în tabelul 2.1. reprezintă valorile medii din câte 25 de experimente. Datele nu au fost sortate în raport de profit. Tipul de capacitate M_1 este cazul capacităţii restrictive şi M_2 este cazul cu capacitate aleatoare. Rezultatele din metodele $A_r(1)$ şi $A_r(2)$ s-au obţinut aplicând regula 5%. Orice alte probabilităţi de alegere a înlocuirii prin vector rectificat dau diferenţe nesemnificative.

https://biblioteca-digitala.ro / https://unibuc.ro

Corelație	Număr elem.	Tip capac.	metoda						
			$A_p(1)$	$A_p(2)$	$A_p(3)$	$A_r(1)$	$A_r(2)$	$A_d(1)$	$A_d(2)$
deloc	100	M_1	X	X	X	62.9	94.0	63.5	59.4
		M_2	398.1	341.3	342.6	344.6	371.3	354.7	353.3
	250	M_1	X	X	X	62.6	135.1	58.0	60.4
		M_2	919.6	837.3	825.5	842.2	894.4	867.4	857.5
	500	M_1	X	X	X	63.9	156.2	61.0	61.4
		M_2	1712.2	1570.8	1565.1	1577.4	1663.2	1602.8	1597.0
slab	100	M_1	X	X	X	39.7	51.0	38.2	38.4
		M_2	408.5	327.0	328.3	330.1	358.2	333.6	332.3
	250	M_1	X	X	X	43.7	74.0	42.7	44.7
		M_2	920.8	791.3	788.5	798.4	852.1	804.4	799.0
	500	M_1	X	X	X	44.5	93.8	43.2	44.5
		M_2	1729.0	1531.8	1532.0	1538.6	1624.8	1548.4	1547.1
tare	100	M_1	X	X	X	61.6	90.0	59.5	59.5
		M_2	741.7	564.5	564.4	566.5	577.0	576.2	576.2
	250	M_1	X	X	X	65.5	117.0	65.5	64.0
		M_2	1631.9	1339.5	1343.4	1345.8	1364.4	1366.4	1359.0
	500	M_1	X	X	X	67.5	120.0	67.1	64.1
		M_2	3051.6	2703.8	2700.8	2709.5	2748.1	2738.0	2744.0

Tabelul 2.1.

Principalele concluzii ce se pot trage din experimentări sunt următoarele:

- Algoritmii bazați pe penalități nu determină rezultate fezabile pentru problemele cu capacitate restrictivă (M_1) oricare ar fi numărul de elemente sau corelația.
- Pentru problemele cu capacitatea aleatoare a rucsacului (M_2) cele mai bune rezultate au fost obținute prin algoritmul cu penalitate logaritmică $A_p(1)$.
- Pentru problemele cu capacitatea rucsacului restrictivă (M_1) cele mai bune rezultate au fost obținute prin algoritmul cu rectificare Greedy $A_r(2)$.

2.3. Algoritmi aproximativi pentru probleme de aritmetică combinatorie

Punerea problemei. Se prezintă o tehnică generală de aproximare pentru o clasă largă de probleme NP-dificile în cadrul cărora se efectuează calcule aritmetice. Drept probleme reprezentative cunoscute se menționează problema rucsacului și cea a produsului submulțimilor. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Aplicații posibile: compilatoare de limbaje de programare, programarea calculatoarelor.

În continuare vom folosi notațiile:

$P_0(A)$ - mulțimea submulțimilor finite ale lui A;

t - funcție cu valori în $P_0(Z^+)$ calculată în timp polinomial cu algoritm nedeterminist;

$(A, t)_{Ext}$ - problemă de optimizare NP, unde Ext = max sau Ext = min.

$sol(a)$ - soluție a problemei $(A, t)_{Ext}$.

$op(a)$ - soluție optimă a problemei $(A, t)_{Ext}$.

Ap - algoritm aproximativ.

2.3.1. Modul de prezentare al calculelor

Fie dată $(a_1, a_2, ..., a_k) \in (Z^+)^k$ și k - 1 operații binare $\beta_1(x, y), ..., \beta_{k-1}(x, y)$, nu neapărat distincte, definite pe întregi. Fiecărei secvențe $(a_1, \beta_1, a_2, \beta_2, ..., a_{k-1}, \beta_{k-1}, a_k)$ i se asociază o secvență de calcule $(m_1, m_2, ..., m_k)$ definită astfel:

a) $m_1 = a_1$.

b) pentru $i = 1, ..., k - 1, m_{i+1} = \beta_i(m_i, a_{i+1})$.

De exemplu, pentru secvența de calcule $(3, x^y, 2, x + y, 1)$, secvența rezultatelor succesive este (3, 9, 10). Valoarea lui m_k se numește *rezultatul calculelor*.

Definiția 2.1. Pentru o mulțime dată B de operații binare, *problema* $Pr1(B)$ este următoarea: fiind dată secvența $(a_1, ..., a_n, b)$, să se găsească cel mai mare întreg k pentru care sunt satisfăcute condițiile:

1. pentru un $i_0 \in \{1, 2, ..., n\}$, k este rezultatul calculului efectuat cu secvența $\left(a_{i_0}, a_{i_0+1}, ..., a_n\right)$ pe B, și
2. $k \leq b$. ■

Fie operația binară u_1 definită astfel: $u_1(x, y) = x$. În acest caz, $Pr1(u_1, x, y)$ este o versiune a problemei rucsacului numită *problema submulțimii de sumă maximă* și care se formulează astfel: fiind dată secvența $(a_1, ..., a_n, b)$, să se maximizeze $\Sigma\ \varepsilon_i p_i$ cu $\varepsilon_i \in \{0, 1\}$ astfel încât $\Sigma\ \varepsilon_i a_i \leq b$.

Definiția 2.2. Operația $\beta \in B$ este *crescătoare* dacă $\beta(x, y) \geq x$ pentru toți $x, y > 0, x, y \in Z$. ■

Operațiile $x + y$, xy, x^y, y^x, u_1 sunt crescătoare, pe când $x - y$ și $y - x$ nu sunt crescătoare.

În continuare, se prezintă un algoritm care, pentru fiecare secvență $a = (a_1, a_2, ..., a_n, b)$ furnizează valoarea lui $op(a)$ într-un timp de ordinul $O(n\ op(a) \log\ \ )(a))$.

https://biblioteca-digitala.ro / https://unibuc.ro

Algoritmul 2.4. (***Calcul-optim***) **Intrare :** $a = (a_1, ..., a_n, b)$. **Ieşire :** $op(a)$.

1. [Iniţializări] Se face $i = 1$ şi $T = \emptyset$.
2. [Completare T] Dacă $T \neq \emptyset$, atunci, pentru fiecare s din T, se face: $T = T \cup \{\beta(s, a_i) \mid \beta \in B \text{ şi } \beta(s, a_i) \leq b\}$.
3. [Adăugare element nou] Se face $T = T \cup \{a_i\}$.
4. [Terminare] Dacă $i = n$, atunci STOP (furnizează $\max(T)$).
5. [Ciclare] Se face $i = i + 1$ şi se merge la pasul 2. ■

Este uşor de văzut că la terminarea parcurgerii algoritmului 2.4. valoarea lui $\max(T)$ este $op(a)$. Utilizând o structură convenabilă pentru reprezentarea lui T (de exemplu, arborii 2-3 din paragrafele 4.9-4.10 din [AHO74]) se poate arăta că timpul de execuţie al paşilor 2-5 este de ordinul $O(|B|\ |T|\ \log|T|)$. Timpul de execuţie al celorlalţi paşi este constant. $|B|$ este constant, deoarece T conţine numai întregi pozitivi mai mici sau egali cu $op(a)$ pe tot parcursul executării algoritmului. Rezultă că timpul de execuţie al algoritmului este de ordinul $O(n\ op(a) \log op(a))$.

2.3.2. Metoda condensării

Definiţia 2.3. Fie $(m_1, m_2, ..., m_k)$ secvenţa rezultatelor obţinute după efectuarea calculelor descrise de $(a_1, \beta_1, a_2, \beta_2, ..., a_{k-1}, \beta_{k-1}, a_k)$ şi fie $0 < \delta \leq 1$. O secvenţă $(m'_1, m'_2, ..., m'_k)$ este o *δ-condensare* a secvenţei $(m_1, m_2, ..., m_k)$ dacă există o secvenţă $(h_1, h_2, ..., h_k) \in R^k$ cu $0 \leq h_i \leq \delta$, astfel încât:

a) $m'_1 = m_1 (1 - h_1) = a_1 (1 - h_1)$,

b) $m'_{i+1} = [\beta_i(m'_i, a_{i+1})](1 - h_{i+1})$, pentru $i = 1, 2, ..., k - 1$. ■

Definiţia 2.4. Un algoritm A_p este *ε-aproximativ* pentru probleme $(A, t)_{max}$, dacă pentru toţi $a \in A$, $A_p(a) \in t(a)$ şi $A_p(a) \geq (1 - \varepsilon)\ op(a)$. ■

Definiţia 2.5. Un algoritm $(A, t)_{max}$ este *complet aproximativ* dacă pentru orice $\varepsilon > 0$ există un algoritm ε-aproximativ al lui $(A, t)_{max}$ având complexitatea $Q(l(a), 1/\varepsilon)$, unde Q este un polinom în două variabile şi $l(a)$ este lungimea lui a. ■

Algoritmul 2.3. descris în continuare este un algoritm complet polinomial pentru $Pr1(B)$, unde B este orice submulţime a lui $\{x + y, xy, u_1\}$.

Algoritmul 2.5. (***Calcul-limită***): **Intrare :** $a = (a_1, ..., a_n, b) \in (Z^+)^{n+1}$, $\varepsilon > 0$.
Ieşire: un întreg $k \in t(a)$ **astfel încât** $op(a) \geq k \geq op(a)\ (1 - \varepsilon)$.

1. [Iniţializare] Se face $r = \max(1/\varepsilon, n)$, $\delta = 1 / r^2$ (δ este parametrul de condensare), $i = 1$ şi $T = \emptyset$.

https://biblioteca-digitala.ro / https://unibuc.ro

2. [Completare] Dacă $T \neq \emptyset$, atunci se face: $T = T \cup \{\beta(s, a_i) \mid \beta \in B$ şi $\beta(s, a_i) \leq b\}$, pentru fiecare s din T.
3. [Adăugare element nou] Se face $T = T \cup \{a_i\}$.
4. [Sortare] Se sortează T (se presupune că $T = (s_1, ..., s_k)$, $s_i < s_{i+1}$).
5. [Terminare] Dacă $i = n$, atunci STOP (furnizează S_i).
6. [Iniţializare condensare] Se face $j = 1$, $k = 2$.
7. [Eliminări] Atâta timp cât $k \leq t$ şi $s_j / s_k > 1 - \delta$, se face $T = T - \{s_k\}$ şi $k = k + 1$.
8. [Ciclare eliminări] Dacă $k<t$, atunci se face $j = k$ şi $k = k+1$ şi se merge la pasul 7.
9. [Ciclare] Se face $i \leftarrow i + 1$ şi se merge la pasul 2. ■

Teorema 2.4. Pentru fiecare $\varepsilon>0$, algoritmul 2.5. dă o ε-aproximare pentru $Pr1(B)$, unde $B \subset \{x + y, xy, u_1\}$ de complexitate $O(\max\{l^4(a), l^2(a) / \varepsilon\})$ în timp. ■

Exemplul 2.2. Modul în care operează algoritmul 2.4. poate fi ilustrat pe următorul exemplu al problemei rucsacului: $(a_1, a_2, a_3, a_4, b) = (1, 3, 4, 6, 7)$.

i	a_i	T
1	1	$T = \emptyset$, $T = \{1\}$
2	3	$T = \{1\} \cup \{1 + 3\}$, $T = \{1, 1 + 3, 3\}$
3	4	$T = \{1, 1 + 3, 3, 1 + 4, 3 + 4\}$, $T = \{1, 1 + 3, 3, 1 + 4, 3 + 4, 4\}$
4	6	$T = \{1, 1 + 3, 3, 1 + 4, 3 + 4, 4, 1 + 6\}$

Tabelul 2.2.

Modul de efectuare a calculelor este prezentat în tabelul 2.2. Rezultă soluţiile: $3 + 4 = 7$ şi $1 + 6 = 7$. Mulţimea T a elementelor distincte este $T = \{1, 4, 3, 5, 7\}$ şi se vede că $|T| = 5 \leq op(a)$.

2.4. Selectarea şi ordonarea dispozitivelor de orientare

Punerea problemei. Pentru asamblarea automată a unor aparate este necesară alimentarea maşinilor de montaj cu piesele componente corect orientate. Orientarea acestor piese se realizează cu ajutorul unor dispozitive conectate în serie, fiecare dispozitiv efectuând o anumită operaţie de orientare. Problema de rezolvat constă în calculul eficienţei unui şir de dispozitive de orientare, adică a probabilităţii unei orientări corecte a unei piese după parcurgerea dispozitivelor. ■

Aplicaţii posibile: fluxuri de producţie, formarea garniturilor de tren.

Enunţul problemei pentru două orientări. Pentru cazul, cel mai simplu, a două orientări, o mulţime $N = \{1,2,...,n\}$ de dispozitive şi o mulţime $M = \{1,2,...,m\}$ de piese, eficienţa unui dispozitiv este definită, pentru fiecare $i \in N$, $j \in M$, de o matrice 2×2 de forma:

https://biblioteca-digitala.ro / https://unibuc.ro

$$P_{ij} = \begin{bmatrix} a_{ij} & b_{ij} \\ c_{ij} & d_{ij} \end{bmatrix},$$

cu condiţiile $0 \le a_{ij}, b_{ij}, c_{ij}, d_{ij} \le 1$, $a_{ij} + b_{ij} \le 1$, $c_{ij} + d_{ij} \le 1$. Fie η_j probabilitatea ca o piesă de tip j să fie corect orientată înainte de a intra în dispozitivul i şi Q_j numărul de piese de tip j. Atunci $I_j = [\eta_j, 1 - \eta_j]$ este distribuţia orientărilor iniţiale ale piesei j şi $S_j = Q_j / \sum_{j=1}^{n} Q_j$ proporţia pieselor de tip j. Dacă se notează cu η_{jk} probabilitatea orientării corecte a piesei j după ce a trecut prin dispozitivul k, atunci orientarea la ieşire $\sigma_{jk} = [\eta_{jk}, 1 - \eta_{jk}]$ este egală cu produsul $I_j \cdot P_{ij}$, şi η_{jk} are valoarea $\eta_j a_{ij} + (1 - \eta_j) c_{ij}$ şi se numeşte *eficienţa dispozitivului* k pentru piesa j având orientarea iniţială I_j.

Un sistem serial de orientare poate fi definit ca fiind o permutare pe mulţimea celor N dispozitive. Fiind dată o permutare $\psi = \{i_1, i_2, ..., i_n\}$, sistemul poate fi definit drept o colecţie de mulţimi ordonate de matrici:

$$\left\{P_j(\psi) = P_{i_1 j}, P_{i_2 j}, \ldots, P_{i_m j}, j \in M\right\},$$

unde $P_j(\psi)$ este mulţimea ordonată a matricilor P_{ij} pentru piesa j. Eficienţele corespunzătoare, adică $\eta_j(\psi), j \in M$, sunt date de:

$$[\eta_j(\psi), 1 - \eta_j(\psi)] = I_j P_j(\psi) = [\eta_j, 1 - \eta_j] P_j(\psi).$$

Eficienţa ponderată a sistemului este:

$$\eta(\psi) = \sum_{j \in M} S_j \eta_j(\psi),$$

unde S_j sunt date.

Ipoteze. În continuare sunt enunţate trei ipoteze privind P_{ij} în vederea simplificării analizei euristicilor propuse. Aceste ipoteze sunt satisfăcute în multe situaţii reale.

1. $0 \le a_{ij}, c_{ij} \le 1$.
2. Piesele nu sunt respinse de nici un dispozitiv, adică $a_{ij} + b_{ij} = c_{ij} + d_{ij} = 1$.
3. Piesele corect orientate sunt preferate de dispozitivul folosit, adică $a_{ij} \ge c_{ij}$.

Teorema 2.5. Dacă matricile P_{ij} satisfac ipotezele 1,2,3, atunci matricile mulţimii $P_j(\psi)$ satisface aceleaşi ipoteze pentru fiecare $j \in M$ şi fiecare permutare ψ a mulţimii N.

Demonstraţia fiind simplă este omisă. ■

Problema selecţiei şi orientării dispozitivelor. Fie m piese şi n dispozitive, nu neapărat distincte. Fiind dată distribuţia orientărilor tuturor pieselor şi caracteristicile fiecărui dispozitiv, pentru fiecare piesă în parte, este interesantă problema selectării a k sau mai puţine dispozitive ($k \le n$) şi ordonarea lor în vederea maximizării eficienţei ponderate a sistemului rezultat. Pentru formalizarea problemei este necesară definirea

https://biblioteca-digitala.ro / https://unibuc.ro

mulţimii sistemelor seriale realizabile. Dacă se pot selecta maximum k dispozitive ($k \leq n$), atunci mulţimea G_k a sistemelor realizabile constă din toate permutările fiecărei submulţimi a lui N conţinând k sau mai puţine dispozitive. Problema de rezolvat constă în:

$$\text{maximizarea lui } \eta(\psi), \text{ unde } \psi \in G_k \text{ fiind date } I_j, P_{ij}, i \in N, j \in M \text{ şi } k \leq n. \quad (2.1)$$

2.4.1. Ordonarea optimală a dispozitivelor în cazul unei singure piese

În acest paragraf, se studiază cazul particular al problemei enunţate mai înainte pentru $k = n$ şi $m = 1$. Indicele j poate fi suprimat, orientarea este definită prin $I = [\eta, 1 - \eta]$ şi eficienţa dispozitivului i, cu $1 \leq i \leq n$, este:

$$P_i = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix}.$$

Matricea de tranziţie limită este definită prin:

$$P_i^\infty = \lim_{L \to \infty} (P_i)^L.$$

Ţinând seama de ipotezele 1 şi 2, se poate arăta că:

$$P_i^\infty = \begin{bmatrix} \dfrac{c_i}{b_i + c_i} & \dfrac{b_i}{b_i + c_i} \\ \dfrac{c_i}{b_i + c_i} & \dfrac{b_i}{b_i + c_i} \end{bmatrix}.$$

Calculând produsul $I \cdot P_i^\infty$ se obţine vectorul:

$$[\eta, 1 - \eta] = [c_i / (b_i + c_i), b_i / (b_i + c_i)],$$

adică are loc:

Teorema 2.6. Eficienţa asimptotică a dispozitivului P_i este $c_i / (b_i + c_i)$. ■

Teorema 2.7. Într-un sistem cu două dispozitive:

$$b_1 / c_1 > (=, \text{respectiv} <)\ b_2 / c_2 \Rightarrow \eta(1, 2) > (=, \text{respectiv} <)\ \eta(2, 1).$$

Demonstraţie. Este suficient să se demonstreze prima implicaţie, adică $b_1/c_1 > b_2/c_2$ implică $\eta(1, 2) > \eta(2, 1)$. Presupunând implicaţia neadevărată, adică $\eta(1, 2) \leq \eta(2, 1)$ şi efectuând calculele ce furnizează pe $\eta(1, 2)$ şi $\eta(2, 1)$ rezultă că $\eta(a_1a_2 + b_1c_2) + (1 - \eta)(c_1a_2 + d_1c_2) \leq \eta(a_1a_2 + b_2c_1) + (1 - \eta)(c_2a_1 + d_2c_1)$, unde $[\eta, 1 - \eta]$ este o distribuţie arbitrară a orientării iniţiale. Efectuând reducerile se obţine $c_1 / b_1 \geq c_2 / b_2$ deci se ajunge la o contradicţie. Rezultă că prima implicaţie este adevărată. În mod similar se demonstrează celelalte implicaţii. ■

În cele ce urmează, se presupune, fără a micşora generalitatea, că dispozitivele satisfac condiţia:

$$c_1 / b_1 \leq c_2 / b_2 \leq \ldots \leq c_n / b_n. \quad (2.2)$$

https://biblioteca-digitala.ro / https://unibuc.ro

Teorema 2.8. În ipoteza (2.2), ordinea optimală a unui sistem cu n dispozitive este $\psi^* = \{1, 2, ..., n\}$.

Demonstraţie. Fie ψ o permutare diferită de ψ^*. Atunci există două dispozitive P_r şi P_s astfel încât P_r este primul predecesor al lui P_s în ψ şi $c_r / b_r \geq c_s / b_s$. Se poate arăta că schimbând între ele dispozitivele P_r şi P_s rezultă o ordonare ψ' astfel încât $\eta(\psi) \geq \eta(\psi')$. Fie η' şi η'' orientările piesei considerate pe P_r, respectiv P_s. În virtutea teoremei 2.7., orientarea η''' a piesei la ieşirea din P_r în ordonarea ψ' satisface relaţia $\eta''' = \eta'' + \delta$, cu $\delta \geq 0$. Dar în ambele ordonări ψ şi ψ' dispozitivele ce urmează după P_r şi P_s pot fi reprezentate printr-un dispozitiv compus:

$$P_0 = \begin{bmatrix} a_0 & b_0 \\ c_0 & d_0 \end{bmatrix},$$

unde $a_0 \geq c_0$ conform teoremei 2.5. Rezultă că au loc relaţiile:

$$\eta(\psi') = (\eta'' + \delta)\, a_0 + (1 - \eta'' - \delta)\, c_0 \text{ şi } \eta(\psi) = \eta''\, a_0 + (1 - \eta'')\, c_0.$$

Deoarece $\delta \geq 0$ şi $a_0 \geq c_0$, $\eta(\psi') \geq \eta(\psi)$. Un număr finit de asemenea schimbări transformă pe ψ în ψ^*, eficienţa nedescrescând la fiecare schimbare. Prin urmare ψ^* este o succesiune optimală. ■

Teorema 2.9. Fiind date condiţiile (2.2) şi o orientare iniţială η, soluţia optimală a problemei (2.1), cu $k = n$, adică soluţia optimală a problemei ordonării a n sau mai puţine dispozitive, este dată de $\psi_0^* \in G_n$, unde ψ_0^* se obţine din ψ^* eliminând fiecare dispozitiv cu o eficienţă:

$$c_i / (b_i + c_i) \leq \eta. \tag{2.3}$$

Demonstraţie. Este uşor de dedus că (2.2) implică:

$$c_1 / (b_1 + c_1) \leq c_2 / (b_2 + c_2) \leq ... \leq c_n / (b_n + c_n).$$

Fie $N' \subset N$ o mulţime de dispozitive ce satisface (2.3), cu $N' \neq \varnothing$. Evident că $N' = \{1, 2, ..., r\}$ pentru un $r \leq n$. În cele ce urmează se arată că eficienţa sistemului creşte dacă se elimină dispozitivul $i = 1$.

În $\psi^* = \{1,2,...,n\}$ dispozitivele ce urmează după P_1 pot fi reprezentate printr-un dispozitiv compus P_0 pentru care $a_0 \geq c_0$ în virtutea teoremei 2.5. Fie η_1 orientarea la ieşirea din dispozitivul P_1 şi ordonarea $\psi_1^* = \{2, 3, ..., n\}$. Au loc relaţiile $\eta(\psi_1^*) = \eta\, a_0 + (1 - \eta)\, c_0$ şi $\eta(\psi^*) = \eta_1\, a_0 + (1 - \eta_1)\, c_0$. Din (2.3) rezultă că $\eta_1 \leq \eta$ şi, deoarece $a_0 \geq c_0$, $\eta(\psi_1^*) \geq \eta(\psi^*)$.

Procedura descrisă se repetă pentru $i = 2, 3, ..., r$. Sistemul rămas este definit de ordonarea $\psi_0^* = \{r + 1, r + 2, ..., n\}$, cu $\eta(\psi_0^*) \geq \eta(\psi)$. Dacă $r = n$, atunci $\psi_0^* = \varnothing$. Rămâne de arătat că eliminarea oricărui dispozitiv $i \geq r + 1$ nu poate îmbunătăţi eficienţa sistemului. Acest lucru se demonstrează în mod analog celui descris mai sus. Aceasta completează demonstraţia teoremei. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Teorema 2.10. Să presupunem că în ordonarea ψ_0^* din teorema 2.9. există un dispozitiv P_i cu proprietatea $a_i = c_i$. Atunci eliminarea tuturor dispozitivelor care preced pe P_i în ψ_0^* nu schimbă eficienţa sistemului şi, în consecinţă, reprezintă o altă soluţie optimală.

Demonstraţie. Afirmaţia rezultă din observaţia că eficienţa lui P_i este a_i independent de orientarea η la intrare. Evident că un sistem optimal cu număr minim de dispozitive se obţine pe calea identificării celui mai din dreapta dispozitiv P_i din ψ_0^* şi eliminarea tuturor dispozitivelor din stânga lui P_i. ■

Corolarul 1. Fiind date (2.2), $c_1 / (b_1 + c_1) > \eta$, orientarea iniţială şi $a_i > c_i$, $i \in N$, unica soluţie optimală a problemei (2.1) este $\psi^* = \{1, 2, ..., n\}$.

Demonstraţie. Concluzia rezultă din teoremele 2.9 şi 2.10. ■

2.4.2. Selectarea şi ordonarea dispozitivelor în cazul unei piese

În acest paragraf, se consideră problema (2.1), cu $1 \le k \le n$, $m = 1$. În virtutea corolarului 1 se presupune că $c_i/(b_i+c_i)>\eta$ şi $a_i>c_i$, pentru $i\in N$, şi că are loc ordonarea (2.2). De asemenea se presupune că $a_i > c_i$, pentru toate dispozitivele din mulţimea N.

Formularea în termeni de programare dinamică. Fie $F_t(p)$ eficienţa maximă la ieşirea din dispozitivul t, dacă exact p=1,2,...,min(t,k) dispozitive au fost selectate dintre primele t dispozitive ale unui n, sistem optimal. Relaţia recursivă are forma:

$$F_t(p) = \max\{F_{t-1}(p), a_t F_{t-1}(p-1) + c_t (1 - F_{t-1}(p-1))\}, \tag{2.4}$$

unde:

$$F_t(0) = \eta, \text{ pentru toţi } t = 1, 2, ..., n \text{ şi } F_1(1) = \eta_1 a_1 + (1 - \eta_1) c_1. \tag{2.5}$$

Cu ajutorul relaţiilor (2.4) şi (2.5) se poate elabora selectarea şi ordonarea optimală a k < n dispozitive.

Proceduri euristice. În continuare se presupune că ordonarea optimală a n dispozitive este $\psi^* = \{1, 2, ..., n\}$.

Euristica 1. În această euristică, cele k dispozitive se aleg la întâmplare din ψ^*. Această euristică nu are o motivare raţională şi este folosită numai pentru compararea cu euristicile ce urmează.

Euristica 2. Această metodă selectează ultimile k dispozitive din ordonarea ψ^*, adică dispozitivele P_{n-k+1}, P_{n-k+2}, ..., P_n. Aceste dispozitive au eficienţe asimptotice mari şi din acest motiv se consideră bine alese.

Exemplul 2.3. Fie $n = 3$, $k = 2$ şi $m = 1$. Matricile dispozitivelor sunt:

$$P_1 = \begin{bmatrix} 0.8 & 0.2 \\ 0.7 & 0.3 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 0.9 & 0.1 \\ 0.6 & 0.4 \end{bmatrix}, \quad P_3 = \begin{bmatrix} 0.9 & 0.1 \\ 0.8 & 0.2 \end{bmatrix}.$$

https://biblioteca-digitala.ro / https://unibuc.ro

Orientarea inițială este $\eta = 0.15$ și $\psi^* = \{1, 2, 3\}$. Ieșirile asimptotice ale celor trei dispozitive sunt respectiv 0.78, 0.86 și 0.89. Euristica 2 generează soluția $P_2\ P_3$ având eficiența 86.45% și sistemul optimal este $P_1\ P_3$ cu eficiența 87.15%. ■

Euristica 3. Metoda acestei euristici constă în selectarea a k dispozitive în conformitate cu performanța lor relativ la orientarea inițială η, adică în conformitate cu valorile maxime ale lui $\eta_i = \eta\, a_i + (1 - \eta)\, c_i$, $1 \le i \le n$. Apoi ele se ordonează ca în ψ^*. În cazul exemplului precedent, euristica 3 selectează pe P_1 și P_3, care reprezintă soluția optimală.

Exemplul 2.4. Fie $n = 5$, $k = 4$, $m = 1$ și matricile P_1, P_2, P_3, P_4, P_5 sunt.

$$P_1 = P_2 = P_3 = \begin{bmatrix} 0.8 & 0.2 \\ 0.7 & 0.3 \end{bmatrix} \quad P_4 = \begin{bmatrix} 0.9 & 0.1 \\ 0.6 & 0.4 \end{bmatrix} \quad P_5 = \begin{bmatrix} 0.9 & 0.1 \\ 0.8 & 0.2 \end{bmatrix}.$$

Orientarea inițială este $\eta = 0.15$ și $\psi^* = \{1, 2, 3, 4, 5\}$. Euristica 3 identifică soluția $P_1\ P_2\ P_3\ P_5$ cu eficiența 88.22%. Soluția optimală este $P_1\ P_2\ P_4\ P_5$ și are eficiența 88.31%. ■

Euristica 4. Această euristică selectează k dispozitive în ordinea definită de ψ^* schimbând valoarea lui η în modul descris mai jos. Fie $\psi^* = \{0, 1, ..., n\}$, unde 0 este indicele unui dispozitiv neutru cu matricea $P_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ utilizat pentru inițializarea procesului.

1. [Inițializări] Se face $P = P_0$, $k_0 = k$, $\psi = \varnothing$, $\eta_0 = \eta$ (orientarea inițială).
2. [Construcție inițială] Se construiește $\psi^* = \{0, 1, 2, ..., n\}$, pentru un sistem optimal D_n cu n dispozitive, unde $D_n = P_0\, P_1\, P_2 \dots P_n$.
3. [Calcul eficiență] Se calculează eficiența tuturor dispozitivelor ce urmează după P_0 în D_n, adică $\eta_i = \eta_0\, a_i + (1 - \eta_0)\, c_i$, pentru $i = 1, ..., n$.
4. [Alegere] Se identifică k dispozitive ce corespund la cele mai mari k valori ale lui η_i. În cazul egalității a două valori ale eficienței, se alege dispozitivul situat la stânga în ψ^*.
5. [Selectare] Se identifică cel mai din stânga dispozitiv f identificat în pasul 4. Fie $P = P_f$ și $\psi = \{\psi, f\}$.
6. [Ajustare] Se face $\eta_0 = \eta_0\, a_f + (1 - \eta_0)\, c_f$ ținând seama că $P_f = \begin{bmatrix} a_f & b_f \\ c_f & d_f \end{bmatrix}$.
7. [Ciclare] Se face $k_0 = k_0 - 1$. Dacă $k_0 \ge 1$, atunci se trece la pasul 3.
8. [Terminare] Eficiența sistemului cu k dispozitive este η_0 și ordinea dispozitivelor este definită de ψ. STOP. ■

https://biblioteca-digitala.ro / https://unibuc.ro

2.4.3. Selectarea şi ordonarea dispozitivelor pentru *m* tipuri de piese

Într-un sistem cu *m* tipuri de piese, matricea asociată piesei de tip *j* este de forma:

$$P_{ij}=\begin{bmatrix} a_{ij} & b_{ij} \\ c_{ij} & d_{ij} \end{bmatrix}, \text{ pentru } i = 1, 2, ..., n \text{ şi } j = 1, 2, ..., m.$$

În acest caz se efectuează o conversie a problemei date într-una cu $m = 1$ folosind matrici de forma:

$$P_i=\begin{bmatrix} \sum_{j=1}^{m} S_j a_{ij} & \sum_{j=1}^{m} S_j b_{ij} \\ \sum_{j=1}^{m} S_j c_{ij} & \sum_{j=1}^{m} S_j d_{ij} \end{bmatrix}, \text{ pentru } i = 1, 2, ..., n,$$

unde S_j, $1 \le j \le m$ sunt ponderi asociate tipurilor pieselor. După efectuarea acestei conversii se aplică euristicile precedente.

2.4.4. Selectarea şi ordonarea dispozitivelor în cazul a *m* tipuri de piese cu *l* orientări

În acest caz, matricea de tranziţie este o matrice $l \times l$ ($l > 2$) de forma $P_{ij}=\left[e_{ij}^{pq}\right]_{l \times l}$, $i \in N$, $j \in M$ unde e_{ij}^{pq} este probabilitatea ca piesa de tip j intrată în dispozitivul i cu orientarea p părăseşte dispozitivul i cu orientarea q. Elementele matricii P_{ij} satisfac condiţia $0 \le e_{ij}^{pq} \le 1$.

Euristica propusă are la bază ideea conversiei matricilor $l \times l$ în matrici 2×2. O matrice convertită P'_{ij} are forma:

$$P'_{ij}=\begin{bmatrix} a_{ij} & b_{ij} \\ c_{ij} & d_{ij} \end{bmatrix}, \text{ unde } a_{ij} = e_{ij}^{11},\ b_{ij} = \sum_{k=2}^{l} e_{ij}^{1k},$$

$$c_{ij} = \sum_{k=2}^{l} \frac{\eta_j^k}{\eta_j^2 + \eta_j^3 + ... + \eta_j^l} \cdot e_{ij}^{k1},\ d_{ij} = \sum_{k=2}^{l} \frac{\eta_j^k}{\eta_j^2 + \eta_j^3 + ... + \eta_j^l} \times \sum_{t=2}^{l} e_{ij}^{kt}.$$

În felul acesta problema studiată este redusă la cea precedentă (paragraful 2.4.3) care la rândul ei se reduce la problema studiată în paragraful 2.4.2.

2.5. Algoritmi aproximativi pentru problema superşirului minimal

Punerea problemei. Fiind date mai multe şiruri de caractere, se cere identificarea unui şir de lungime minimală care să acopere toate şirurile date, adică este un superşir, în sensul că oricare dintre şiruri este un subşir al şirului determinat. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Aplicații posibile: codificări date, proiectarea unor fluxuri industriale.

Exemplul 2.5. Fie S = {egiach, bfgiak, hfdegi, iakfd, fgiakh} o mulțime de cinci șiruri. Șirul bfgiakhfdegiach este un superșir al șirurilor din S. ■

În continuare se presupune că șirurile din mulțimea studiată satisfac condiția că nici unul dintre ele nu este *subșir* al altuia. Fie $s_1 = a_1 \ldots a_r$ și $s_2 = b_1 \ldots b_s$ două șiruri ce satisfac condiția precedentă. Acoperirea maximală a acestor două șiruri are lungimea definită prin:

$$\Psi(s_1, s_2) = \max\{k \geq 0 \mid a_{r-k+i} = b_i,\ 1 \leq i \leq k\}.$$

Dacă $\Psi(s_1, s_2) = k$, atunci șirul $s_1 \circ s_2$ este definit prin $a_1 \ldots a_r b_{k+1} \ldots b_s$. Operația o este asociativă, adică $s_1 \circ (s_2 \circ s_3) = (s_1 \circ s_2) \circ s_3$.

Problema superșirului maxim (*PSM*) este legată, într-un sens ce urmează să fie precizat, cu problema drumului maximal (*PDM*) într-un graf. Fiind dat un graf orientat complet $G = (V, E)$ în care fiecare arc (u, v) are o lungime întreagă nenegativă $l(u, v)$, lungimea unui drum p în G este definită ca sumă a lungimilor arcelor sale și se notează cu $\lambda_p(G, l)$. Problema drumului maximal constă în identificarea unui drum hamiltonian p (adică un drum care include orice nod) în G care maximizează pe $\lambda_p(G, l)$. Lungimea unui asemenea drum este notată cu $\lambda^*(G, l)$.

Fie $S = (s_1, \ldots, s_n)$ datele unei probleme *PSM*. *PDM*(S) definește datele (G, l) ale problemei *PDM* cu:

$$V = \{u_1, \ldots, u_n\},\ E = V \times V,\ l(u_i, u_j) = \Psi(s_i, s_j),\ 1 \leq i, j \leq n,\ i \neq j.$$

Un exemplu ce ilustrează această transformare este arătat în figura 2.4.

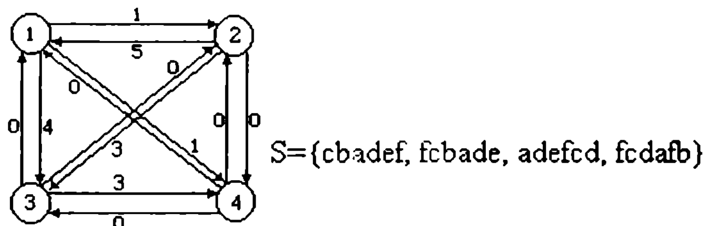


Fig. 2.4.

În continuare vom prezenta câțiva algoritmi aproximativi pentru rezolvarea problemei *PDM*.

2.5.1. Algoritmul împachetării

O *împachetare* într-un graf $G = (V, E)$ este o mulțime de muchii ce nu au un nod comun. O împachetare M este maximă dacă lungimea ei, adică suma lungimilor

https://biblioteca-digitala.ro / https://unibuc.ro

arcelor respective, notată $l(M)$, este maximă. Valoarea unei împachetări maxime este dată de formula $\mu^*(G, l) = \max_M l(M)$. Există algoritmi pentru identificarea unei împachetări maxime având o complexitate temporală de $O(n^3)$ unde $n = |V|$. Pentru problema *PDM* se poate folosi următoarea procedură numită *MATCH*.

Algoritmul 2.6. (*MATCH*).

1. [Împachetare] Se identifică o împachetare maximală M în G şi se calculează valoarea ei.
2. [Eliminări arce] Pentru $(u, v) \in M$, se elimină din G toate arcele de forma (u, x) sau (y, v).
3. [Unificare] Se unifică u şi v într-un singur nod.
4. [Ciclare] Dacă există arce neparcurse, atunci se trece la pasul 1, altfel STOP. ■

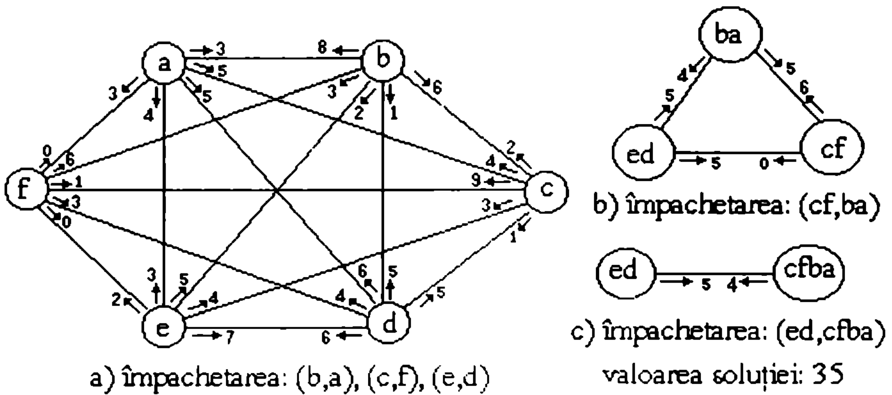


Fig. 2.5.

Exemplul 2.6. Un exemplu ilustrativ este arătat în figura 2.5.a). Prima împachetare maximală este (b, a), (c, f), (e, d) şi are valoarea $8 + 9 + 7 = 24$. Eliminând arcele divergente din b, c, e respectiv cele convergente în a, f, d, unificând vârfurile (b, a), (c, f), (e, d) se obţine graful din figura 2.5.b). Reluând pasul 1, rezultă împachetarea (cf, ba) ce are valoarea 6. În continuare se obţine graful din figura 2.5.c) pentru care se obţine împachetarea $(ed, cfba)$ de cost 5. Rezultă că drumul maxim identificat are lungimea 35. ■

2.5.2 Algoritmul împachetării direcţionate

O *împachetare direcţionată* într-un digraf $G = (V, E)$ este o mulţime de arce ce satisfac condiţia că nici o pereche din ele nu converge într-un acelaşi nod şi nici nu diverge dintr-un acelaşi nod. Altfel spus, o împachetare direcţionată este o colecţie de

https://biblioteca-digitala.ro / https://unibuc.ro

drumuri şi cicluri disjuncte. Algoritmul prin care se determină o împachetare direcţionată are următoarea structură.

Algoritmul 2.7. (Împachetare direcţionată)

1. [Împachetare] Se identifică o împachetare maximală direcţionată M în G.
2. [Eliminare cicluri] Se construieşte $M' := M$ - un arc de cost minimal din fiecare ciclu al lui M.
3. [Transformări graf] Pentru fiecare drum $(u_1, u_2, ..., u_n) \in M'$ se elimină din G:
   a) arcele de forma (u_i, x), $2 \leq i \leq r$;
   b) arcele de forma (x, u_i), $1 \leq i \leq r - 1$;
   c) arcul (u_r, u_1), dacă există;
   d) se compactează drumul într-un singur vârf.
4. [Ciclare] Dacă există arce neparcurse, atunci se trece la pasul 1, altfel STOP. ■

Exemplul 2.7. Pentru graful din figura 2.5, împachetarea maximală direcţionată (pasul 1) este arătată pe figura 2.6.a). Eliminând arcele (a, c), respectiv (d, e), (pasul 2) rezultă drumurile din figura 2.6.b) care au costurile 23 respectiv 7. După parcurgerea pasului 3 se obţine graful din figura 2.6.c). Împachetarea maximă direcţionată corespunzătoare este: $(ed, cfba)$, $(cfba, ed)$. După parcurgerea pasului 2 se obţine perechea $(ed, cfba)$ de cost 5. Drumul căutat are costul (lungimea) 35 şi coincide cu cel din exemplul precedent. ■

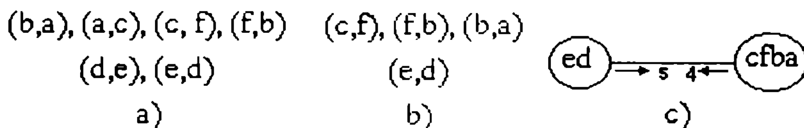


Fig. 2.6.

2.5.3. Algoritmul de tip Greedy

În cadrul acestui algoritm, arcele sunt parcurse în ordinea nedescrescătoare a lungimii, un arc (u, v) fiind selectat dacă anterior nu a fost selectat un arc de forma (u, x) sau (y, v) şi dacă mulţimea arcelor selecţionate nu include un drum de la v la u.

Exemplul 2.8. Pentru graful din figura 2.5.a), algoritmul selectează arcele (c, f), (b, a), (e, d), (f, b), (d, c), în această ordine. Aceste arce definesc drumul (e, d, c, f, b, a) de lungime: $7 + 5 + 9 + 6 + 8 = 35$. ■

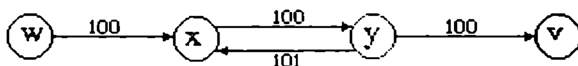


Fig. 2.7.

https://biblioteca-digitala.ro / https://unibuc.ro

Acest algoritm nu furnizează întotdeauna o soluţie apropiată de cea optimală, după cum rezultă din figura 2.7. pentru care soluţia optimală are valoarea 300 şi cea furnizată de algoritm, valoarea 101.

Algoritmul Greedy pentru *PSM*. Algoritmul Greedy pentru *PDM* poate fi reformulat pentru *PSM* după cum urmează.

Fiind dată o mulţime nevidă de şiruri *S*, se repetă pasul următor până când *S* va conţine un singur şir:

Se selectează o pereche de şiruri s_1, $s_2 \in S$ ce maximizează $\Psi(s_1, s_2)$, apoi se elimină s_1 şi s_2 din *S* înlocuindu-le cu s_1 o s_2.

Teorema 2.11. Fie *S* o mulţime de şiruri. Are loc următoarea inegalitate: $\Psi^*(S) \leq 2\,\Psi SGREEDY$, unde *SGREEDY* este valoarea din algoritmul precedent. ■

Complexitatea algoritmului *SGREEDY* este de cel puţin $O(n^2)$. În continuare se prezintă un algoritm ce utilizează reprezentarea datelor prin arbori sufix *trie* şi care are o complexitate logaritmică.

Problema *PSM* şi problema comisvoiajorului. Fie $S = (s_1,s_2,...,s_n)$ o problemă *PSM*. O problemă a comisvoiajorului asociată problemei *PSM* se poate defini considerând o mulţime $C = (c_1, c_2, ..., c_n, c_{n+1})$ de localităţi şi mulţimea distanţelor:

$$d(c_i, c_j) = \begin{cases} |s_i| - \Psi(s_i, s_j) & 1 \leq i \leq n, 1 \leq j \leq n, i \neq j, \\ |s_i| & 1 \leq i \leq n, j = n+1, \\ \|S\| & i = n+1, 1 \leq j \leq n \end{cases}$$

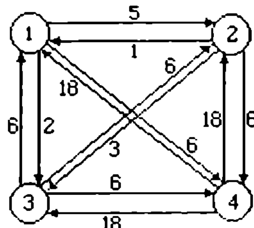


Fig. 2.8.

Exemplul 2.9. Un exemplu ilustrativ este arătat pe figura 2.8., unde $S = \{cbadef, fcbade, adefcd\}$ şi vârfurile 1, 2, 3 reprezintă şirurile *cbadef*, *fcbade*, *adefcd*. Ţinând seama de felul în care au fost definite distanţele $d(c_i, c_j)$, localitatea finală a drumului comisvoiajorului este c_{n+1}. Drumul minimal este (2, 1, 3, 4) şi are lungimea 9. Acestui drum îi cerespunde superşirul *fcabdefadefcd*. ■

2.6. Problema acoperirii unui set de mulţimi

Punerea problemei. Problema acoperirii unui set de mulţimi (*SCP*) constă în acoperirea rândurilor unei matrici zero-unu (0-1), având *m* linii şi *n* coloane, cu o

https://biblioteca-digitala.ro / https://unibuc.ro

submulţime de coloane astfel încât costul acoperirii să fie minimal. Dacă $x_j = 1$, în cazul când coloana j, de cost $c_j > 0$, figurează în soluţie, în caz contrar $x_j = 0$, problema *SCP* se enunţă astfel:

$$\text{să se minimizeze } \sum_{j=1}^{n} c_j x_j, \tag{2.6}$$

cu condiţiile:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, \text{ pentru } i = 1, 2, ..., m, \tag{2.7}$$

$$x_j \in \{0, 1\}, \text{ pentru } j = 1, 2, ..., n. \blacksquare \tag{2.8}$$

Aplicaţii posibile: aprovizionarea cu piese de schimb, obţinerea unor împrumuturi.

2.6.1. Prezentarea algoritmului

Fără a micşora generalitatea, se presupune că anterior coloanele au fost ordonate în ordinea crescătoare a costului, coloanele cu cost egal fiind ordonate în ordinea descrescătoare a numerelor rândurilor pe care le acoperă. Aceasta înseamnă că pentru orice rând i, coloana $\min\{j \mid a_{ij} = 1, j = 1, 2, ..., n\}$ este cea mai "bună" coloană ce acoperă rândul i.

Notând cu t_i (≥ 0), $i = 1, ..., m$, multiplicatorii lui Lagrange asociaţi cu ecuaţiile (2.7), se obţine următoarea problemă (*LLBP*) pentru marginea inferioară a problemei, deci pentru cazul semnului "=" în (2.7):

$$\text{să se minimizeze } \sum_{j=1}^{n} \left(c_j - \sum_{i=1}^{m} t_i a_{ij} \right) x_j + \sum_{i=1}^{m} t_i, \tag{2.9}$$

cu condiţiile:

$$x_j \in \{0, 1\}, \text{ pentru } j = 1, 2, ..., n. \tag{2.10}$$

Notând cu C_j coeficientul lui x_j în (2.9) şi cu X_j valorile soluţiei pentru x_j, rezultă că $X_j = 1$, dacă $C_j \leq 0$ (altfel $X_j = 0$) şi valoarea marginii inferioare Z_{LB} a soluţiei optimale dată de:

$$Z_{LB} = \sum_{j=1}^{n} C_j X_j + \sum_{i=1}^{m} t_i.$$

Pentru maximizarea marginii inferioare obţinute în urma rezolvării problemei *LLBP* se foloseşte o formă particulară a metodei gradientului (*subgradient optimisation*) prezentată în continuare.

Fie Z_{max} valoarea maximă identificată a marginii inferioare, Z_{UB} cea mai bună soluţie găsită la un moment dat şi P_k marginea inferioară în cazul când coloana k, $k = 1, 2, ..., n$, este obligată să aparţină soluţiei. Se poate obţine o soluţie pentru problema *SCP* aplicând algoritmul următor.

https://biblioteca-digitala.ro / https://unibuc.ro

Algoritmul 2.8. (*SCP*)

1. [Inițializări] Se face $Z_{max} = -\infty$, $Z_{UB} = \infty$, $P_k = c_k$, pentru $k = 1, 2, ..., n$, şi $t_i = \min \{c_j \mid a_{ij} = 1, j = 1, 2, ..., n\}$, pentru $i = 1, 2, ..., m$.
2. [Soluție *LLBP*] Se rezolvă problema *LLBP* cu mulțimea curentă a multiplicatorilor t_i, soluția obținută fiind notată cu Z_{LB}, X_j, $j = 1, 2, ..., n$. Se efectuează atribuirea $Z_{max} = \max[Z_{max}, Z_{LB}]$.
3. [Soluție *SCP*] Se construieşte o soluție S a problemei originale *SCP* în modul următor:

   a) Fie $S = \{j \mid X_j = 1, j = 1, 2, ..., n\}$.

   b) Pentru fiecare rând i neacoperit, adică pentru care $\sum_{j=1}^{n} a_{ij} X_j = 0$, se adaugă la S coloana corespunzătoare lui $\min\{j \mid a_{ij} = 1, c_j < \infty, j = 1, 2, ..., n\}$.

   c) Se parcurg coloanele $j \in S$ în ordinea descrescătoare a indicelui j şi, dacă $S - \{j\}$ este o soluție a *SCP*, se efectuează atribuirea $S = S - \{j\}$.

   d) Se înlocuieşte Z_{UB} cu $Z_{UB} = \min\{Z_{UB}, \sum_{j \in S} c_j\}$.

   Pasul b) poate fi efectuat într-un mod eficient, din punct de vedere computațional, observând că, neglijând coloanele eliminate, se poate identifica aceeaşi coloană ce acoperă un rând neacoperit.
4. [Terminare] Dacă $Z_{max} = Z_{UB}$, atunci STOP (Z_{UB} este soluția optimală).
5. [Eliminare coloane] Din examinarea problemei originale *SCP* (ecuațiile (2.6) - (2.8)) şi a problemei *LLBP* (ecuațiile (2.9) şi (2.10)) rezultă că, impunând constrângerea adițională ca o coloană particulară k să figureze în soluția optimală, adică $x_k = 1$, rezultă o margine inferioară egală cu $Z_{LB} + C_k$ dacă $X_k = 0$, altfel Z_{LB} nu se schimbă ($X_k = 1$). Rezultă că valoarea lui P_k poate fi schimbată în conformitate cu:

   $$P_k = \max(P_k, Z_{LB} + C_k), \text{ dacă } X_k = 0, \text{ pentru } k = 1, 2, ..., n$$
   $$P_k = \max(P_k, Z_{LB}), \text{ dacă } X_k = 1, \text{ pentru } k = 1, 2, ..., n.$$

   Deci din problema de rezolvat pot fi eliminate coloane considerând:

   $$c_k = \infty, \text{ dacă } P_k > Z_{UB}, \text{ pentru } k = 1, 2, ..., n,$$

   deoarece reținerea unei coloane k cu P_k mai mare ca Z_{UB} nu poate furniza o soluție îmbunătățită.
6. [Calcul subgradienți] Se calculează subgradienții G_i conform definiției:

   $$G_i = 1 - \sum_{j=1}^{n} a_{ij} X_j, \text{ pentru } i = 1, 2, ..., m.$$

   În cazul când gradienții astfel calculați nu contribuie în mod efectiv la schimbarea valorilor multiplicatorilor, ei sunt ajustați astfel:

   $$G_i = 0, \text{ dacă } t_i = 0 \text{ şi } G_i < 0, \text{ pentru } i = 1, 2, ..., m.$$

https://biblioteca-digitala.ro / https://unibuc.ro

7. [Terminare] Dacă $\sum_{i=1}^{m}(G_i)^2 = 0$, atunci STOP (deoarece în acest caz nu se poate defini o valoare convenabilă a pasului (conform definiţiei (2.11) de mai jos).
8. [Calcul pas] Calculează valoarea pasului în conformitate cu:

$$T = f(1.05\ Z_{UB} - Z_{LB}) / \left(\sum_{i=1}^{m}(G_i)^2\right), \qquad (2.11)$$

unde iniţial $f = 2$. Dacă Z_{max} nu creşte în ultimile 30 de iteraţii ale procedurii calculului subgradientului, plecând de la valoarea curentă a lui f, atunci f se înjumătăţeşte. Expresia lui T din (2.11) are forma standard folosită în metoda subgradientului, cu excepţia factorului 1.05 care s-a dovedit, în urma testărilor pe calculator, util pentru a evita micşorarea prea mare a lui T atunci când valorile Z_{UB} şi Z_{B} devin apropiate.
9. [Terminare] Dacă $f \leq 0.005$, atunci STOP (Criteriu arbitrar de încheiere a calculelor).
10. [Schimbare multiplicatori] Se schimbă valorile multiplicatorilor Lagrange în conformitate cu:

$$t_i = \max(0, t_i + T\,G_i), \text{ pentru } i = 1, 2, ..., m,$$

adică în conformitate cu definiţia standard a modificării multiplicatorilor Lagrange negativi, şi se trece la pasul 2. ■

La încheierea euristicii descrise, Z_{UB} este valoarea celei mai bune soluţii identificate şi Z_{max} este valoarea celei mai bune margini inferioare.

Calitatea soluţiei este apreciată cu criteriul:

$$(Z_{UB} - Z_{max}) / Z_{max}.$$

Structura euristicii Lagrange este:

a) Se identifică o relaxare Lagrange a problemei uşor de rezolvat.
b) Se foloseşte optimizarea subgradientului în vederea maximizării marginii inferioare obţinută în urma relaxării.
c) La fiecare iteraţie a pasului b) se ajustează, dacă este posibil, marginea inferioară curentă într-o soluţie a problemei date.

2.7. Minimizarea sumei maximale pe liniile unei matrice

Punerea problemei. Fiind dată o matrice cu elemente nenegative de dimensiuni $m \times n$ se studiază problema permutării independente a elementelor din fiecare coloană în vederea minimizării sumei maximale pe linii.

https://biblioteca-digitala.ro / https://unibuc.ro

Aplicaţii posibile: planificarea activităţilor.

Această problemă este NP-completă.

2.7.1. Algoritmul *RDI* pentru cazul general

Înainte de a prezenta algoritmul *RDI* se descrie un algoritm exact ce rezolvă cazul particular $m \times 2$, numit *DI*.

Algoritmul 2.9. (*DI*)

1. [Coloana 1] Se aranjază elementele coloanei 1 în ordine crescătoare începând de la rândul 1 la m.
2. [Coloana 2] Se aranjază elementele coloanei 2 în ordine descrescătoare. ■

Complexitatea temporală a algoritmului *DI* este $O(m \log m)$.

Lema 2.1. Algoritmul *DI* este optimal pentru cazul $m \times 2$ al problemei studiate.

Demonstraţie. Să presupunem că există un cel mai scurt contraexemplu. Dacă, într-o soluţie optimală, a_1 este cuplat cu b_m, atunci a_1 şi b_m pot fi eliminaţi, deoarece atât în soluţia optimală cât şi în cea furnizată de algoritmul *DI* a_1 şi b_m aparţin aceluiaşi rând, după rezultat obţinându-se un contraexemplu mai scurt decât cel presupus. Să presupunem atunci că într-o soluţie optimală a_1, b_m sunt cuplaţi cu b', a', adică matricea transformată este de forma:

$$\begin{bmatrix} a_1 & b' \\ \vdots & \vdots \\ a' & b_m \end{bmatrix},$$

cu

$$a_1 + b' \leq \omega_{OPT} \text{ şi } a' + b_m \leq \omega_{OPT}.$$

Deoarece $a' \leq a_1$ şi $b_m \leq b'$, se deduce că:

$$a_1 + b_m \leq a_1 + b' \leq \omega_{OPT} \text{ şi } a' + b' \leq a' + b_m \leq \omega_{OPT}.$$

Rezultă că, interschimbând între ele elementele b' şi b_m în soluţia examinată, se obţine o nouă soluţie optimală în care au loc cuplările (a_1, b_m) şi (a', b'). Eliminând pe a_1 şi b_m se obţine un contraexemplu mai scurt decât cel acceptat iniţial şi prin urmare ipoteza că algoritmul *DI* nu este optimal se respinge. ■

Pe baza algoritmului *DI* se poate obţine un algoritm aproximativ numit *RDI* pentru cazul general $m \times n$, în modul descris în continuare. În cadrul algoritmului se calculează suma finală pe rânduri adunând succesiv câte o coloană.

https://biblioteca-digitala.ro / https://unibuc.ro

Algoritmul 2.10. (*RDI*)

1. [Aplicare *DI*] Se aplică algoritmul *DI* la primele două coloane ale matricei şi se face $j = 2$.
2. [Terminare] Dacă $j = n$, se obţin permutările corespunzătoare elementelor coloanelor parcurgând în sens invers procedurile de generare ale coloanelor şi apoi STOP.
3. [Însumări] Se însumează coloanele $(j - 1)'$ cu j obţinând coloana auxiliară j' (se consideră în loc de coloana 1' coloana 1 în cazul $j = 1$) şi se face $j = j + 1$.
4. [Aplicare *DI*] Se aplică algoritmul *DI* coloanelor $(j - 1)'$ şi j apoi se merge la pasul 2. ■

Algoritmul descris are complexitatea $O(m\ n \log m)$.

Teorema 2.12. $\omega_{RDI} \leq (2 - 1 / m)\ \omega_{OPT}$ pentru matrici $m \times n$.

Demonstraţie. Fie d_j cel mai mare element din coloana j, pentru $j = 1, ..., n$. Fie γ_1 şi γ_2 elementele maxim, respectiv minim, din coloana n' şi $d^* = \max\{d_j\}$.

Aserţiune. $\gamma_1 - \gamma_2 \leq d^*$.

Demonstraţia se efectuează prin inducţie. Mai întâi se consideră parcurgerea coloanelor 1 şi 2. Fie $a + c$, respectiv $b + l$, elementul maxim, respectiv minim, din coloana 2', unde a şi b aparţin coloanei 1 şi c şi l aparţin coloanei 2. Au loc două situaţii:

i) $a > b$. În acest caz, în virtutea algoritmului *DI*, $c < l$ şi prin urmare:

$$(a + c) - (b + l) = (a - b) + (c - l) \leq a - b \leq d_1.$$

ii) $a \leq b$. Atunci:

$$(a + c) - (b + l) = (a - b) + (c - l) \leq c - l \leq d_2.$$

Rezultă că diferenţa dintre $a + c$ şi $b + l$ este mărginită superior de $\max(d_1, d_2)$ care este cel mult d^*. Aplicând succesiv argumentele precedente coloanelor $(k - 1)'$ şi k, rezultă că diferenţa dintre cel mai mare şi cel mai mic element din coloana k' este cel mult $\max(d^*, d_k) = d^*$, rezultat adevărat şi pentru coloana n'. Cu aceasta, demonstraţia aserţiunii este încheiată.

În continuare, se observă că:

$$\omega_{RDI} = \gamma_1,\ \omega_{OPT} \geq \gamma_2 + \frac{1}{m}(\gamma_1 - \gamma_2) \text{ şi } \omega_{OPT} \geq \max(d_1, d_2, ..., d_n) = d^*,$$

de unde rezultă:

$$\omega_{RDI} - \omega_{OPT} \leq \gamma_1 - \gamma_2 + \frac{1}{m}(\gamma_1 - \gamma_2) = \left(1 - \frac{1}{m}\right)(\gamma_1 - \gamma_2) \leq \left(1 - \frac{1}{m}\right)d^* \leq \left(1 - \frac{1}{m}\right)\omega_{OPT}.$$

Prin urmare $\omega_{RDI} \leq (2 - 1 / m)\ \omega_{OPT}$. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Exemplul 2.10. Următorul exemplu ilustrează situația în care marginea superioară este atinsă.

$$A_{\mathrm{OPT}}=\begin{bmatrix} 1 & m-1 & 0 \\ 2 & m-2 & 0 \\ \vdots & \vdots & \vdots \\ m-2 & 2 & 0 \\ m-1 & 1 & 0 \\ 0 & 0 & m \end{bmatrix} \quad A_{\mathrm{RDI}}=\begin{bmatrix} m-1 & 0 & m \\ m-2 & 1 & 0 \\ \vdots & \vdots & \vdots \\ 2 & m-3 & 0 \\ 1 & m-2 & 0 \\ 0 & m-1 & 0 \end{bmatrix}.$$

În soluția optimă maximul sumelor pe linii este m pe când aplicând algoritmul euristic se obține maximul sumei pe linii $2m - 1$. ■

2.7.2. Algoritmul *RLPT* pentru cazul $m \times 3$

În continuare se prezintă un algoritm, care furnizează în cel mai rău caz o soluție mai bună decât cea a algoritmului *RDI*, pentru cazul matricilor $m \times 3$.

Algoritmul 2.11. (***RLPT***)

1. [Ordonare elemente] Se ordonează descrescător toate elementele matricei.
2. [Terminare] Dacă toate elementele au fost plasate, atunci STOP.
3. [Alegere] Se identifică cel mai mare element curent $a_{i^* j^*}$ din lista ordonată.
4. [Plasare] Se atribuie acest element rândului cu cea mai mică sumă, dintre rândurile a căror locuri din coloana j^* nu sunt încă ocupate, prin schimbarea celor două elemente între ele.
5. [Ciclare] Se merge la pasul 2. ■

Complexitatea algoritmului *RLPT* este $O(m \log m)$.

Teorema 2.13. $\omega_{\mathrm{RLPT}} \leq 3/2\ \omega_{\mathrm{OPT}}$ pentru matrici $m \times 3$.

Demonstrație. Presupunând că afirmația nu este adevărată, fie un cel mai scurt contraexemplu, adică unul cu r minim, cu elementele $t_1, t_2, ..., t_r$, considerate în ordinea descrescătoare. Evident că, în virtutea euristicii *RLPT*, elementul t_r trebuie introdus într-un rând a cărui sumă finală este ω_{RLPT}, adică obținută pe baza aplicării algoritmului *RLPT*. În caz contrar, elementul t_r poate fi eliminat, ω_{RLPT} și ω_{OPT} rămân neschimbate, drept rezultat obținându-se un contraexemplu mai scurt. Rezultă că rămâne de demonstrat inconsistența ipotezei existenței unui contraexemplu cu r elemente. Fără a micșora generalitatea, se presupune că t_r este plasat în coloana 3.

Pentru aceasta se consideră soluția euristică după ce au fost plasate elementele $t_1, t_2, ..., t_{r-1}$ și înaintea plasării lui t_r. Fie $i_1, i_2, ..., i_k$ locurile neocupate din coloana 3. Concluzia teoremei rezultă drept consecință a următoarelor două aserțiuni:

https://biblioteca-digitala.ro / https://unibuc.ro

Aserțiunea 1. Locurile $i_1, i_2, ..., i_k$ din coloanele 1 și 2 sunt ocupate.

Demonstrație. În virtutea *RLPT*, cele mai mari m elemente sunt plasate în rânduri distincte. Rezultă că $r > m$ și că pentru o soluție optimală $\omega_{OPT} \geq 2t_r$. Fie d suma minimală pentru rândurile $\{i_1, i_2, ..., i_k\}$. Atunci t_r va trebui plasat într-unul dintre rândurile $i_1, i_2, ..., i_k$ pentru care suma este d. Dacă $d \leq \omega_{OPT}$ atunci:

$$\omega_{RLPT} - \omega_{OPT} = (d + t_r) - \omega_{OPT} \leq t_r, \text{ deci } \frac{\omega_{RLPT}}{\omega_{OPT}} \leq 1 + \frac{t_r}{\omega_{OPT}} \leq 1 + \frac{1}{2} = \frac{3}{2},$$

în contradicție cu ipoteza, privind existența contraexemplului. Prin urmare, $d > \omega_{OPT}$ și, cum $\omega_{OPT} \geq \max_{1 \leq k \leq r} \{t_k\}$, rezultă că $d > \max_{1 \leq k \leq r} \{t_k\}$. Cu aceasta demonstrarea aserțiunii 1 este încheiată.

Deoarece cele mai mari m elemente sunt plasate în m rânduri distincte, rezultă că trebuie să existe exact k dintre aceste elemente în rândurile $i_1, i_2, ..., i_k$. Unul dintre aceste elemente, t_i^*, trebuie să fie plasat într-un rând ce conține măcar un alt element pozitiv în coloana 3 a aranjamentului optimal, deoarece în coloana 3 există numai $k - 1$ locuri neocupate. Prin urmare $t_i^* + t_r \leq \omega_{OPT}$. Fie t_j^* elementul plasat de algoritmul *RLPT* în același rând cu t_i^*. Din $d > \max_{1 \leq k \leq r} \{t_k\}$ rezultă că d, definit ca fiind suma minimală pentru rândurile $\{i_1, i_2, ..., i_k\}$, trebuie să cuprindă măcar un element din coloanele 1 și 2.

Aserțiunea 2. $2t_j^* \leq \omega_{OPT}$.

Demonstrație. Deoarece t_i^* este plasat înaintea lui t_j^* și cele mai mari m elemente sunt plasate în m rânduri distincte, rezultă că $t_j^* \leq t_m$ și $j^* > m$, unde j^* reprezintă indicele lui t_j^*. Prin urmare, $2t_j^* \leq \omega_{OPT}$.

În concluzie:

$$\omega_{RLPT} \leq t_i^* + t_j^* + t_r \leq (t_i^* + t_r) + t_j^* \leq \omega_{OPT} + 1/2\ \omega_{OPT} = 3/2\ \omega_{OPT},$$

în contradicție cu ipoteza că $\{t_1, t_2, ..., t_r\}$ este un contraexemplu. ■

Exemplul 2.11. Un exemplu ilustrativ, parametrizat prin ε, pentru cazul în care marginea superioară de mai sus este atinsă asimptotic este:

$$A_{OPT} = \begin{bmatrix} 0 & 1+\varepsilon & 1-\varepsilon \\ 1 & 1-2\varepsilon & 0 \end{bmatrix} \quad A_{RLPT} = \begin{bmatrix} 0 & 1+\varepsilon & 0 \\ 1 & 1-2\varepsilon & 1-\varepsilon \end{bmatrix}.$$

Raportul $\omega_{RLPT} / \omega_{OPT}$ este egal cu $(3 - 3\varepsilon) / 2$ și tinde către 3/2 dacă ε tinde la zero. ■

https://biblioteca-digitala.ro / https://unibuc.ro

3 PROBLEME ÎN TEORIA GRAFURILOR

3.1. O euristică pentru problema arborelui lui Steiner

Punerea problemei. Fie $G = (N, E)$ un graf conex, neorientat, cu muchii ponderate pozitiv, adică:

$$w(e) > 0, \text{ pentru orice } e \in E.$$

Fiind dată $S \subseteq N$, se cere să se identifice un subgraf $G_1 = (Y, F)$ al lui G astfel încât G_1 să fie conex, $S \subseteq Y$ şi suma $\sum_{e \in F} w(e)$ este minimă. ■

Aplicaţii posibile: turism, transporturi, energetică, telefonie, canalizare.

Această problemă este cunoscută ca *arborele lui Steiner* şi este NP-completă.

În cele ce urmează se descrie o euristică simplă care rezolvă problema cu performanţa:

$$\varphi_{H} / \varphi_{opt} \leq r = 2(k - 1) / k,$$

unde $k = |S|$, φ_{H} este soluţia furnizată de euristica propusă şi φ_{opt} este soluţia optimală.

Lema 3.1. Fie $T = (N, E)$ un arbore ponderat şi $S \subseteq N$ o mulţime de noduri ce satisface condiţia că toate nodurile terminale ale lui T aparţin lui S. Atunci:

1. Pentru orice indexare a nodurilor lui $S = [S_1, S_2, ..., S_k]$ are loc inegalitatea:

$$\varphi_T = \sum_{e \in E} w(e) \leq 1/2[\sum_{i=1}^{k-1} w_T(S_i\, S_{i+1}) + w_T(S_k\, S_1)];$$

2. Există un mod de indexare $S = [S_1, S_2, ..., S_k]$ astfel încât:

$$\varphi_T = \sum_{e \in E} w(e) = 1/2[\sum_{i=1}^{k-1} w_T(S_i\, S_{i+1}) + w_T(S_k\, S_1)],$$

unde $w_T(S_i\, S_{i+1})$ este valoarea lanţului elementar ce conectează nodurile S_i cu S_{i+1} în arborele T.

Demonstraţie. 1. În evaluarea $\sum_{i=1}^{k-1} w_T(S_i\, S_{i+1}) + w_T(S_k\, S_1)$, valoarea $w(e)$ a fiecărei muchii e figurează cel puţin de două ori. Într-adevăr, fie o muchie $e = xy$ ce

https://biblioteca-digitala.ro / https://unibuc.ro

conectează doi subarbori T_1 şi T_2 ai lui T ce rezultă în urma eliminării muchiei e (vezi figura 3.1.).

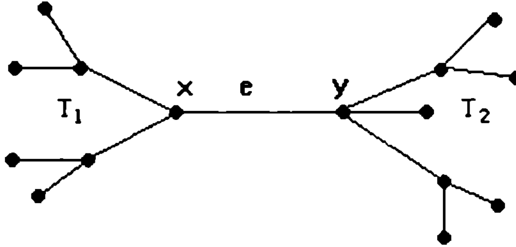


Fig. 3.1.

Pot avea loc două cazuri:

a) S_1 şi S_k aparţin lui T_1. T_2 conţine atunci un S_i cu $1 < i < k$, $S_{i-1} \in T_1$ şi $S_{i+1} \in T_1$ şi ponderea muchiei e apare atât în $w_T(S_{i-1}\, S_i)$ cât şi în $w_T(S_i\, S_{i+1})$;

b) $S_1 \in T_1$ şi $S_2 \in T_2$. Există atunci un i cu $1 \le i \le k$, cu $S_i \in T_1$ şi $S_{i+1} \in T_2$ şi ponderea muchiei e apare o dată în $w_T(S_i\, S_{i+1})$ şi încă o dată în $w_T(S_k\, S_1)$.

2. Fie o indexare a nodurilor lui S care verifică următoarea proprietate: pentru orice muchie e toate nodurile lui S ce aparţin lui $T_1 \cap S$ sunt indexate cu $[S_i, S_{i+1}, \ldots, S_p]$ şi cele aparţimând lui $T_2 \cap S$ sunt indexate cu $[S_1, \ldots, S_{i-1}, S_{p+1}, \ldots, S_k]$. Atunci:

$$\varphi_T = \sum_{e \in T} w(e) = 1/2\left[\sum_{i=1}^{k-1} w_T\,(S_i\, S_{i+1}) + w_T(S_1\, S_k)\right].$$

Pentru o asemenea metodă de indexare, fiecare muchie este numărată exact de două ori şi anume o dată în $w_T(S_{i-1}\, S_i)$ şi o dată în $w_T(S_p\, S_{p+1})$. Această indexare corespunde parcurgerii unui ciclu eulerian ce înconjoară arborele T, fără a intersecta vreo muchie (vezi figura 3.2.). ■

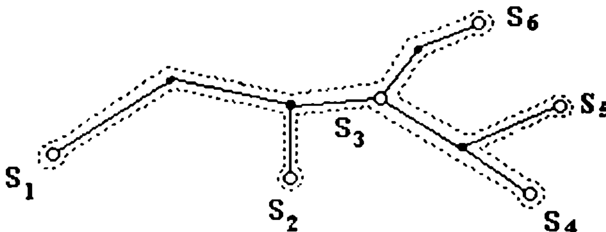


Fig. 3.2.

Pentru rezolvarea problemei studiate se poate utiliza următoarea euristică:

Euristica H.

1. Se construieşte graful complet $G(S) = (S, E_S)$ cu muchii ponderate cu $w_G(S_i\, S_j)$ = = valoarea celui mai scurt lanţ ce leagă S_i cu S_j în graful iniţial ponderat cu w.

https://biblioteca-digitala.ro / https://unibuc.ro

2. Se construieşte un arbore T_S maximal de pondere minimală a grafului $G(S)$.
3. Fie $R = \varnothing$. Pentru fiecare muchie $e_S = S_iS_j \in T_S$ se face $R = R \cup$ {nodul aparţinând celui mai scurt lanţ ce leagă pe S_i cu S_j în graful G}.
4. Se construieşte un arbore maximal de pondere minimală a grafului $Y = S \cup R$.
5. Se elimină nodurile terminale ce nu aparţin lui S. ■

Euristica propusă necesită un timp de $O(n^3)$ pentru calculul matricei simetrice a distanţelor şi două căutări de $O(m \log \log n)$ a arborilor maximali de pondere minimală, unde n este numărul de noduri şi m este numărul de muchii.

Teorema 3.1. $\varphi_H / \varphi_{opt} \leq 2(k - 1) / k$.

Demonstraţie. Fie T_o o soluţie optimală, toate nodurile terminale ale lui T_o aparţinând lui S. Conform lemei 3.1., există un mod de indexare a nodurilor lui S astfel încât:

$$\varphi_{opt} = \varphi_{T_o} = 1/2\ [\sum_{i=1}^{k-1} w_{T_o}(S_i S_{i+1}) + w_{T_o}(S_k S_1)].$$

Deoarece $w_G(S_i S_{i+1}) \leq w_{T_o}(S_i S_{i+1})$ rezultă că:

$$\varphi_{opt} \geq 1/2\ [\sum_{i=1}^{k-1} w_G(S_i S_{i+1}) + w_G(S_k S_1)].$$

Fie ciclul $C = S_1, S_2, ..., S_k, S_1$ în graful $G(S)$; pentru acest ciclu:

$$\varphi_C = \sum_{i=1}^{k-1} w(S_i S_{i+1}) + w(S_k S_1) \text{ şi } \varphi_{opt} \geq 1/2\ \varphi_C.$$

Ciclul C conţine un arbore maximal T_S al grafului $G(S)$. Acest arbore se obţine prin eliminarea din C a unei muchii de pondere maximală. Notând ponderea acestei muchii cu w_{max} rezultă că $\varphi_{T_S} = \varphi_C - w_{max}$, deci:

$$\varphi_{T_S} / (k - 1) - \varphi_C / k = (\varphi_C - w_{max}) / (k - 1) - \varphi_C / k = (\varphi_C / k - w_{max}) / (k - 1) \leq 0,$$

deoarece $\varphi_C / k - w_{max} \leq 0$. În consecinţă:

$$\varphi_{T_S} \leq \varphi_C\ (k - 1) / k.$$

În virtutea euristicii, $\varphi_H \leq \varphi_{T_S}$, deci:

$$\varphi_H \leq \varphi_{T_S} \leq \varphi_C\ (k - 1) / k,$$

de unde rezultă, ţinând seama de relaţia dintre φ_{opt} şi φ_C:

$$\varphi_H / \varphi_{opt} \leq 2(k - 1) / k. \blacksquare$$

Exemplul 3.1. Pentru lanţurile $[S_1,x_2,S_2]$, $[S_2,x_3,S_3]$, respectiv $[S_1,x_1,S_2]$, $[S_2,x_1,S_3]$, ale grafului G din figura 3.3. se obţin soluţiile date în figura 3.4., a doua fiind optimală.

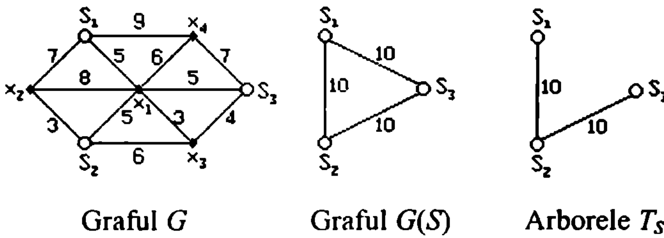
Prima soluţie este extremă, în sensul că se realizează egalitatea:

$$\varphi_H / \varphi_{opt} = 2(k - 1) / k.$$

Într-adevăr:

$$\varphi_H / \varphi_{opt} = 20 / 15 = 4 / 3. \blacksquare$$

https://biblioteca-digitala.ro / https://unibuc.ro



Graful G Graful $G(S)$ Arborele T_S

Fig 3.3.

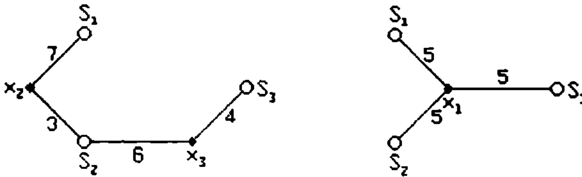
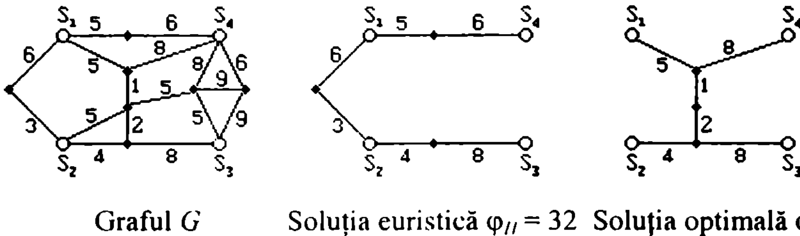


Fig. 3.4.

Exemplul 3.2. În figura 3.5. se dau pentru graful G soluţia euristică şi soluţia optimală. Are loc relaţia: $\varphi_H / \varphi_{opt} = 32 / 28 = 8 / 7 < 2(4 - 1) / 4 = 3 / 2$. ■



Graful G Soluţia euristică $\varphi_H = 32$ Soluţia optimală $\varphi_{opt} = 28$

Fig. 3.5.

Exemplul 3.2. demonstrează faptul că euristica propusă nu furnizează întotdeauna o soluţie optimală.

3.2. Algoritm euristic pentru arbori Steiner rectiliniari

Punerea problemei. Se studiază elaborarea de arbori Steiner, reprezentând conexiunile de pe plăcile circuitelor electronice, care asigură conectarea între ele a n puncte coplanare. Din motive tehnologice şi inginereşti, segmentele unei astfel de conectări sunt orizontale şi verticale. Distanţa dintre două puncte u şi v este:

$$w(u, v) = |x_u - x_v| + |y_u - y_v|. \blacksquare$$

Aplicaţii posibile: electronică, energetică, telefonie.

https://biblioteca-digitala.ro / https://unibuc.ro

Prima abordare a problemei se datorează lui Hanan [HAN66], care a stabilit următorul rezultat. Fie S o mulţime de n terminale ale unor piese electronice situate într-un plan. Cu ajutorul lor se construieşte o reţea externă, în modul ilustrat în figura 3.6., în care terminalele sunt reprezentate prin cercuri.

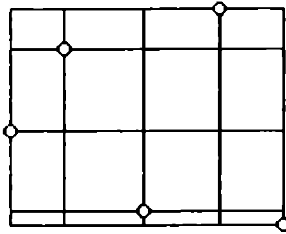


Fig. 3.6.

Fie graful $G_S(V, E)$, în care V este mulţimea punctelor de intersecţie ale liniilor orizontale şi verticale din figura 3.6. şi E este mulţimea segmentelor ce conectează direct două puncte situate pe aceeaşi linie orizontală sau verticală. În lucrarea [HAN66] se demonstrează că un arbore Steiner definit de terminalele date este conţinut în G_S. *Problema Steiner pentru grafuri* (*PSG*) constă în identificarea unui subgraf de pondere minimală ce traversează mulţimea S. În virtutea teoremei lui Hanan [HAN66], rezultă că orice algoritm pentru *PSG* poate fi utilizat pentru identificarea unui *arbore Steiner rectiliniar* (*ASR*).

Algoritmul Hanan începe cu sortarea celor n puncte (terminale) în ordinea definită de coordonatele y. Dacă au fost găsite două puncte cu aceeaşi coordonată y, ele sunt sortate în ordinea definită de coordonata x. Primul subarbore identificat conţine un singur punct, având cea mai mică valoare a coordonatei y. Pe mă..ră ce un nou punct este identificat, el este conectat la subarborele curent printr-o conexiune de lungime minimală. Pentru eliminarea preferinţei alegerii unor puncte datorită direcţiei de parcurgere de jos în sus, distribuţia punctelor este rotită cu 90^0 de trei ori, de fiecare dată fiind construit un nou *ASR* minimal. Urmează alegerea unui cel mai bun dintre cei patru arbori.

Observaţia cheie în implementarea algoritmului constă în aceea că partea de jos a arborelui devine efectiv ascunsă şi poate fi uitată. Un punct a acoperă un punct b dacă b se află în interiorul conului de 90^0 cu vârful în a. Formal, a acoperă pe b dacă $y_b < y_a$, $x_a + y_a \geq x_b + y_b$ şi $x_a - y_a \leq x_b - y_b$.

Lema 3.2. Fie a şi b două puncte oarecare ale arborelui curent atunci când este procesat punctul c. Dacă a acoperă pe b, atunci:

$$w(c, a) \leq w(c, b).$$

https://biblioteca-digitala.ro / https://unibuc.ro

Demonstrație. O demonstrație simplă poate fi construită pe baza analizei cazurilor. De exemplu, dacă $x_c \leq x_b \leq x_a$, atunci $w(c, a) = x_a - x_c + y_c - y_a$ și $w(c, b) = x_b - x_c + y_c - y_b$, deci relația din enunțul lemei este satisfăcută. O demonstrație geometrică a rezultatului poate fi obținută plecând de la observația că un *cerc* de rază $w(c, a)$ în jurul lui c, desenat ca un pătrat în plan, nu conține nici un punct din conul de sub a. Într-adevăr, un cerc de rază $w(c, a)$ este un pătrat cu centrul în c, cu laturi înclinate la 45^0 față de un reper cu axa cx orizontală și cy verticală și având o latură ce trece prin a (vezi figura 3.7.). Coordonatele punctelor de pe latura ce trece prin a verifică relația $x + y = x_a + y_a = w(c, a)$. Pentru orice punct b, din interiorul conului definit de a, există un punct b' pe latura pătratului ce trece prin a pentru care $x_a < x_{b'}$ deci $w(c, a) = w(c, b') < w(c, b)$. ■

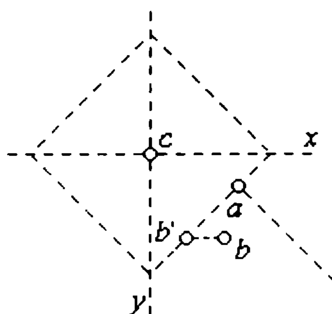


Fig 3.7.

Lema 3.3. Fie a, b și c ca în lema 3.2. astfel încât b nu acoperă pe a. Dacă $x_c \geq x_a \geq x_b$ sau $x_b \geq x_a \geq x_c$, atunci $w(c, a) \leq w(c, b)$.

Demonstrație. Inegalitatea din enunț rezultă prin calcul elementar aplicând definițiile de mai sus. ■

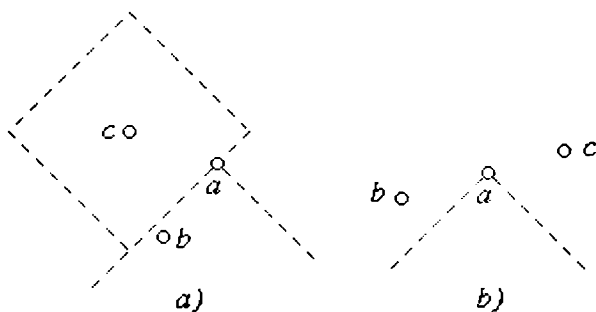


Fig. 3.8.

Lemele 3.2. și 3.3. sunt ilustrate pe figurile 3.8.a), respectiv 3.8.b). O consecință a lemei 3.2. constă în faptul că orice segment orizontal și extremitatea fiecărui segment vertical neacoperit generează un con și porțiunea relevantă a

https://biblioteca-digitala.ro / https://unibuc.ro

arborelui este mulţimea punctelor neacoperite de alte puncte. Această consecinţă este ilustrată pe figura 3.9.

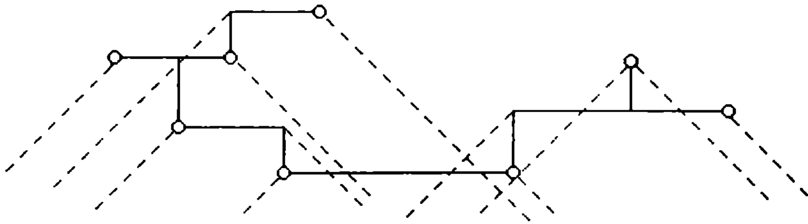


Fig. 3.9.

Dacă porţiunea acoperită a arborelui este eliminată, ceea ce rămâne este o mulţime de segmente orizontale şi puncte singulare. Aceste puncte singulare vor fi interpretate în continuare drept segmente orizontale degenerate. Figura 3.10. ilustrează, cu ajutorul unor linii verticale trasate prin unele puncte terminale, modul în care mulţimea punctelor terminale stângi partiţionează mulţimea coordonatelor x.

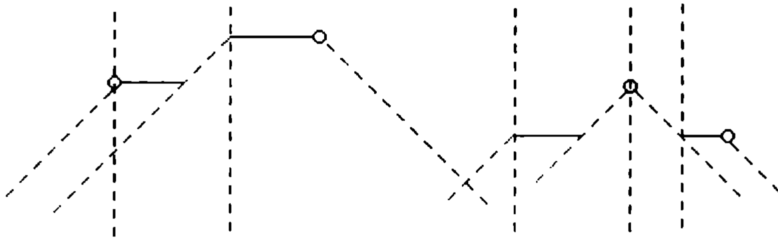


Fig. 3.10.

Algoritmul 3.1. tratează una dintre cele patru parcurgeri. Structura esenţială de date, *SEGMENTS*, este prelucrată de rutinele **pred-succ** şi **addsegment**. Această structură păstrează, pentru arborele curent, mulţimea segmentelor orizontale ce nu sunt acoperite şi segmentele sunt ordonate de către punctele lor terminale stângi. Atunci când un nou punct p este procesat, **pred-succ**(p) returnează cele două segmente, a şi b, adiacente în ordine, astfel încât p este situat în dreapta punctului drept sau deasupra lui a şi p este situat la stânga punctului terminal stâng al lui b. Din lema 3.3. rezultă că singurele puncte ce sunt candidaţi pentru a fi cel mai apropiat punct de p sunt punctele lui a sau punctul terminal stâng al lui b.

Algoritmul 3.1. (***Hanan***)

1. [Iniţializare] Se consideră un arbore vid.
2. [Terminare] Dacă toate punctele au fost incluse în arbore, atunci STOP.
3. [Alegere] Se alege cel mai apropiat punct p dintre punctele neprelucrate.
4. [Vecine] Se determină pentru p segmentele vecine (a, b) = pred-succ(p).
5. [Test apropiere] Dacă p este mai apropiat de b, atunci se merge la pasul 8.

https://biblioteca-digitala.ro / https://unibuc.ro

6. [Poziţionare cu *a*] Dacă *p* este în dreapta lui *a*, atunci se extinde *p* către extremitatea dreaptă a lui *a* (vezi figura 3.11.) şi se conectează cu extremitatea dreaptă a lui *a*; altfel se conectează *p* vertical în jos la *a*.
7. [Continuare] Se merge la pasul 9.
8. [Poziţionare cu *b*] Se extinde *p* către extremitatea stângă a lui *b* şi se conectează *p* cu extremitatea stângă a lui *b*.
9. [Adăugare] Se adaugă segmentul rezultat pentru *p* la arbore cu addsegment.
10. [Ciclare] Se merge la pasul 2. ■

Operaţiile descrise în ciclul algoritmului sunt ilustrate, parţial, pe figura 3.11.

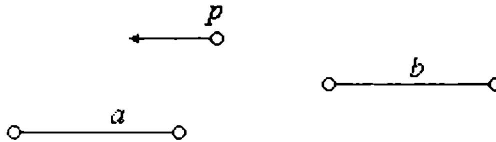


Fig. 3.11.

În funcţie de modul de implementare a algoritmului propus, complexitatea algoritmului este cuprinsă între $O(n^2)$ şi $O(n^{3/2})$.

În lema 3.3. se presupune implicit că punctul *b* poate fi acoperit de *a*, ceea ce în cazul figurii 3.11. revine la a accepta că pot exista mai multe puncte de pe segmentul *a* ce satisfac condiţiile lemei 3.3.

3.3. Algoritm euristic pentru identificarea drumurilor minime

Punerea problemei. Fiind dat un graf orientat şi un nod sursă al său, se cere identificarea celor mai scurte drumuri de la nodul sursă la toate celelalte noduri. ■

Aplicaţii posibile: transporturi, energetică, telefonie.

Algoritmul Bellman-Ford [FOR62] este un algoritm clasic de complexitate temporală $O(n\ m)$, unde n şi m reprezintă numărul de noduri, respectiv arce, ale grafului. Există de asemenea diferite euristici mai performante decât algoritmul Bellman-Ford în multe probleme particulare. O asemenea euristică este prezentată în continuare.

Problema celor mai scurte drumuri este notată (G, s, l), unde $G = (V, E)$ este un graf orientat, $l: E \rightarrow R$ este funcţia lungime şi $s \in V$ este nodul sursă. Soluţia constă în identificarea drumurilor minimale de la s la toate celelalte noduri sau în găsirea unui ciclu al lui G de lungime negativă. Dacă G are un asemenea ciclu se consideră că problema este nerealizabilă. Se presupune că G nu are arce multiple şi $|V|$ şi $|E|$ se notează cu n, respectiv m.

https://biblioteca-digitala.ro / https://unibuc.ro

O *funcţie potenţial* este o funcţie reală definită pe noduri. Fiind dată o funcţie potenţial d, se defineşte *funcţia de cost redus* l_d: $E \to R$ prin:

$$l_d(v, w) = l(v, w) + d(v) - d(w),$$

unde (v, w) este arcul de la v la w. Fiind dată o funcţie potenţial d, se zice că arcul a este *admisibil* dacă $l_d(a) \leq 0$ şi mulţimea arcelor admisibile este notată cu E_d. Graful $G_d = (V, E_d)$ se numeşte *admisibil*.

Arborele drumurilor minimale ale lui G este un arbore minimal având rădăcina s astfel încât, pentru fiecare $v \in V$, drumul de la v la s în arbore este drumul minimal de la s la v.

3.3.1. Metoda corectării etichetelor

În cadrul acestei metode, fiecărui nod v i se asociază potenţialul său $d(v)$, tatăl $\pi(v)$ şi statutul $S(v) \in$ {*neparcurs*, *etichetat*, *examinat*}. Iniţial, pentru fiecare nod v, $d(v) = \infty$, $\pi(v)$ = *nimic* şi $S(v)$ = *neparcurs*. Metoda începe cu $d(s) = 0$ şi $S(v)$ = *etichetat* şi aplică procedura *SCAN* tuturor nodurilor etichetate, până când nu mai există nici unul, caz în care aplicarea metodei se încheie.

Procedura *SCAN* are următoarea formă:

Algoritmul 3.2. (*SCAN*(v))

1. [Ciclare] Pentru orice w astfel încât $(v, w) \in E$ se execută pasul 2.
2. [Prelucrare w] Dacă $d(v) + l(v, w) < d(w)$, atunci se face $d(w) = d(v) + l(v, w)$, $S(w)$ = *etichetat* şi $\pi(w) = v$.
3. [Terminare] Se face $S(v)$ = *examinat* şi STOP. ■

Dacă v este etichetat, atunci $d(v) < \infty$ şi $d(v) + l(v, w)$ este finită.

Metoda se încheie dacă şi numai dacă G nu are cicluri de lungime negativă. În momentul încheierii, etichetele nodurilor *tată* definesc un arbore corect al drumurilor minimale şi, pentru fiecare $v \in V$ $d(v)$, dă cel mai scurt drum de la s la v.

3.3.2. Algoritmul examinării topologice

Fie v şi w etichetate şi $l(v, w) < 0$. Atunci este mai convenabil să se examineze v înaintea lui w deoarece, atunci când v este examinat, $d(w)$ creşte şi w devine etichetat. Algoritmul examinării topologice foloseşte o generalizare a acestei idei.

Pentru simplificarea descrierii, se presupune că G nu are cicluri de lungime mai mică sau egală cu 0 şi, prin urmare, pentru orice d, G_d este aciclic. Mai departe, se va arăta cum se poate renunţa la această presupunere.

Algoritmul menţine mulţimea nodurilor etichetate în două mulţimi, A şi B. Orice nod etichetat aparţine numai uneia din aceste mulţimi. Iniţial $A = \varnothing$ şi $B = \{s\}$.

https://biblioteca-digitala.ro / https://unibuc.ro

La începutul fiecărei parcurgeri, algoritmul foloseşte mulţimea B pentru a determina mulţimea A a nodurilor ce urmează să fie examinate dealungul parcurgerii după care mulţimea B devine vidă. A este o mulţime liniar ordonată. În timpul parcurgerii, elementele sunt extrase conform ordinei din A şi examinate. Noile noduri create se introduc în B. Parcurgerea se încheie când A devine vidă. Algoritmul se încheie când B devine vidă la sfârşitul parcurgerii.

Algoritmul calculează pe A din B după cum urmează.

Algoritmul 3.3. (***Ordonare Topologică***)

1. [Eliminare] Pentru fiecare $v \in B$ din care nu diverge un arc cu cost negativ redus, se elimină v din B şi se marchează ca fiind examinat.
2. [Etichetare] Fie A mulţimea tuturor nodurilor din G_d accesibile din B. Se marchează toate nodurile din A ca fiind etichetate.
3. [Ordonare] Se aplică sortarea topologică pentru a ordona pe A astfel încât pentru fiecare pereche de noduri din A cu $(v, w) \in G_d$, v precede pe w şi prin urmare v va fi examinat înaintea lui w. ■

În cazul, menţionat mai înainte, în care G are cicluri de lungime mai mică sau egală cu 0, este necesar ca graful G_d să nu fie aciclic. Dacă totuşi, G_d are un ciclu de lungime negativă, calculul se poate încheia deoarece se pot obţine drumuri de lungimi oricât de mici. Dacă G_d are cicluri de lungime nulă, aceste cicluri se elimină, prin contractare, şi calculul continuă.

Exemplul 3.3. Pentru graful din figura 3.12., cele mai scurte drumuri de la nodul 1 la nodurile 2, 3, 4 pot fi uşor identificate şi sunt: 1 - 2, 1 - 2 - 3, 1 - 2 - 4 şi 1 - 2 - 3 - 4, având respectiv lungimile 1, 2, 3 şi 3. După cum se vede, pentru nodul 4 există două drumuri minimale, ambele de lungime 3. Modul în care aceste soluţii se obţin cu ajutorul metodelor descrise este prezentat în continuare.

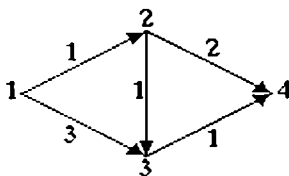


Fig. 3.12.

În cazul aplicării metodei corectării etichetelor, soluţia se obţine succesiv după cum urmează.

Iniţializare: $d(1)=d(2)=d(3)=d(4)=\infty$, $\pi(1)=\pi(2)=\pi(3)=\pi(4)=$ *nimic*, $S(1)=S(2)=$ $=S(3)=S(4)=$ *neparcurs*. Metoda începe cu atribuirile $d(1)=0$, $S(1)=$*etichetat*, urmată de parcurgerea procedurii *SCAN*.

https://biblioteca-digitala.ro / https://unibuc.ro

Arcul (1, 2): $d(1) + 1 < \infty$, $d(2) = d(1) + 1 = 1$, $S(2) =$ *etichetat*, $\pi(2) = 1$.
Arcul (1, 3): $d(1) + 3 < \infty$, $d(3) = d(1) + 3 = 3$, $S(3) =$ *etichetat*, $\pi(3) = 1$.
$S(1) =$ *examinat*.

Noile noduri etichetate sunt 2 şi 3. Continuând cu nodul 2, se obţine:
Arcul (2, 3): $d(2) + 1 = 1 + 1 < 3$, $d(3) = d(2) + 1 = 2$, $S(3) =$ *etichetat*, $\pi(3) = 2$.
Arcul (2, 4): $d(2) + 2 < \infty$, $d(4) = d(2) + 2 = 3$, $S(4) =$ *etichetat*, $\pi(4) = 2$.
$S(2) =$ *examinat*.

Trecând la nodul 3 se obţine:
Arcul (3, 4): $d(3) + 1 = 2 + 1 = 3$, deci procedura *SCAN* nu se mai aplică.
Urmărind calculele precedente, rezultă că:

$$\pi(2) = 1,\ \pi(3) = 2,\ \pi(4) = 2,\ d(2) = 1,\ d(3) = 2 \text{ şi } d(4) = 3.$$

Arborele ce descrie soluţia este arătat în figura 3.13.

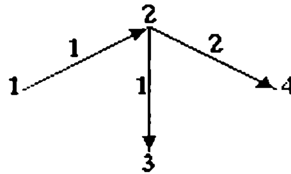


Fig. 3.13.

În soluţia precedentă a fost aplicată, fără a fi menţionată, metoda examinării topologice. Într-adevăr, nodul 2 precede pe 3 şi a fost examinat înaintea lui 3. Dacă se parcurge mai întâi nodul 3, atunci se obţine:
Arcul (3,4): $d(3) + 1 = 3 + 1 = 4 < \infty$, $d(4) = d(3) + 1 = 3$, $S(4) =$ *etichetat*, $\pi(4) = 3$.
$S(3) =$ *examinat*.

Trecând la nodul 2, rezultă:
Arcul (2, 3): $d(2) + 1 = 1 + 1 < 3$, $d(3) = d(2) + 1 = 2$, $S(3) =$ *etichetat*, $\pi(3) = 2$.
Arcul (2, 4): $d(2) + 2 = 3 < 4$, $d(4) = d(2) + 2 = 3$, $S(4) =$ *etichetat*, $\pi(4) = 2$.
$S(2) =$ *examinat*.

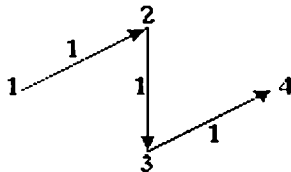


Fig. 3.14.

Urmărind calculele precedente, rezultă că:

$$\pi(2) = 1,\ \pi(3) = 2,\ \pi(4) = 2,\ d(2) = 1,\ d(3) = 2 \text{ şi } d(4) = 3,$$

https://biblioteca-digitala.ro / https://unibuc.ro

adică aceeaşi soluţie cu cea precedentă, dar obţinută parcurgând şi arcul (3, 4).

Se mai poate observa că, în ambele cazuri, nu s-a obţinut şi soluţia din figura 3.14. care se poate obţine dacă, după parcurgerea arcului (2, 3), se parcurge arcul (3, 4), dar această parcurgere nu este conformă algoritmului. ■

Se poate demonstra că algoritmul parcurgerii topologice este corect, are o complexitate temporală $O(n\ m)$ şi că, în multe cazuri, acest algoritm este mai eficient decât algoritmul Bellman-Ford.

3.4. Metodă euristică pentru identificarea clicilor de cardinalitate dată şi de cost minimal

Punerea problemei. Fie dat un graf direcţionat ponderat complet cu mulţimea de noduri $V = \{1, 2, ..., m\}$ şi ponderile arcelor $t_{ij} \geq 0$, $i, j = 1, 2, ..., m$, $i \neq j$, ale arcelor. Se cere ca pentru un n dat, $1 < n < m$, să se identifice o mulţime $C \subset V$ cu $|C| = n$ astfel încât expresia:

$$D(C) = \sum_{i \in C} \sum_{j \in C} t_{ij}$$

să fie de valoare minimă. ■

Aplicaţii posibile: stabilirea unor arondări.

Este uşor de văzut că este suficient să se considere grafuri neorientate având o matrice simetrică a ponderilor:

$$s_{ij} = t_{ij} + t_{ji}, \text{ pentru } i, j = 1, ..., m, i \neq j$$

şi

$$D(C) = \sum_{i \in C} \sum_{\substack{j \in C \\ j > i}} s_{ij}. \tag{3.1}$$

Există un număr de $\binom{m}{n}$ grafuri C care, pentru valori mari ale lui m, sunt greu de enumerat. Pentru $n = 2$ şi $n = m - 1$ soluţiile pot fi găsite uşor. Astfel, pentru $n = 2$:

$$D(C) = s_{pq}, p, q \in C, |C| = 2,$$

astfel încât $C = \{p, q\}$ poate fi identificat căutând un element minimal s_{ij}. Pentru $n = m - 1$ şi $\{k\} = M \setminus C$ rezultă:

$$D(C) = 1/2 \sum_{i \in C} \sum_{j \in C} s_{ij} = 1/2 \sum_{i=1}^{m} \sum_{j=1}^{m} s_{ij} - 1/2 \sum_{j=1}^{m} s_{kj} - 1/2 \sum_{i=1}^{m} s_{ik},$$

astfel încât C poate fi identificat alegând un k pentru care:

$$\min_{C} D(C) = \max_{k \in V} \sum_{i=1}^{m} s_{ik}.$$

https://biblioteca-digitala.ro / https://unibuc.ro

Cei doi algoritmi prezentaţi în continuare constau în îmbunătăţirea succesivă a unei soluţii iniţiale C, aleasă la întâmplare, pe calea schimbării succesive între ele a două noduri $p \in C$ şi $q \notin C$. Pentru (3.1) o asemenea schimbare este:

$$D(C \setminus \{p\} \cup \{q\}) = D(C) - \sum_{\substack{j \in C \\ j \neq p}} s_{pj} + \sum_{\substack{j \in C \\ j \neq q}} s_{qj}.$$

Algoritmul 3.4 (*G1*)

1. [Iniţializări] Se consideră un $C^{(0)}$ iniţial cu n noduri oarecare şi $t = 0$.
2. [Căutare] Se consideră *toate* cele n $(m - n)$ interschimbări posibile (p, q) între noduri din $C^{(t)}$ şi noduri din afara lui $C^{(t)}$.
3. [Terminare] Dacă nu se determină nici o reducere a lui D, atunci STOP.
4. [Modificare] Fie o pereche (p', q'), cu $p' \in C^{(t)}$ şi $q' \notin C^{(t)}$, care asigură o reducere maximală. Se efectuează schimbarea $C^{(t+1)} = C^{(t)} \setminus \{p'\} \cup \{q'\}, t = t + 1$.
5. [Ciclare] Se merge la pasul 2. ■

Algoritmul 3.5. (*G2*)

1. [Iniţializări] Se consideră un $C^{(0)}$ iniţial cu n noduri oarecare şi $t = 0$.
2. [Ordonare] Se ordonează lexicografic toate perechile posibile de interschimbări (p, q) între noduri din $C^{(t)}$ şi noduri din afara lui $C^{(t)}$.
3. [Modificare] Se parcurge lista ordonată la pasul 2 şi, pentru fiecare pereche (p, q) care asigură o reducere, se efectuează schimbarea $C^{(t)} = C^{(t)} \setminus \{p\} \cup \{q\}$ şi se elimină din listă perechile care au p pe primul loc.
4. [Terminare] Dacă nu se determină nici o reducere a lui D, atunci STOP.
5. [Ciclare] Se face $C^{(t+1)} = C^{(t)}$, $t = t + 1$ şi sc trece la pasul 2. ■

Ambii algoritmi se încheie după un număr finit de paşi dar nu garantează obţinerea unei soluţii optimale. Testările experimentale arată că algoritmul *G1* se execută într-un timp mai mare decât algoritmul *G2*.

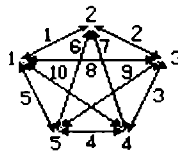


Fig. 3.15.

Exemplul 3.4. Pentru graful din figura 3.15., în care numerele de pe arce reprezintă $t_{ij} + t_{ji}$, numărul de clici cu câte trei noduri este egal cu $\binom{5}{3} = 10$. Aceste clici şi costurile lor sunt date în tabelul 3.1.

https://biblioteca-digitala.ro / https://unibuc.ro

Clica	{1,2,3}	{1,2,4}	{1,2,5}	{1,3,4}	{1,3,5}	{1,4,5}	{2,3,4}	{2,3,5}	{2,4,5}	{3,4,5}
Costul	11	18	12	21	22	19	12	17	17	16

Tabelul 3.1.

Fie $C^{(0)} = \{3, 4, 5\}$ cu $D(C^{(0)}) = 16$. Deoarece $m = 5$ şi $n = 3$, numărul de interschimbări ce urmează a fi efectuate este egal cu 3 (5 - 3) = 6. Aceste schimbări şi costurile respective sunt:

$C^{(1)}=\{1,4,5\}$, $C^{(2)}=\{2,4,5\}$, $C^{(3)}=\{3,1,5\}$, $C^{(4)}=\{3,2,5\}$, $C^{(5)}=\{3,4,1\}$, $C^{(6)}=\{3,4,2\}$;
$D(C^{(1)}) = 19$, $D(C^{(2)}) = 17$, $D(C^{(3)}) = 22$, $D(C^{(4)}) = 17$, $D(C^{(5)}) = 21$, $D(C^{(6)}) = 12$.

Deoarece $D(C^{(6)}) = 12 < 16 = D(C^{(0)})$, reluând aplicarea căutării cu $C^{(0)} = \{3, 4, 2\}$, $D(C^{(0)}) = 12$ rezultă:

$C^{(1)}=\{1,4,2\}$, $C^{(2)}=\{5,4,2\}$, $C^{(3)}=\{3,1,2\}$, $C^{(4)}=\{3,5,2\}$, $C^{(5)}=\{3,4,1\}$, $C^{(6)}=\{3,4,5\}$;
$D(C^{(1)}) = 18$, $D(C^{(2)}) = 17$, $D(C^{(3)}) = 11$, $D(C^{(4)}) = 17$, $D(C^{(5)}) = 21$, $D(C^{(6)}) = 16$.

Se observă că unele dintre clicile noi identificate se întâlnesc şi în aplicarea anterioară. Deoarece $D(C^{(3)})=11<12=D(C^{(0)})$, se reia căutarea cu $C^{(0)}=\{3,1,2\}$, $D(C^{(0)})=11$.

$C^{(1)}=\{4,1,2\}$, $C^{(2)}=\{5,1,2\}$, $C^{(3)}=\{3,4,2\}$, $C^{(4)}=\{3,5,2\}$, $C^{(5)}=\{3,1,4\}$, $C^{(6)}=\{3,1,5\}$;
$D(C^{(1)}) = 18$, $D(C^{(2)}) = 12$, $D(C^{(3)}) = 12$, $D(C^{(4)}) = 17$, $D(C^{(5)}) = 21$, $D(C^{(6)}) = 22$.

Deoarece nu are loc o descreştere a valorii 11 obținută anterior, rezultă că soluția obținută este {3, 1, 2} cu costul 11.

Pentru a aplica algoritmul *G2* se ordonează lexicografic perechile de numere de noduri ale grafului dat:

(1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (3, 5), (4, 5).

Plecând de la $C^{(0)}=\{3,4,5\}$, $D(C^{(0)})=16$, rezultă că prima schimbare ce poate fi efectuată este definită de perechea (1,3) şi se obține $C^{(1)}=\{1,4,5\}$, cu $D(C^{(1)})=19>16$. Următoarea pereche de noduri ce poate fi utilizată este (1,4) drept rezultat obținându-se $C^{(1)}=\{3,1,5\}$, cu $D(C^{(1)})=22>16$. Continuând cu perechea (1,5) se obține $C^{(1)}=\{3,4,1\}$, cu $D(C^{(1)})=21>16$. Urmează perechea (2,3) pentru care se obține $C^{(1)}=\{2,4,5\}$, cu $D(C^{(1)})=17>16$. Urmează perechea (2,4) pentru care $C^{(1)}=\{3,2,5\}$, cu $D(C^{(1)})=17>16$. Apoi pentru perechea (2,5) se obține $C^{(1)}=\{3,4,2\}$, cu $D(C^{(1)})=12<16$. În continuare algoritmul se reia cu $C^{(0)}=\{3,4,2\}$, cu $D(C^{(0)})=12$, continuând cu perechea (3,5). Rezultă $C^{(1)}=\{3,5,2\}$ cu $D(C^{(1)})=17>12$ şi se constată că perechea (4,5) nu este utilizabilă. Deoarece a existat o reducere, se reia parcurgerea perechilor cu $C^{(1)}=\{3,4,2\}$, cu $D(C^{(1)})=12$. Rezultă succesiv:

perechea (1, 2), $C^{(2)} = \{3, 4, 1\}$, cu $D(C^{(2)}) = 21 > 12$,
perechea (1, 3), $C^{(2)} = \{1, 4, 2\}$, cu $D(C^{(2)}) = 18 > 12$,
perechea (1, 4), $C^{(2)} = \{3, 1, 2\}$, cu $D(C^{(2)}) = 11 < 12$,
se face $C^{(1)}=\{3, 1, 2\}$ şi $D(C^{(1)})=11$,

https://biblioteca-digitala.ro / https://unibuc.ro

perechea (2, 5), $C^{(2)} = \{3, 4, 5\}$, cu $D(C^{(2)}) = 16 > 11$,
perechea (3, 5), $C^{(2)} = \{3, 5, 2\}$, cu $D(C^{(2)}) = 17 > 11$,
perechea (4, 5), $C^{(2)} = \{3, 5, 2\}$, cu $D(C^{(2)}) = 17 > 11$.

Urmează parcurgerea algoritmului pentru $C^{(2)} = \{3, 1, 2\}$, cu $D(C^{(2)}) = 11$. În continuare se obţine succesiv:

perechea (1, 4), $C^{(3)} = \{3, 4, 2\}$, cu $D(C^{(3)}) = 12 > 11$,
perechea (2, 5), $C^{(3)} = \{3, 4, 5\}$, cu $D(C^{(3)}) = 16 > 11$,
perechea (3, 4), $C^{(3)} = \{4, 1, 2\}$, cu $D(C^{(3)}) = 18 > 11$,
perechea (3, 5), $C^{(3)} = \{5, 1, 2\}$, cu $D(C^{(3)}) = 12 > 11$.

Deci, soluţia obţinută, minimală, este $C^{(2)} = \{3, 1, 2\}$, cu $D(C^{(2)}) = 11$. ■

După cum se vede, ambii algoritmi necesită, în exemplul considerat, un număr de paşi mai mare decât cel necesar calculului direct al celor 10 clici şi identificarea clicii minimale. Există exemple pentru care algoritmii propuşi sunt mai eficienţi. Se poate limita numărul de paşi făcuţi, obţinându-se o aproximare a soluţiei finale. În exemplul considerat, dacă numărul de paşi este limitat la 6, atunci ambii algoritmi identifică soluţia $C = \{3, 4, 2\}$, cu $D(C) = 12$.

3.5. Localizarea mai multor centre într-un graf

Punerea problemei. 1. Fiind date mai multe puncte de cerere a unor servicii, situate în nodurile unui graf, şi p centre de executare a acestor servicii, se cere aşezarea acestor centre în nodurile sau pe arcele grafului, astfel încât durata de aşteptare sau distanţa parcursă necesară sosirii la cel mai îndepărtat punct solicitant să fie minimală. Această problemă se numeşte *problema p-centrului absolut al unui graf*.

2. O altă problemă, care se poate rezolva în mod asemănător, prin aceeaşi metodă, constă în identificarea celui mai mic număr de centre de deservire şi localizarea lor, pentru o valoare *critică* dată a duratei de aşteptare, respectiv a distanţei parcurse. Aceasta se numeşte *problema p-centrului*. ■

Aplicaţii posibile: servicii, energetică, telefonie.

Notaţiile utilizate în continuare sunt:

G - reprezentarea unui graf,
x - nodurile lui G,
y - puncte oarecare, pe arcele sau nodurile unui graf,
X_p - o mulţime de p noduri ale lui G,
Y_p - o mulţime de orice p puncte ale lui G,
v_j - *ponderea* nodului x_j,
$G(X_n, U)$ - graf în care X_n este mulţimea celor n noduri şi U mulţimea celor m arce.

https://biblioteca-digitala.ro / https://unibuc.ro

În continuare se presupune că graful G este conex şi nedirecţionat. *Matricea distanţelor*, notată cu $D = [d(x_i, x_j)]$, este matricea celor mai scurte distanţe dintre perechile de noduri ale grafului G. Mai precis, cu $d(y_i, y_j)$ se notează distanţa minimă dintre orice pereche de puncte y_i, y_j de pe G.

Fiind dat un nod x_i, operația de parcurgere a tuturor drumurilor divergente din x_i până la o distanţă de x_i egală cu $\delta_i = \lambda / v_i$, unde λ este o constantă dată, se numeşte *penetrare*. Fie $Q_\lambda(x_i)$ mulţimea tuturor punctelor y ale lui G faţă de care distanţa unui nod x_i al lui G este mai mică sau egală cu δ_i, pentru un λ dat, adică:

$$Q_\lambda(x_i) = \{y \mid d(y, x_i) \leq \delta_i\}$$

Mulţimea $Q_\lambda(x_i)$, numită *accesibilă din* x_i, poate fi calculată foarte uşor folosind algoritmul lui Dijkstra pentru identificarea drumurilor minimale. O *regiune* R_λ se defineşte ca fiind mulţimea punctelor pentru care distanţa la aceeaşi mulţime de noduri ale lui G este mai mică sau egală cu δ_i, pentru un λ dat. O regiune poate fi un segment al unui arc, un punct al lui G sau o reuniune de segmente de arce având noduri comune.

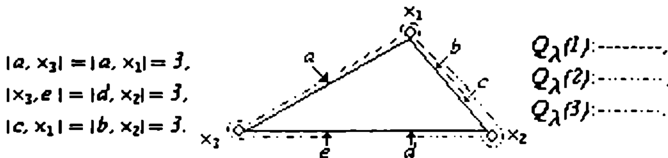


Fig. 3.16.

Exemplul 3.5. Drept exemplificare, fie graful din figura 3.16., în care toate nodurile au ponderea 1 şi lungimile arcelor sunt $c_{1,2}=4$, $c_{2,3}=8$ şi $c_{3,1}=6$ unităţi. Pentru $\lambda=3$ unităţi, rezultă $\delta_i=3$ pentru toţi i şi şase regiuni menționate mai jos. Mulţimile $Q_\lambda(x_i)$ sunt reprezentate grafic în modul indicat în dreapta grafului G. Cele şase regiuni sunt:

Regiunea 1: Punctul a care este punctul unic aflat la distanța 3 de nodurile x_1 şi x_3.

Regiunea 2: Secţiunea dintre b şi c a arcului (x_1, x_2). Orice punct din această regiune este situat la o distanţă mai mică sau egală cu 3 faţă de x_1 şi x_2.

Regiunea 3: Reuniunea secțiunilor (a, x_1) şi (x_1, b). Orice punct din această regiune are acces la nodul x_1 pe o distanță mai mică sau egală cu 3.

Regiunea 4: Reuniunea secţiunilor (a, x_3) şi (x_3, e). Din această regiune se poate ajunge la nodul x_3 parcurgând o distanță mai mică sau egală cu 3.

Regiunea 5: Reuniunea secțiunilor (c, x_2) şi (x_2, d). Din această regiune se poate ajunge la nodul x_2 parcurgând o distanță mai mică sau egală cu 3.

Regiunea 6: Secţiunea (e, d) a arcului (x_2, x_3). Orice punct din această regiune este situat la o distanţă mai mare ca 3 de oricare dintre nodurile x_1, x_2 şi x_3. ■

https://biblioteca-digitala.ro / https://unibuc.ro

În general, o regiune R_λ poate fi calculată cu ajutorul mulţimilor accesibile $Q_\lambda(x_i)$ după cum urmează. Regiunile ce nu au acces la nici un nod sunt definite de:

$$R_\lambda(0) = \{y \mid y \text{ este situat pe } G\} - \bigcup_i Q_\lambda(x_i). \qquad (3.2)$$

Regiunile ce au acces la t noduri dintre $x_{i_1}, x_{i_2}, ..., x_{i_t}$, pentru orice t=1,...,n, sunt date de:

$$R_\lambda(x_{i_1}, x_{i_2}, ..., x_{i_t}) = \bigcap_{q=1}^{t} Q_\lambda(x_{i_q}) - [\bigcap_{q=1}^{t} Q_\lambda(x_{i_q})] \cap [\bigcup_{q=t+1}^{n} Q_\lambda(x_{i_q})]. \qquad (3.3)$$

Algoritmul pentru determinarea p-centrului absolut pentru un p dat este:

Algoritmul 3.6. (*Centru absolut*)

1. [Iniţializare] Se face $\lambda = 0$.
2. [Creştere] Se face $\lambda = \lambda + \Delta\lambda$, cu un $\Delta\lambda$ mic.
3. [Partiţionare] Se determină mulţimile $Q_\lambda(x_i)$, pentru toţi $x_i \in X_n$, şi se calculează regiunile R_λ.
4. [Graf asociat] Se construieşte graful $G' = (X', X_n, U')$, unde X' este mulţimea nodurilor reprezentând regiuni şi U' este mulţimea arcelor ce unesc un nod x_i cu un nod regiune dacă şi numai dacă x_i este accesibil din această regiune.
5. [Acoperire] Se identifică acoperirea minimală a grafului G'.
6. [Ciclare] Dacă numărul de regiuni din X' este mai mare decât p, atunci se trece la pasul 2; altfel STOP. În acest din urmă caz, regiunile din X' definesc p-centrul absolut al grafului G. ■

În cazul când se cere determinarea celei mai mici valori p astfel încât fiecare nod să fie accesibil dintr-un anumit centru parcurgând o distanţă critică dată (problema 2. menţionată la început), paşii 3 până la 6 ai algoritmului descris trebuie parcurşi o singură dată pentru o valoare critică dată a lui λ. Numărul corespunzător al regiunilor din acoperirea minimală a lui G' (pasul 5) reprezintă valoarea cerută a p-centrului.

Aspecte computaţionale. Pentru un λ dat, fie $\delta_i = \lambda / v_i$ distanţele asociate fiecărui nod i al grafului G. Un arc oarecare poate fi accesibil din nodul i în întregime, parţial, sau deloc. Dacă numai o parte a arcului este accesibilă, atunci punctului ei limită i se asociază un *marcaj*. Mulţimea acestor marcaje conţine informaţia necesară pentru determinarea mulţimilor $Q_\lambda(x_i)$ şi anume $Q_\lambda(x_i)$ cuprinde toate arcele sau părţile lor de pe drumurile minimale dintre orice marcaj şi nodul x_i. După ce toate marcajele au fost plasate, fiecare arc rezultă divizat în mai multe secţiuni, fiecare secţiune fiind definită de nodurile la care punctele ei au acces. Rezultă că, o secţiune oarecare poate fi descrisă cu ajutorul unui vector cu valori binare $\{j_1, j_2, ..., j_n\}$, în care $j_k = 1$ dacă secţiunea considerată are acces la nodul k şi $j_k = 0$ în caz contrar.

https://biblioteca-digitala.ro / https://unibuc.ro

Totalitatea secţiunilor cărora le este asociat acelaşi vector definesc o aceeaşi regiune R_λ ce poate fi identificată cu ajutorul ecuaţiilor (3.2) şi (3.3). În continuare, vectorii ce definesc o regiune vor fi notaţi cu *SI*.

Dacă mulţimea de valori 1 a unei secţiuni $(SI)_1$ este inclusă în mulţimea de valori 1 a unei secţiuni $(SI)_2$, atunci punctele ei au acces la toate nodurile din $(SI)_2$ şi se zice că $(SI)_1$ *domină* pe $(SI)_2$. Mai precis, $(SI)_1$ domină pe $(SI)_2$ dacă $(SI)_1 \otimes (SI)_2 = (SI)_1$, unde $\otimes$ este produsul boolean al vectorilor $(SI)_1$ şi $(SI)_2$.

Teorema 3.2. Pentru un λ dat, acoperirea minimală G' poate fi identificată prin excluderea din mulţimea X' a nodurilor ce corespund la *SI*-uri dominate de altele.

Demonstraţia este imediată şi rezultă din definiţia relaţiei de dominare. ■

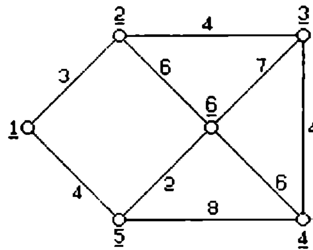


Fig. 3.17.

Exemplul 3.6. Fie graful din figura 3.17. în care ponderile nodurilor sunt egale cu 1 şi numerele de lângă arce reprezintă lungimile lor. Se cere identificarea unui *p*-centru absolut pentru cazul când $\lambda = 3.5$.

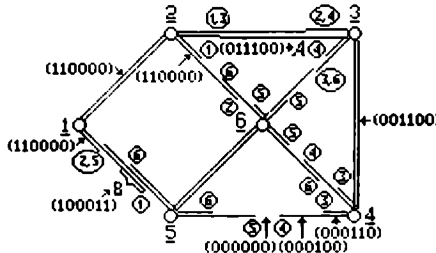


Fig. 3.18.

Marcajele asociate extremităţilor mulţimilor $Q_\lambda(i)$ pot fi identificate urmărind figura 3.17. şi sunt încercuite pe figura 3.18., în care mulţimile $Q_\lambda(i)$ corespund la subgrafuri conexe, unele dintre arcele lor fiind suprapuse. Punctele în dreptul cărora sunt încercuite două cifre corespund la segmente punctuale, situate la distanţa λ de fiecare dintre vîrfurile încercuite. Pe figura 3.18. sunt, de asemenea, indicaţi, pentru exemplificare, şi câţiva vectori *SI*. Perechea (000100), (000110) din stânga nodului 4 ilustrează relaţia de dominare şi anume faptul că segmentul definit de (000110) este

https://biblioteca-digitala.ro / https://unibuc.ro

inclus în segmentul (000100). Numărul total de secțiuni, inclusiv cele punctuale, este egal cu 33. Unele dintre aceste secțiuni, de exemplu cele două din jurul nodurilor 1 și 2, sunt identice și pot fi unificate în vederea obținerii unei singure regiuni. Efectuând toate unificările rezultă următoarele 18 regiuni:

(1)	000000	(7)	000011	(13)	000101
(2)	010000	(8)	110000	(14)	001001
(3)	001000	(9)	001100	(15)	111000
(4)	000100	(10)	100010	(16)	011100
(5)	000010	(11)	011000	(17)	110010
(6)	000001	(12)	010001	(18)	100011.

Utilizând relația de dominare, mai rămân 7 regiuni și anume cele numerotate mai sus de la 12 la 18. Se recunoaște ușor că mulțimea minimală de regiuni ce asigură accesul la toate nodurile grafului G este formată din regiunile:

$$A = (011100) \text{ și } B = (100011),$$

unde A este o regiune punctuală.

Identificarea regiunilor A și B poate fi efectuată și formal în modul următor. Notând cele 7 regiuni cu literele a, b, c, d, e, f, g se construiește forma conjunctivă:

$$(d+f+g)\,(a+d+e+f)\,(c+d+e)\,(b+e)\,(f+g)\,(a+b+c+g),$$

unde fiecare paranteză conține regiunile ce au acces la același nod al grafului G și parantezele succesive corespund nodurilor 1, ..., 6 ale lui G. O literă oarecare, de exemplu a, este interpretată ca reprezentând o regiune ce are acces la nodul asociat perechii de paranteze în care figurează. Un produs de forma fa, obținut prin desfacerea parantezelor, este interpretat ca reprezentând afirmația: nodurile lui G ce corespund regiunilor f și a sunt accesibile din puncte aparținând reuniunii regiunilor f și a. Desfăcând parantezele și ținând sema de regulile:

$$x\,x = x \text{ și } x\,y + x = x,$$

ce au o semnificație evidentă, se identifică cel mai scurt produs de litere reprezentând regiuni asociate nodurilor lui G. Pentru exemplul studiat se obțin succesiv:

$$(d+f+g\,a+g\,e)\,(e+c\,b+d\,b)\,(g+f\,a+f\,b+f\,c) =$$
$$= (d\,e+d\,c\,b+d\,b+f\,e+f\,c\,b+f\,d\,b+g\,a\,e+g\,a\,c\,b+$$
$$+g\,a\,d\,b+g\,e+g\,e\,c\,b+g\,e\,d\,b)\,(g+f\,a+f\,b+f\,c) =$$
$$= (d\,e+d\,b+f\,e+f\,c\,b+g\,a\,c\,b+g\,e)\,(g+f\,a+f\,b+f\,c) =$$
$$= g\,e + \text{termeni ce conțin mai mult de două litere.}$$

Deci regiunile notate cu e și g, adică A și B de mai înainte constituie soluția problemei.

Graful G' corespunzător exemplului studiat este arătat pe figura 3.19. egiunile din acoperirea minimală sunt reprezentate de nodurile 16 și 18. ■

https://biblioteca-digitala.ro / https://unibuc.ro

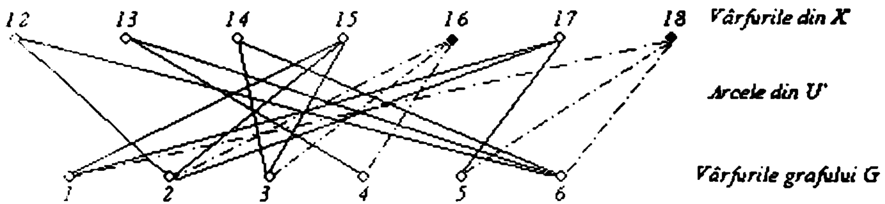


Fig. 3.19.

Rezultatele experimentale confirmă eficiența algoritmului prezentat. Se pot menționa cel puțin două avantaje ale metodei propuse şi anume:

1. Algoritmul se poate încheia atunci când o anumită aproximare a poziției centrelor este acceptabilă şi
2. Metoda poate fi uşor modificată în vederea generării de soluții suboptimale.

3.6. Descompunerea unui graf

Punerea problemei. Fiind dat un graf de complexitate mare, se caută reducerea lui la un graf redus ale cărui noduri reprezintă componente suficient de conexe ale grafului dat, după care problema inițială se reformulează pentru graful redus. ■

Aplicații posibile: rețele de telecomunicații, calcul distribuit şi calcul paralel.

3.6.1. Identificarea clicilor (θ, α) maximale

Definiția 3.1. Fie un graf nedirecționat $G = (X, E)$, unde X este mulțimea nodurilor şi E este mulțimea muchiilor. Fie apoi $\alpha \in [0,1]$ un număr real. O parte $A \subset X$ se numeşte *α - clică* dacă şi numai dacă:

$$t(A) = 2\lambda / (n\ (n - 1)) = \alpha,$$

unde λ este numărul de muchii din A şi n este cardinalul lui A. ■

Ținând sema că un graf cu n noduri are cel mult $n\ (n - 1) / 2$ muchii, rezultă că $\alpha \leq 1$, egalitatea având loc dacă şi numai dacă A este clică.

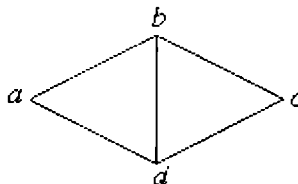


Fig. 3.20.

Exemplul 3.7. Pentru graful din figura 3.20 şi $A = X$, $\alpha = 2{\cdot}5 / (4{\cdot}3) = 0.833$, deci $A = \{a, b, c, d\}$ este o 0,833-clică. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Funcţia $t(A)$ va fi numită în continuare *grad de prezenţă a arcelor.*

Fie un subgraf A, x un nod ce nu aparţine lui A şi $m(x, A)$ numărul de muchii din G de la x la noduri din A. Dacă n este numărul de noduri din A şi numărul de arce din subgraful corespunzător este λ, atunci:

$$t(A \cup \{x\}) = 2\,(\lambda + m(x, A)) \,/\, (n\,(n + 1)).$$

Dacă se cere ca $t(A \cup \{x\}) \geq \beta$, atunci trebuie ca:

$$2\lambda + 2m(x, A) \geq \beta n^2 + \beta n. \tag{3.4}$$

Presupunând că $t(A) \geq \beta$, rezultă:

$$2\lambda \geq \beta n^2 - \beta n. \tag{3.5}$$

Din (3.4) şi (3.5) se obţine condiţia suficientă:

$$m(x, A) \geq \beta n,$$

care permite identificarea recursivă de mulţimi A care sunt β - clici maximale.

Dacă se cere un subgraf cu cel mult θ noduri, atunci se consideră un test suplimentar pentru numărul de noduri.

În continuare se prezintă o generalizare a definiţiilor precedente, deoarece în cadrul reducerilor succesive se efectuează suprapuneri de noduri, eliminarea unor muchii şi suprapunerea unora dintre ele. Din acest motiv, nodurile şi muchiile grafului dat şi ale celui redus sunt ponderate pentru a conserva, în graful redus, un maximum de informaţie privind graful dat. Fie:

$$G(X, E, f, g), f\colon X \to N, g\colon E \to N$$

graful dat, unde f şi g sunt funcţiile pondere ale nodurilor şi muchiilor. Funcţia $t(A)$ are definiţia:

$$t(A) = 2 \sum_{\substack{[x,y] \in E \\ x,y \in A}} g([x,y]) \Big/ \left(\left(\sum_{x \in A} f(x) \right) \left(\sum_{x \in A} f(x) - 1 \right) \right).$$

Pentru simplificarea scrierii se foloseşte notaţia:

$$\sum_{x \in A} f(x) = p(A) \text{ (ponderea submulţimii } A\text{)}.$$

De asemenea, se introduce un criteriu numit *legătură* definit prin:

$$l(A) = \sum_{\substack{[x,y] \in E \\ x \in A, y \in X - A}} g([x,y])$$

şi care permite identificarea dintre părţile unui graf ce au acelaşi grad de prezenţă a aceleia care este cel mai puţin legată de restul grafului dat.

În continuare se prezintă procedura de identificare a submulţimilor grafului dat G, numite (α, θ)-clici maximale cu legături minimale. Această procedură se notează $ALPHA(G, f, g, \theta, \alpha, E)$ sau mai simplu $ALPHA(\alpha, E)$.

(G, f, g) este graful ponderat.

https://biblioteca-digitala.ro / https://unibuc.ro

Valoarea $\alpha \in [0, 1]$ este aleasă a priori.

A este cea mai bună (α, θ)-clică maximală cu criteriul l astfel încât $p(A) \neq 0$.

Pentru mulțimile A ce nu sunt muchii $p(A) > 2$. Într-adevăr, dacă $A = \{a, b\}$, $[a, b] \in E$ și $f(a) = f(b) = 1$, $g([a, b]) = 1$, atunci:

$$t(A) = 2 \cdot 1 / (2 \cdot (2 - 1)) = 1.$$

Rezultă că, în acest caz, A este o 1-clică. Aceste mulțimi nefiind semnificative, se vor elimina în pasul 5 al algoritmului.

Algoritmul 3.7. (***ALPHA***)

1. [Inițializări] Se face $A_0 = \varnothing$, $l_0 = +\infty$ și toate muchiile din E sunt inițial nemarcate.
2. [Terminare] Dacă nu există $[a, b] \in E$ nemarcată, atunci STOP.
3. [Marcare] Se alege o muchie $[a, b] \in E$ nemarcată, se face $A = \{a, b\}$ și se marchează $[a, b]$.
4. [Test clică] Dacă $t(A) < \alpha$ sau $p(A) > 0$, se trece la pasul 2.
5. [Test continuare] Dacă nu există x cu $p(A \cup x) \leq \theta$ și $t(A \cup x) \geq \alpha$, se trece la pasul 8.
6. [Alegere] Se alege x_0 astfel încât: $p(A \cup x_0) \leq \theta$ și pentru orice x pentru care $p(A \cup x) \leq \theta$ să fie adevărată inegalitatea $t(A \cup x) \leq t(A \cup x_0)$.
7. [Adăugare nod la clică] Se face $A = A \cup x_0$ și se trece la pasul 5.
8. [Redefinire] Dacă $p(A) \leq 2$ și $l(A) < l_0$, se face $l_0 = l(A)$ și $A_0 = A$.
9. [Ciclare] Se trece la pasul 2. ■

3.6.2. Procedura contractării grafului

În contractarea unui graf se aplică următoarele trei principii:

1. Graful G cu n noduri și m muchii se transformă într-un graf G' având n' noduri și m' muchii astfel încât: $n > n'$ și $m > m'$.
2. Se păstrează invariantă structura (α, θ)-clicilor între G și G', mai precis, dacă ϕ este aplicația $P(X) \to P(X')$ indusă de aplicația $\pi : X \to X'$ de contracție, definită mai departe, atunci se cere ca:
$$t'[\phi(A)] = t(A) \text{ și } p'[\phi(A)] = p(A).$$
3. Fiind date două părți disjuncte A și B ale grafului G, ordinea în care ele sunt folosite pentru a contracta graful G este arbitrară. Aplicațiile p și t ale lui $P(X)$ în R^+ sunt construite plecând de la aplicațiile f și g ale lui X, respectiv E, în N. Pentru construirea grafului este necesară conservarea informației conținute în f și g.

În continuare se folosesc notațiile: $G=(X,E,f,g)$, A este o parte conexă a lui X, $G'=(X',E',f',g')=G/A$ este contracția grafului G după A, $X'=(X-A)\cup\{a\}$, cu $a \notin X$, $E'=\{E \cap P_2(X-A)\} \cup E''$, cu $P_2(V)=\{W \subseteq V \mid 1 \leq |W| \leq 2\}$, cu $E''=\{[x,a] \mid x \in X; \exists u \in A; [x,u] \in E\}$.

https://biblioteca-digitala.ro / https://unibuc.ro

Funcţiile f' şi g' au definiţiile:

$$f'(x)=\begin{cases} f(x), & x\in X-A \\ p(A), & x=a \end{cases}$$

$$g'([x,y])=\begin{cases} g([x,y]), & (x,y)\in(X-A)\times(X-A) \\ \sum\limits_{u\in A} g([x,u]), & \text{dacă } y=a \text{ şi } x\neq a \\ \sum\limits_{u\in A, v\in A} g([u,v]), & \text{dacă } x=y=a. \end{cases}$$

Figura 3.21. ilustrează modul de efectuare al contractării unui graf.

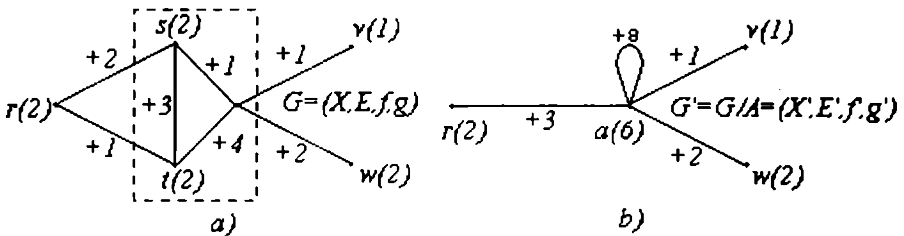


Fig. 3.21.

Algoritmul 3.8. (***Alfa-clică***)

1. [Iniţializare] Se face $\alpha = 1$.
2. [Construire clică] Se aplică algoritmul 3.7. *ALPHA*(α; *E*).
3. [Test] Dacă nu există o *A* (α, θ)-clică nevidă, se trece la pasul 7.
4. [Contractare] Se contractează graful *G* în *G* / *A*.
5. [Ciclare] Dacă problema rezultată nu are un grad de simplitate suficient, se trece la pasul 2.
6. [Terminare] Se identifică o soluţie a problemei reduse şi STOP.
7. [Schimbare α] Se face $\alpha = \alpha - \Delta\alpha$.
8. [Ciclare] Dacă $\alpha > \alpha_{min}$, se trece la pasul 2.
9. [Diagnostic negativ] Se trece la pasul 6. ■

Observaţii. 1. Datele la intrare sunt α_{min} şi $\Delta\alpha$ (decrementul lui α).

2. Drept grad de simplitate se poate alege, de exemplu, numărul de muchii.

3. În cazul unui diagnostic negativ (pasul 9) se trece totuşi la căutarea unei soluţii, deoarece această soluţie s-ar putea dovedi acceptabilă.

Exemplul 3.8. Un exemplu ilustrativ al procedurii de contractare este arătat în figura 3.22. Nodurile şi muchiile grafului *G* au ponderea 1 şi $\alpha_{min} = 0.5$.

Nodurile grafului *G*' se obţin prin contopirea nodurilor încadrate în contururi punctate, notate cu *A*, *B*, ..., *J* şi au ponderile notate lângă aceste contururi. Muchiile grafului *G*' au ponderi egale cu 1, în afară de muchia ce conectează nodurile corespunzătoare contururilor *A* şi *B*, care au ponderea 2. ■

https://biblioteca-digitala.ro / https://unibuc.ro

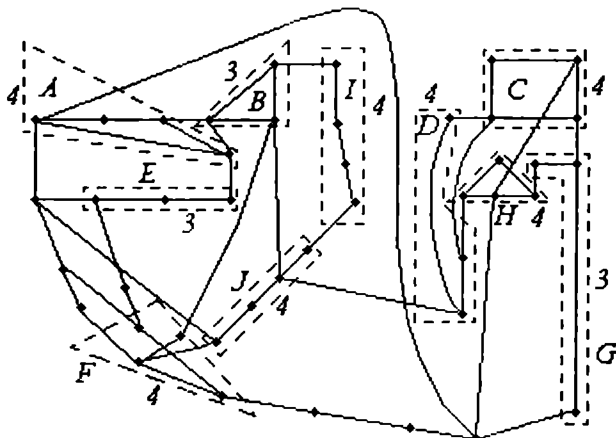


Fig. 3.22.

Graful G' obţinut în modul descris în cele două euristici poate fi utilizat pentru rezolvarea unei probleme de optimizare ce furnizează o soluţie aproximativă pentru o problemă iniţială, reprezentată prin graful G.

3.7. Aşezarea facilităţilor într-o reţea planară

Punerea problemei. Se cere plasarea unor facilităţi în plan în vederea elaborării unor sisteme cât mai eficiente. În termeni de teoria grafurilor, problema se poate formula astfel: Fiind dat un graf nedirecţionat şi ponderat, $G = (V, E)$, să se identifice un subgraf planar al lui, $G_1 = (V, E_1)$, unde $E_1 \subseteq E$, având o pondere totală maximă a muchiilor. ■

Aplicaţii posibile: servicii, telefonie.

3.7.1. Euristica triunghiului

În cadrul euristicii triunghiului se încearcă identificarea unor subgrafuri planare convenabile pe calea construirii succesive a unei triangulaţii. Algoritmul începe cu identificarea primelor patru noduri aplicând procedura *SELECT* sau *AVARICE*.

Algoritmul 3.9. (*SELECT*)

1. [Ponderi] Se calculează pentru fiecare nod i suma: $M(i) = \sum_{j \in V} w_{ij}$, unde w_{ij} este ponderea muchiei ce uneşte nodurile v_i şi v_j.
2. [Ordonare] Se ordonează nodurile în ordinea nedescrescătoare a valorilor $M(i)$.
3. [Selectare] Se selectează primele patru noduri. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Algoritmul 3.10. (*AVARICE*)

1. [Muchie inițială] Se identifică o muchie *ab* de pondere maximă.
2. [Triunghi inițial] Se caută un nod *c* astfel încât triunghiul {*a*, *b*, *c*} să fie de pondere maximă.
3. [Tetraedrul inițial] Se caută un nod *d* astfel încât graful complet {*a*, *b*, *c*, *d*} să fie de pondere maximă. Fie K_4 graful complet definit de aceste noduri (*tetraedru*). Graful are muchiile *ab*, *ac*, *ad*, *bc*, *bd* și *cd* și triunghiurile *abc*, *abd*, *acd* și *bcd*.
4. [Adăugare noduri] Se adaugă nodurile rămase, succesiv câte unul în ordinea definită de algoritmul 3.9., identificând de fiecare dată triunghiul în care se inserează nodul. Fie de inserat nodul *u*. Drept triunghi de inserare se alege triunghiul *abc* pentru care suma $w_{au} + w_{bu} + w_{cu}$ are valoarea maximă. Procesul de adăugare (vezi figura 3.23.) poate fi formalizat în felul următor:

$V = V \cup \{u\}$, $E = E \cup \{au, bu, cu\}$ și $T = T \cup \{abu, acu, bcu\} \setminus \{abc\}$. ■

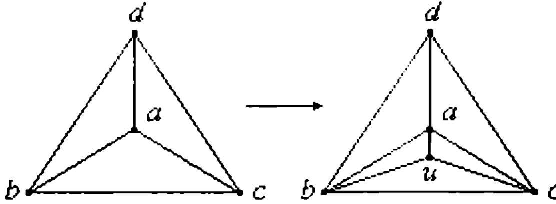


Fig. 3.23.

Procedeul de inserare generează succesiv grafuri planare așa încât nu este necesară testarea planarității la aplicarea algoritmului 3.10.

3.7.2. Euristica expandării roților

Această euristică începe, ca și în cazul precedent, de la un tetraedru furnizat de *SELECT* sau *AVARICE*, aplicând succesiv operația de *desbinare* a nodurilor, care constă în înlocuirea unui nod *v* de grad cel puțin patru cu două noduri *v*' și *v*", graful obținut fiind o triangulație.

O *roată cu n noduri* este un graf ce conține un ciclu de *n* - 1 noduri (*obadă*), fiecare nod al lui fiind adiacent cu un același nod (*butuc*). Desbinarea nodului *butuc*, ilustrată pe figura 3.24, crează, intuitiv, două roți suprapuse.

Procedura expandării începe cu generarea unui tetraedru inițial *H*, utilizând *SELECT* sau *AVARICE*. La fiecare pas următor se adaugă la *H* un nod în modul următor. Se identifică un nod *x* în *H* și un nod *y* în *G* \ *H*. Se caută două noduri *k* și *l* pe obada roții w_x, având ca butuc pe *x*. Se expandează w_x în vederea formării roții w'_x și a unei noi roți w_y, astfel încât *k* și *l* să fie situate pe obada ambelor roți. Procedura este ilustrată pe figura 3.25.

https://biblioteca-digitala.ro / https://unibuc.ro

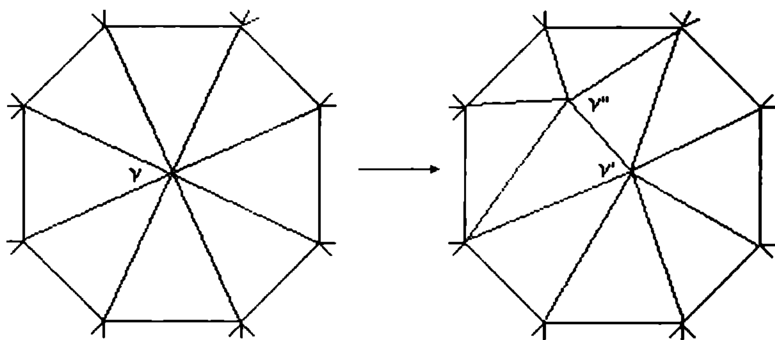


Fig. 3.24.

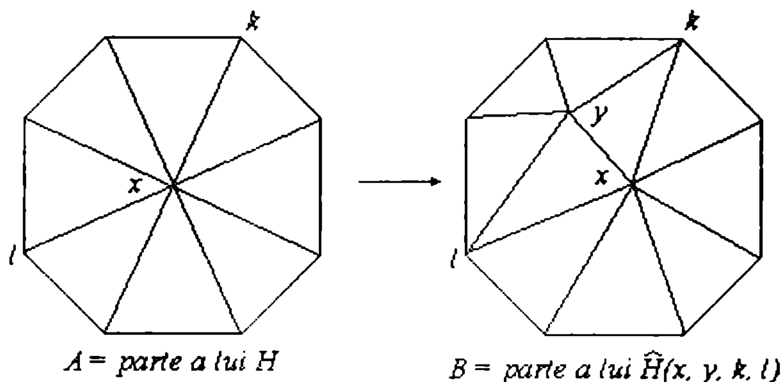


Fig. 3.25.

Graful $\hat{H}(x,y,k,l)$ obţinut din H, depinde de alegerea nodurilor x, y, k, l. Pentru obţinerea lui A se poate utiliza o strategie *Greedy* care constă în alegerea nodurilor x, y, k şi l astfel încât $\hat{H}(x,y,k,l)$ să fie cât mai greu.

3.7.3. Euristica Greedy

În cadrul acestei euristici, se pleacă de la un graf cu n noduri, fără muchii. În continuare se ordonează muchiile necrescător în raport cu ponderile, apoi se identifică succesiv muchiile ce păstrează planaritatea grafului. Procedura se încheie după ce au fost selectate $3n$ - 6 arce. Testul de planaritate reprezintă efortul computaţional major, chiar dacă testul lui Hopcroft şi Tarjan [HOP74] are complexitate liniară în timp.

3.7.4. Îmbunătăţirea soluţiilor finale

Fiind obţinută o soluţie cu una dintre euristicile propuse, ea poate fi îmbunătăţită în următoarele două moduri.

https://biblioteca-digitala.ro / https://unibuc.ro

1. **Înlocuirea muchiilor.** Eliminarea unei muchii *ab* dintr-o triangulație crează un patrulater, de exemplu *acbd*. Dacă a doua diagonală *cd* a patrulaterului nu aparține soluției, *ab* se poate înlocui cu *cd*, soluția fiind mai bună dacă $w_{ab} < w_{cd}$.
2. **Relocarea nodurilor.** Un nod *a* de gradul 3 și arcele incidente cu el *ae*, *af* și *ag* poate fi relocat într-un nou triunghi al triangulației, *bcd*, introducând arcele *ab*, *ac* și *bd*, în modul descris în figura 3.26.

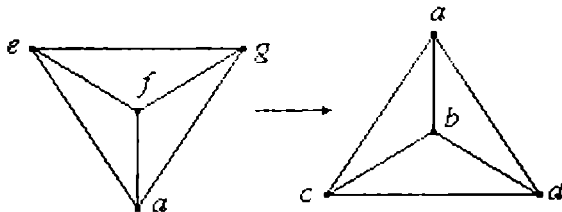


Fig. 3.26.

Noua triangulație este mai bună dacă:

$$w_{ae} + w_{af} + w_{ag} < w_{ab} + w_{ac} + w_{ad}.$$

3.7.5. Complexitatea euristicilor propuse

În continuare, cele trei euristici sunt notate cu *DELTA*, *W-E* și *GREEDY*.

Pentru euristica *DELTA* se observă că, la fiecare etapă *i* a inserării unui nod, există O(*i*) triunghiuri candidate de luat în considerație. Rezultă că, notând cu *C* complexitatea în timp:

$$C(DELTA) = \sum_{i=1}^{n-4} O(i) = O(n^2).$$

Pentru euristica *W-E*, în [EAD82] se demonstrează că $C(W\text{-}E) = O(n^4)$ în cazul cel mai defavorabil. Pentru grafuri generate aleator și având un grad al nodurilor limitat, complexitatea este $O(n^3)$.

În cazul euristicii *GREEDY*, în cel mai rău caz, sunt parcurse $O(n^2)$ muchii, testul Hopcroft și Tarjan fiind apelat ca subrutină. Rezultă o complexitate $O(n^3)$.

3.8. Drum hamiltonian într-un graf planar maximal

Un *drum hamiltonian* într-un graf este un drum minimal ce trece prin fiecare nod măcar o dată. Lungimea drumului hamiltonian este dată de numărul total de muchii parcurse de drum. Fiind dat un graf $G = (V, E)$, numărul nodurilor este notat cu $p = |V|$. Un *drum de lungime k* al lui *G* este o secvență $v_0\ e_1\ v_1\ e_2 \ldots e_k\ v_k$, astfel încât

https://biblioteca-digitala.ro / https://unibuc.ro

nodurile muchiei e_i sunt v_{i-1} şi v_i, pentru fiecare $1 \le i \le k$. Lungimea unui drum W este notată $l(W)$. Un drum W este *drum închis maximal* (engl. *closed spanning walk*) al lui G dacă $v_0 = v_k$ şi fiecare nod din G apare măcar o dată. Un *drum hamiltonian* în G este un drum închis maximal de lungime minimală. Fie $h(G)$ lungimea unui drum hamiltonian în G. Este evident că $p \le h(G) \le 2(p - 1)$. Un *ciclu* este un drum închis ale cărui noduri sunt distincte. Un *ciclu hamiltonian* în G este un drum închis maximal de lungime p, adică un ciclu ce trece prin fiecare nod al lui G exact o dată. Un graf este *hamiltonian* dacă conţine un ciclu hamiltonian. Un *graf planar maximal* este un graf planar dacă prin adăugarea oricărei alte muchii se obţine un graf neplanar. Orice faţă a unui graf planar maximal este un triunghi. Un triunghi al unui graf planar maximal este numit *triunghi non-faţă* (engl. *nonface triangle*) dacă nu este frontiera unei feţe.

Pentru un triunghi non-faţă a unui graf planar maximal G, se notează cu $G_{TI}=(V_{TI},E_{TI})$ subgraful lui G din interiorul lui T şi cu $G_{TO}=(V_{TO},E_{TO})$ sugraful lui G din exteriorul lui T. Mai precis, dacă $T = x\ y\ z$ $(x, y, z \in V)$, $U'(T)$ este mulţimea nodurilor din interiorul lui T şi $U''(T)$ este mulţimea nodurilor din exteriorul lui T, atunci G_{TI} este subgraful lui G definit de mulţimea de noduri $\{x,y,z\} \cup U'(T)$, adică $G_{TI}=G-U''(T)$ şi G_{TO} este subgraful lui G definit de nodurile $\{x,y,z\} \cup U''(T)$, adică $G_{TO}=G-U'(T)$. Cu p_{TI} şi p_{TO} se notează respectiv $|V_{TI}|$ şi $|V_{TO}|$.

În algoritmul propus se folosesc două strategii şi anume: *divide et impera* în Algoritmul 3.11., numit *HWALK*, şi *augmentarea*, descrisă mai jos, în Algoritmul 3.12., numit *LCYCLE*. Fiind dat un triunghi T ce satisface anumite condiţii exprimate cu ajutorul valorilor p_{TI} şi p_{TO}, se procedează astfel:

Algoritmul 3.11. (*HWALK*)

1. [Divizare] Se împarte graful G în două grafuri maximale planare G_{TI} şi G_{TO}.
2. [Prelucrare] Se apelează algoritmul pentru drumuri maximale în G_{TI} şi G_{TO}.
3. [Combinare] Se combină drumurile maximale din G_{TI} şi G_{TO} într-un drum maximal al grafului G. ■

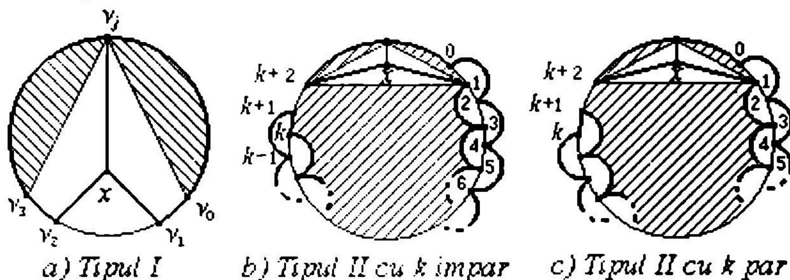


Fig. 3.27.

https://biblioteca-digitala.ro / https://unibuc.ro

Algoritmul 3.12. (*LCYCLE*)

Ori de câte ori un graf conține un nod x al unui ciclu C curent obținut și satisfăcând anumite condiții, unele muchii ale lui C sunt înlocuite cu alte muchii, astfel încât x să devină un nod al unui nou ciclu, având o lungime cu unu mai mare. Fie configurațiile din figura 3.27, unde C este reprezentat sub forma $C = v_0\, v_1\, v_2 \ldots v_0$. În figură un ciclu vechi este desenat cu linii subțiri și unul nou cu linii pline.

1. Figura 3.27.a) reprezintă o configurație în care G are un nod x al lui C adiacent cu nodurile v_1 și v_2 ale muchiei $[v_1,v_2]$ din C. De notat că, probabil, $v_j = v_3$, unde v_j este un al treilea nod la care este adiacent x. (Se știe că orice nod din C are gradul 3). Evident că ciclul $C' = v_0\, v_1\, x\, v_2\, v_3 \ldots v_0$ din G este mai lung decât C.
2. Figurile 3.27.b) și 3.27.c) reprezintă configurații, în care pentru un anumit $k \geq 1$ sunt satisfăcute condițiile:

   i) $(v_{i-1}, v_{i+1}) \in E$ pentru fiecare i, $1 \leq i \leq k$, și

   ii) un nod x al lui C este adiacent cu v_1 și v_{k+2}.

Pentru simplificare, nodurile v_i sunt notate în figurile 3.27.b) și 3.27.c) cu i. Dacă k este impar, atunci ciclul:

$$C' = v_0\, v_2\, v_4 \ldots v_{k-1}\, v_{k+1}\, v_k\, v_{k-2} \ldots v_3\, v_1\, x\, v_{k+2} \ldots v_0$$

al lui G este mai lung decât C (vezi figura 3.27.b). Dacă k este par, atunci ciclul:

$$C' = v_0\, v_2\, v_4 \ldots v_k\, v_{k+1}\, v_{k-1} \ldots v_3\, v_1\, x\, v_{k+2} \ldots v_0$$

din G este mai lung decât C (vezi figura 3.27.c). De notat că o configurație de tipul 1. poate fi interpretată ca un caz special al tipului 2. cu $k = 0$. ■

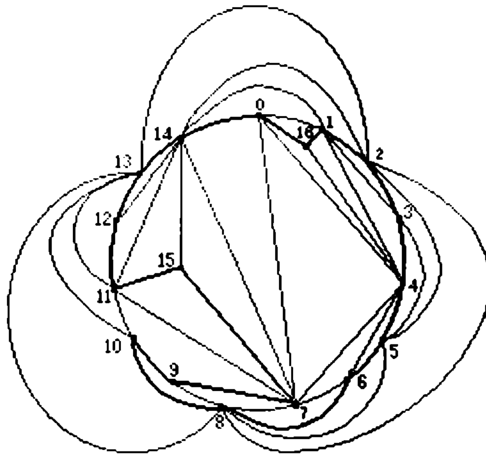


Fig. 3.28.

Exemplul 3.9. În figura 3.28. este dat un graf planar maximal $G = (V, E)$, cu $V = \{0, 1, 2, \ldots, 16\}$. Ciclul $C = 0\ 1\ 2 \ldots 14\ 0$ este reprezentat prin arce pe cerc.

https://biblioteca-digitala.ro / https://unibuc.ro

Nodurile 15 şi 16 ale ciclului C au gradul 3. Graful G conţine o configuraţie de tipul 1. relativ la nodul 16: [16, 0], [16, 1] $\in E$ şi o configuraţie de tip 2. relativ la nodul 15: [6, 8], [7, 9], [8, 10], [15, 7], [15, 11] $\in E$ ($k = 3$). Noul ciclu:

$$C' = 0\ 16\ 1\ 2\ 3\ 5\ 4\ 5\ 6\ 8\ 10\ 9\ 7\ 15\ 11\ 12\ 13\ 14\ 0,$$

mai lung decât C, este desenat cu linii pline. ■

Algoritmul *HWALK* identifică într-un timp $O(p^2)$ un drum maximal închis a cărui lungime este cel mult egală cu 3/2 (p -3) dacă graful are $p \geq 11$ noduri.

3.9. Algoritm euristic pentru generarea de separatori mici în grafuri arbitrare

Punerea problemei. În multe aplicaţii este util să se poată efectua descompunerea unui graf G în două părţi G_1 şi G_2 conectate printr-un număr mic de arce. Un exemplu ilustrativ îl constituie proiectarea circuitelor electronice dreptunghiulare, astfel încât cele două părţi ale unui astfel de circuit să fie interconectate printr-un număr mic de conexiuni. ■

Aplicaţii posibile: circuite electronice şi electrice, proiectarea unor sisteme şi baze de date distribuite.

O *secţiune* într-un graf $G=(V,E)$ cu n noduri este o partiţie $(W,\overline{W})$ a mulţimii V. Un *separator* S într-un graf $G = (V, E)$ cu n noduri este o partiţie $C = (W,\overline{W})$ a mulţimii V astfel încât $|W| \geq n / 3$ şi $|\overline{W}| \geq n / 3$, unde cu $|A|$ este notat numărul de elemente al mulţimii A. Un asemenea separator se numeşte *mic* dacă numărul de arce (v, w) cu v în W şi w în $\overline{W}$ este mic.

Algoritmul 3.13. (A)

Ideile de bază ale euristicii propuse, numită *algoritmul A*, sunt următoarele. Fiind dat un graf conex G, se consideră problema p a fluxului, în care o jumătate din nodurile lui G sunt alese drept surse şi cealaltă jumătate drept destinaţii (engl: *sink*). Fie V_A mulţimea surselor şi V_Z mulţimea destinaţiilor. Fie de asemenea un nou nod sursă A conectat prin arce cu toate sursele şi un nou nod destinaţie Z conectat prin arce cu toate destinaţiile. Capacităţile (de transport) ale tuturor arcelor incidente cu A şi Z sunt egale cu unu şi capacităţile arcelor lui G au valoarea x (în ambele sensuri). Mărimea x constitue *parametrul problemei transporturilor*. Iniţial valoarea x se alege suficient de mare astfel încât nici unul dintre arcele lui G nu este saturat de fluxul generat de capacităţile arcelor incidente cu A. Pentru aceasta, este suficientă o valoare x egală cu n / 2. Arcele din A în V_A şi, respectiv, cele din V_Z în Z sunt saturate

https://biblioteca-digitala.ro / https://unibuc.ro

(deoarece capacităţile lor au valoarea 1). În continuare, valoarea x este redusă succesiv, folosind de exemplu căutarea binară, până când se identifică o valoare x' astfel încât problema fluxului $p(x' + \varepsilon)$ defineşte un flux maximal în care nici unul dintre arcele lui G nu este saturat, dar problema $p(x')$ defineşte un flux maximal care saturează unele dintre arcele lui G. De asemenea, problema fluxului $p(x' - \varepsilon)$ are un flux maximal care nu saturează un arc oarecare (A, y) sau un arc oarecare (w, Z). De exemplu, dacă G are un singur arc între a şi b, a fiind sursă şi b destinaţie, atunci dacă $x > 1$, arcele (A, a) şi (b, Z) vor fi saturate în cazul unui flux maximal. Dacă $x = 1$, arcul de la a la b va fi de asemenea saturat în cazul unui flux maximal. Rezultă că problema $p(1 + \varepsilon)$ are un flux maximal pentru care nici unul dintre arcele lui G nu este saturat, dar $p(1)$ are un flux maximal care saturează un arc oarecare din G. De asemenea, $p(1 - \varepsilon)$ are un flux maximal care nu saturează arcele (A, a) şi (b, Z). Dacă valoarea iniţială a lui x este n, atunci $\log(n)$ iteraţii reduc incertitudinea la 1 şi fiecare iteraţie următoare adaugă un supliment semnificativ de informaţie. Rezultă că $O(\log(n))$ iteraţii sunt suficiente pentru a obţine o valoare semnificativă a fluxului.

Fie A' mulţimea nodurilor y din G astfel încât arcul (A, y) este nesaturat într-un flux maximal pentru problema $p(x' - \varepsilon)$. Fie Z' mulţimea nodurilor din G astfel încât arcul (y, Z') nu este saturat în acest flux. Se poate arăta că arcele saturate ale lui G separă pe A' de Z' şi că A' şi Z' sunt nevide. Rezultă că arcele saturate ale lui G separă pe G şi cu ajutorul lor se poate obţine o secţiune C care separă pe A' de Z' astfel încât toate fluxurile prin această secţiune au acelaşi sens. Această secţiune se poate obţine uşor folosind algoritmul de flux maximal. ■

Eficacitatea algoritmului este dată de următoarea teoremă.

Teorema 3.3. Algoritmul A identifică întotdeauna o secţiune având o capacitate a arcelor aproape maximală. ■

Pentru deducerea unei margini privind performanţa algoritmului A se introduce următoarea definiţie.

Definiţia 3.2. Calitatea $g(C)$ a unei secţiuni $C = (W, W')$ este definită de raportul $\min(|W|,|\overline{W}|)/|C|$. ■

Teorema 3.4. Fie $C=(W,\overline{W})$ o secţiune a grafului G. Atunci, pentru n, $|W|$ şi $|\overline{W}|$ tinzând către infinit, probabilitatea ca algoritmul A să identifice o secţiune având o calitate diferită de $g(C)$ cu mai puţin de $g(C)/\sqrt{n}$ se apropie de o constantă mai mare ca 0.3. Această probabilitate creşte dacă raportul $\min(|W|,|\overline{W}|)/n$ scade. ■

Se poate întâmpla ca secţiunea C să nu fie un separator, dacă ea partiţionează mulţimea nodurilor în două submulţimi de mărimi foarte diferite. În acest caz, este

https://biblioteca-digitala.ro / https://unibuc.ro

posibilă aplicarea algoritmului A pe mulţimea mai mare şi repeta acest proc·s de mai multe ori. Combinând împreună aceste secţiuni într-un mod convenabil se poate obţine un separator acceptabil.

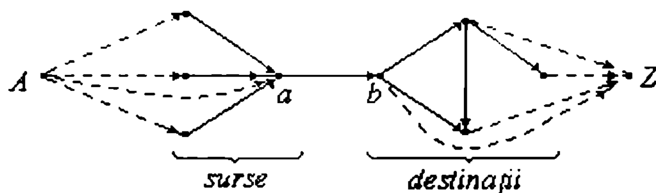


Fig. 3.29.

Observaţii. 1. Se presupune că numărul de noduri din G este par.

2. Un exemplu de graf pentru care valoarea n / 2 este atinsă este de forma celui din figura 3.29. (pentru cazul n = 8).

Arcele punctate conectează pe A cu mulţimea V_A a nodurilor sursă şi pe Z cu mulţimea V_Z a nodurilor destinaţie. În acest caz o capacitate de transport . arcului (a, b) egală cu patru nu este suficientă pentru a asigura nesaturarea acestui arc.

https://biblioteca-digitala.ro / https://unibuc.ro

4 PROBLEMA COMISVOIAJORULUI

Problema comisvoiajorului este definită în paragraful 1.1.2.2.1., unde este prezentat şi un prim algoritm euristic de soluţionare (*NN*). Un alt algoritm euristic, bazat pe arborele parţial al unui graf, se află în paragraful 1.1.2.4.2. În continuare vom da alte câteva metode euristice de soluţionare a problemei comisvoiajorului, unele restricţii şi generalizări ale acestei probleme şi, în final, sunt prezentate aplicaţii în care intervin astfel de algoritmi.

4.1. Metode euristice pentru problema comisvoiajorului

Metodele aproximative propuse sunt de tip secvenţial şi anume la fiecare pas are loc conectarea unor perechi de oraşe. Procesul conectării oraşelor trebuie să evite generarea de cicluri. Fiecare oraş este conectat cu cel mult altele două şi oraşele situate la extremităţile unui drum nu pot fi conectate decât în cazul în care toate oraşele au fost parcurse şi aparţin toate unui singur drum.

4.1.1. Metoda extinderii bilaterale

În prima metodă propusă aici, numită *metoda extinderii bilaterale*, se pleacă de la un drum ce conţine o muchie de cea mai mică lungime şi sunt adăugate pe rând la acest drum celelalte oraşe. La fiecare nouă conectare se alege unul dintre oraşele ce nu se află în drum şi care se află cel mai aproape de una din extremităţile drumului deja construit. În final se unesc cele două capete ale drumului obţinându-se un tur. Aceasta constituie ideea de bază folosită în construirea algoritmului 4.1. În acest algoritm se consideră drept intrare un graf nedirecţionat complet cu n oraşe ce determină distanţele dintre perechile de oraşe considerate. Acest algoritm este asemănător cu metoda *NN*. Deosebirea dintre ele este aceea că în metoda *NN* extinderea se face prin adăugarea unui nou oraş la un singur capăt al drumului,

https://biblioteca-digitala.ro / https://unibuc.ro

celălalt capăt rămânând fix (oraşul de plecare), pe când în acest algoritm drumul curent se poate extinde în ambele direcţii (adăugarea se poate face la oricare dintre cele două capete).

Algoritmul 4.1. (Extindere bilaterală)

1. [Iniţializare] Se determină drumul $P_2 = [a, b]$, unde $[a, b]$ este una din cele mai scurte muchii şi se face $i = 2$.
2. [Terminare] Dacă $i = n$, se unesc capetele drumului şi STOP.
3. [Alegere] Se alege din mulţimea $\{[x,a] | x \notin P_i\} \cup \{[x,b] | x \notin P_i\}$, a şi b fiind cele două capete ale drumului P_i, cea mai scurtă muchie $[x,y]$ astfel încât $x \notin P_i$ şi $y \in \{a,b\}$.
4. [Adăugare] Se face $i = i + 1$; dacă $y = a$, atunci se face $P_i = [x, a, ..., b]$ şi $a = x$, altfel se face $P_i = [a, ..., b, x]$ şi $b = x$.
5. [Ciclare] Se merge la pasul 2. ■

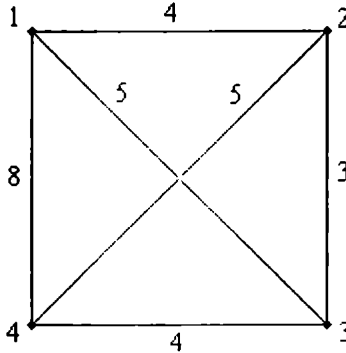


Fig. 4.1.

Exemplul 4.1. Pentru graful din figura 4.1., aplicând algoritmul 4.1. se obţin drumurile: $P_2 = [2,3]$, $P_3 = [1,2,3]$, $P_4 = [1,2,3,4]$ şi în final se obţine ciclul $[1,2,3,4,1]$ de lungime 19. Un tur optimal este $[1,2,4,3,1]$ de lungime 18. ■

4.1.2. Metoda reducerii

Această metodă poate fi privită ca o generalizare a metodei precedente. Se construiesc trei mulţimi de noduri: *IZOLAT*, ce conţine oraşele care nu au fost unite cu alte oraşe, *CAPĂT*, ce conţine oraşele care au fost unite cu un singur alt oraş şi *PROCESAT*, ce conţine oraşele care au fost unite cu alte două oraşe şi o mulţime de muchii *TUR*, ce conţine muchiile selectate pentru turul selectat. Algoritmul este:

Algoritmul 4.2. (*Reducere*)

1. [Iniţializare] Se face $IZOLAT=\{1,2,...,n\}$, $CAPĂT=\emptyset$, $PROCESAT=\emptyset$, $TUR=\emptyset$ şi se ordonează muchiile grafului în ordinea nedescrescătoare a lungimii lor.

https://biblioteca-digitala.ro / https://unibuc.ro

2. [Terminare] Dacă *IZOLAT* = ∅ şi *CAPĂT* conţine două oraşe, se adaugă la mulţimea *TUR* muchia ce uneşte oraşele aflate în *CAPĂT* şi STOP.
3. [Muchie nouă] Se elimină prima muchie $[x, y]$ din mulţimea ordonată de muchii.
4. [Muchie nefolositoare] Dacă $x \in$ *PROCESAT* sau $y \in$ *PROCESAT* sau x şi y sunt legate prin muchiile deja considerate în *TUR*, atunci se merge la pasul 3.
5. [Adăugare] Se adaugă muchia $[x, y]$ la mulţimea *TUR*..
6. [Situare x] Dacă $x \in$ *IZOLAT*, se scoate x din *IZOLAT* şi se trece în *CAPĂT*, altfel se scoate x din *CAPĂT* şi se trece în *PROCESAT*.
7. [Situare y] Dacă $y \in$ *IZOLAT*, se scoate y din *IZOLAT* şi se trece în *CAPĂT*, altfel se scoate y din *CAPĂT* şi se trece în *PROCESAT*.
8. [Ciclare] Se merge la pasul 2. ■

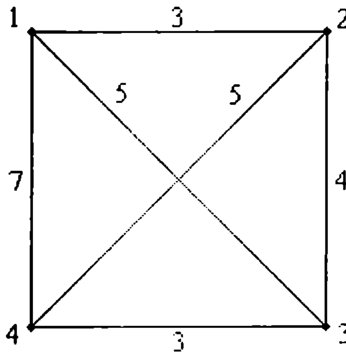


Fig. 4.2.

Exemplul 4.2. Pentru graful din figura 4.2., aplicând algoritmul 4.2. se obţine mai întâi *IZOLAT*={1,2,3,4}, *CAPĂT*=∅, *PROCESAT*=∅, *TUR*=∅ şi mulţimea muchiilor ordonate {[1,2], [3,4], [2,3], [1,3], [2,4], [1,4]}. Considerarea muchiei [1,2] produce următoarele modificări: *IZOLAT*={3,4}, *CAPĂT*={1,2}, *PROCESAT*=∅, *TUR*={[1,2]} şi mulţimea muchiilor ordonate {[3,4], [2,3], [1,3], [2,4], [1,4]}. Considerarea muchiei [3,4] produce următoarele modificări: *IZOLAT*=∅, *CAPĂT*={1,2,3,4}, *PROCESAT*=∅, *TUR*={[1,2],[3,4]} şi mulţimea muchiilor ordonate {[2,3],[1,3], [2,4], [1,4]}. Considerarea muchiei [2,3] produce următoarele modificări: *IZOLAT*=∅, *CAPĂT*={1,4}, *PROCESAT*={2,3}, *TUR*={[1,2],[3,4],[2,3]} şi mulţimea muchiilor ordonate {[1,3],[2,4],[1,4]}. Acum sunt îndeplinite condiţiile din pasul 2 rezultând *TUR*={[1,2],[3,4],[2,3],[1,4]}, din care rezultă ciclul [1,2,3,4,1] de lungime 17. Un tur optimal este [1,2,4,3,1] de lungime 16. ■

https://biblioteca-digitala.ro / https://unibuc.ro

4.1.3. Metoda proximității

Deoarece fiecare oraş este conectat cu altele două, rezultă că lungimea minimală a turneului, respectiv marginea inferioară, este cel puțin egală cu jumătatea sumei distanțelor dintre fiecare oraş şi două dintre oraşele cele mai apropiate lui. În practică această limită este atinsă destul de rar.

În metoda următoare, numită *metoda proximității* (engl. *proximity method*), oraşele sunt conectate în ordinea crescătoare a distanțelor de conectare. Pentru oraşele izolate se consideră cele mai apropiate două oraşe care au cel mult o legătură, dar nelegate între ele şi pentru oraşele cu o legătură se consideră acea legătură şi cel mai apropiat oraş cu cel mult o legătură şi diferit de celălat capăt al drumului la care aparține oraşul respectiv. Algoritmul este următorul:

Algoritmul 4.3. (***Proximitate***)

1. [Inițializare] Se face $IZOLAT = \{1, 2, ..., n\}$, $CAPĂT = \varnothing$, $PROCESAT = \varnothing$, $TUR = \varnothing$ şi $CAP(i) = 0$, pentru $i = 1, 2, ..., n$.
2. [Terminare] Dacă $IZOLAT = \varnothing$ şi *CAPĂT* conține două oraşe, se adaugă la mulțimea *TUR* muchia ce uneşte oraşele aflate în *CAPĂT* şi STOP; altfel, dacă $IZOLAT = \{i\}$ şi $CAPĂT = \{j, k\}$, se adaugă la mulțimea *TUR* muchiile $[i, j]$ şi $[i, k]$ apoi STOP.
3. [Oraşe izolate] Pentru fiecare $i \in IZOLAT$ se determină două oraşe j şi k din $IZOLAT \cup CAPĂT$, $k \neq CAP(j)$ care minimizează expresia $d_i = w(i, j) + w(i, k)$.
4. [Oraşe capete] Pentru fiecare $i \in CAPĂT$ se determină oraşul $j \neq CAP(i)$ din $IZOLAT \cup CAPĂT$ care minimizează $d_i = w(i, j) + w(i, k)$, unde $[i, k] \in TUR$.
5. [Alegere] Se alege $i \in IZOLAT \cup CAPĂT$ pentru care valoarea d_i este minimă.
6. [i izolat] Dacă $i \in IZOLAT$, se adaugă la *TUR* muchiile $[i, j]$ şi $[i, k]$, unde j şi k sunt determinate la pasul 3, se mută j şi k din *IZOLAT* în *CAPĂT* sau din *CAPĂT* în *PROCESAT* şi se schimbă corespunzător valorile din *CAP* pentru j k şi alte capete legate de ele.
7. [i capăt] Dacă $i \in CAPĂT$, se adaugă la *TUR* muchia $[i, j]$, unde j este determinat la pasul 4, se mută j din *IZOLAT* în *CAPĂT* sau din *CAPĂT* în *PROCESAT* şi se schimbă valorile din *CAP* pentru j şi eventual celălalt capăt legat de el.
8. [Marcare] Se trece i în *PROCESAT* şi se face $CAP(i) = 0$.
9. [Ciclare] Se merge la pasul 2. ■

Exemplul 4.3. Pentru rețeaua din figura 4.3.a), marginea inferioară a lungimii turneului este egală cu:

$$\frac{1}{2}\,[(1 + 3) + (1 + 2) + (2 + 3) + (3 + 4) + (2 + 3)] = \frac{1}{2}(4 + 3 + 5 + 7 + 5) = 12,$$

https://biblioteca-digitala.ro / https://unibuc.ro

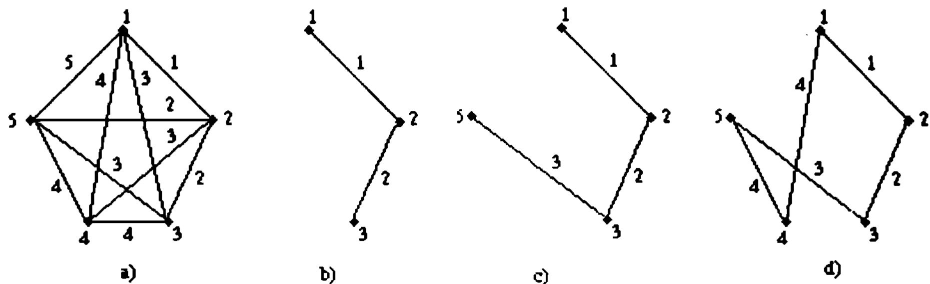


Fig. 4.3.

unde perechile de numere din patrantezele interioare se referă la nodurile 1, 2, ..., 5 ale reţelei date obţinând valorile $d_1 = 4$, $d_2 = 3$, $d_3 = 5$, $d_4 = 7$ şi $d_5 = 5$. Se vede că oraşul 2 urmează a fi ales, deoarece suma distanţelor lui la oraşele 1 şi 3 este cea mai mică dintre sumele minimale ale distanţelor celorlate oraşe la câte alte două oraşe. Configuraţia obţinută este arătată pe figura 4.3.b). Noile valori calculate sunt $d_1 = 5$, $d_3 = 5$, $d_4 = 7$ şi $d_5 = 7$. Următoarea conectare ce evită generarea unui ciclu şi asigură o creştere minimală este, de exemplu, cea dintre oraşele 3 şi 5 (ar mai putea fi şi cea dintre 1 şi 4), configuraţia generată fiind arătată pe figura 4.3.c). Acum sunt îndeplinite condiţiile din pasul 2 (partea a doua) şi se uneşte oraşul 4 cu capetele 1 şi 5 ale drumului deja generat cum se arată în figura 4.3.d). Turneul generat are lungimea 14, care depăşeşte cu 2 marginea inferioară. ■

4.1.4. Metode bazate pe pierderi

Pentru formalizarea altor euristici, se introduce noţiunea de *pierdere* (engl. *loss*) a oraşului *i*, notată *pierd*(*i*) şi definită după cum urmează. Fie $l(1, i)$, $l(2, i)$, şi $l(3, i)$ cele mai apropiate trei oraşe de *i*, excluzându-le pe cele conectate de două ori şi, dacă *i* este deja conectat, excluzând oraşul situat la cealaltă extremitate lanţului. şi $d(1, i)$, $d(2, i)$, şi $d(3, i)$ distanţele corespunzătoare dintre *i* şi cele trei oraşe menţionate. Pierderea unui oraş *i*, notată *pierd*(*i*), măsoară diferenţa dintre realizarea celei mai favorabile configuraţii pentru nodul respectiv şi cea mai defavorabilă configuraţie ce s-ar putea obţine în urma unei conexiuni provocată de un alt oraş ce se poate uni cu unul dintre candidaţii oraşului *i* la un moment anterior, anulând astfel unele dintre posibilităţile de cuplare.

Algoritmul euristic care ţine seama de pierderi are următoarea structură:

Algoritmul 4.4. (***Pierderi***)

1. [Iniţializare] Se face *IZOLAT* = {1, 2, ..., *n*}, *CAPĂT* = ∅, *PROCESAT* = ∅, *TUR* = ∅ şi *CAP*(*i*) = 0, pentru *i* = 1, 2, ..., *n*.

https://biblioteca-digitala.ro / https://unibuc.ro

2. [Terminare] Dacă *IZOLAT* = ∅ şi *CAPĂT* conţine două oraşe, se adaugă la mulţimea *TUR* muchia ce uneşte oraşele aflate în *CAPĂT* şi STOP; altfel, dacă *IZOLAT* = $\{i\}$ şi *CAPĂT* = $\{j, k\}$, se adaugă la mulţimea *TUR* muchiile $[i, j]$ şi $[i, k]$ apoi STOP.
3. [Pierderi] Se calculează *pierd*(i) pentru fiecare oraş $i \in$ *IZOLAT* ∪ *CAPĂT*.
4. [Alegere] Se alege din mulţimea *IZOLAT* ∪ *CAPĂT* oraşul i pentru care *pierd*(i) este cea mai mare.
5. [Modificări] Se face pentru i conexiunea cea mai favorabilă şi se efectuează modificările corespunzătoare ei.
6. [Ciclare] Se merge la pasul 2. ■

În continuare vom da câteva metode posibile de calcul pentru pierderile asociate unui oraş. Aceste funcţii sunt comparabile ca eficienţă, putându-se da contraexemple prin care se demonstrează că nu există o relaţie de dominare între ele.

4.1.4.1. Pierdere simplă

Dacă i nu este conectat cu un alt oraş, adică este izolat, atunci sunt necesare două conexiuni, în situaţia cea mai favorabilă fiind oraşele $l(1, i)$ şi $l(2, i)$ şi cea defavorabilă fiind oraşele $l(2, i)$ şi $l(3, i)$, deci:

$$pierd(i) = d(3, i) - d(1, i)$$

şi, dacă i este deja conectat, atunci este necesară o conexiune, în situaţia cea mai favorabilă fiind oraşul $l(1, i)$ şi cea defavorabilă fiind oraşul $l(2, i)$, deci:

$$pierd(i) = d(2, i) - d(1, i),$$

În ambele cazuri, dacă *pierd*(i) este pierderea maximală dintre toate pierderile nodurilor care nu au două conexiuni, i este conectat la cel mai apropiat vecin, indicat de $l(1, i)$. Această funcţie are avantajul simplităţii.

4.1.4.2. Pierdere compusă

Să considerăm trei exemple pentru care pierderea simplă este *logic inconsistentă*. Primul dintre ele este al unui oraş i neconectat, cele două mai apropiate oraşe de el fiind conectate între ele. În acest caz, oraşul i nu poate fi conectat la cele mai apropiate două oraşe, deoarece în felul acesta ar rezulta un ciclu. Legăturile alese vor fi atunci către primul cel mai apropiat oraş şi cel de al treilea. Dacă legătura preferată la cel mai apropiat oraş nu este obţinută, pierderea minimală sau costul minimal se realizează prin conectarea la al doilea cel mai apropiat oraş şi, dacă legătura la al treilea cel mai apropiat oraş nu este obţinută, atunci pierderea minimală sau costul minimal se realizează pe calea conectării la al patrulea cel mai apropiat oraş. Deci, dacă $l(1, i)$ este conectat cu $l(2, i)$ şi i este neconectat, atunci:

https://biblioteca-digitala.ro / https://unibuc.ro

$$pierd(i) = \max\{d(2, i) - d(1, i), d(4, i) - d(3, i)\},$$

după care se preferă conectarea lui i la $l(1, i)$, respectiv $l(3, i)$, după cum $pierd(i)$ este egală cu $d(2, i) - d(1, i)$, respectiv $d(4, i) - d(3, i)$.

Al doilea exemplu este cel în care cele două capete ale unui drum au ambele drept candidat de legătură un acelaşi oraş. Al treilea exemplu este cel în care capetele unui drum au printre candidate oraşe ce sunt capetele altui drum, fiecare în parte un alt capăt. Se mai pot obţine multe alte combinaţii care pot duce la cicluri.

Ţinând seama de toate aceste situaţii se poate construi o funcţie *pierd*(i) mai complicat de explicat prin multitudinea situaţiilor analizate. Pentru cei interesaţi prezentăm o variantă a acestei funcţii în limbaj C. Menţionăm că în limbajul C cea mai mică valoare a unui index este 0, nu 1 ca în alte limbaje, şi, de aceea, valorile corespunzătoare indicilor oraşelor sunt diminuate cu o unitate. Numărul maxim de oraşe este dat de NMO fiind în cazul nostru 1000. Parametrii de intrare sunt numărul n de oraşe, matricea l care dă pe fiecare coloană i indicii celor mai apropiate patru oraşe cu cel mult o legătură de oraşul i, matricea d care dă distanţele corespunzătoare oraşelor indicate în matricea l şi vectorul *cap* în care se pune indicele celuilalt capăt al unui drum dacă oraşul curent este un capăt de drum sau se pune valoarea -2 dacă oraşul curent are deja două legături sau se pune valoarea -1 dacă oraşul curent nu are nici o legătură, adică este nod izolat. Parametrii de ieşire sunt vectorii *pref* în care se păstrează preferinţa fiecărui oraş pentru o legătură care ar produce o pierdere mai mare prin nerealizare şi *pierd* care dă pierderea maximă prin nerealizarea uneia dintre legăturile optime. Fiecare rând al funcţiei este comentat pentru a se înţelege diferitele cazuri considerate.

```
#define NMO 1000
void pierdere(int n, int l[4][NMO], float d[4][NMO], int cap[NMO], int pref[NMO],
                float pierd[NMO])
{int i, j, k, a, b, c;
 for(i = 0; i < n; i++)            // parcurgere oraşe pentru calculul pierderilor;
 {pierd[i] = 0; pref[i] = -1;      // iniţializare pierdere şi preferinţă pentru oraşul i;
  if(cap[i] == -2) continue;       // pentru oraş i cu două legături nu se fac investigaţii;
  a = l[0][i]; b = l[1][i];        // fixare legături posibile cele mai apropiate de i;
  pref[i] = a;          // fixare cel mai apropiat oraş ca preferatul principal al lui i;
  if(cap[i] == -1)      // cazul unui oraş izolat (fără legături deja considerate);
  {pierd[i] = d[2][i] - d[0][i];             // se leagă i cu oraşul l(3, i) în loc de l(1, i);
   if(cap[a] == l[1][i])                     // oraşele l(1, i) şi l(2, i) legate între ele;
```

https://biblioteca-digitala.ro / https://unibuc.ro

```
 {pierd[i] = d[1][i] - d[0][i];              // pierdere oraş l(2, i) în loc de l(1, i);
  if(pierd[i] < d[3][i] - d[2][i])           // pierdere mai mare cu l(4, i) în loc de l(3, i);
  {pierd[i] = d[3][i] - d[2][i]; pref[i] = l[2][i];}        // schimbare în i;
 }                                   // terminare calcule oraşele l(1, i) şi l(2, i) legate;
 else                                        // oraşele l(1, i) şi l(2, i) nu sunt legate;
  if(cap[a] == l[2][i])                      // oraşele l(1, i) şi l(3, i) legate între ele;
  {if(pierd[i] < d[3][i] - d[1][i])          // pierdere mai mare cu l(4, i) în loc de l(2, i);
   {pierd[i] = d[3][i] - d[1][i]; pref[i] = l[1][i];}       // schimbare în i;
  }                                  // terminare calcule oraşele l(1, i) şi l(3, i) legate;
  else                                       // oraşele l(1, i) şi l(3, i) nu sunt legate;
  {if(cap[b] == l[2][i])                     // oraşele l(2, i) şi l(3, i) legate între ele;
    pierd[i] = d[3][i] - d[0][i];            // pierdere oraş l(4, i) în loc de l(1, i);
  }                        // terminare calcule oraşele l(1, i), l(2, i) şi l(3, i) nelegate;
}                                            // terminare calcule pentru oraş i izolat;
else                                         // cazul unui oraş cu o legătură deja stabilită;
{pierd[i] = d[1][i] - d[0][i];               // pierdere oraş l(2, i) în loc de l(1, i);
 j = cap[i];                                 // j este celălalt capăt legat de i;
 if(j < i)                                   // capătul corespunzător lui i a fost procesat;
                                    // coordonează alegerile între cele două capete;
 {k = l[1][j];                               // k este noua preferinţă posibilă pentru j;
  if(l[0][i] == l[0][j])                                    // l(1, i) acelaşi cu l(1, j);
   if(l[1][i] == l[1][j])                                   // l(2, i) acelaşi cu l(2, j);
    if(l[1][i] == cap[a])                          // oraşele l(1, i) şi l(2, i) legate;
     if(d[2][i] - d[0][i] < d[2][j] - d[0][j])              // alegere între i şi j;
     {pierd[i] = d[3][i] - d[2][i]; pref[i] = l[2][i];}     // schimbare i;
     else                                                   // se modifică alegerea din j;
     {pierd[j] = d[3][j] - d[2][j]; pref[j] = l[2][j];}     // schimbare j;
    else                                           // oraşele l(1, i) şi l(2, i) nelegate;
     if(pierd[j] > pierd[i])                 // se face schimbare pentru preferinţa lui i;
     {pierd[i] = d[2][i] - d[1][i];          // pierdere oraş l(3, i) în loc de l(2, i);
      pref[i] = l[1][i];                     // se preferă oraşul l(2, i) în i;
      if(l[2][i] == cap[a])                  // sunt legate oraşele l(1, i) cu l(3, i),
       pierd[i] = d[3][i] - d[1][i];         // pierdere oraş l(4, i) în loc de l(2, i);
      if(l[2][j] == cap[k])                  // sunt legate oraşele l(2, j) şi l(3, j);
       pierd[j] = d[3][j] - d[0][j];         // pierdere oraş l(4, j) în loc de l(1, j);
      else                                   // nu sunt legate oraşele l(2, j) şi l(3, j);
```

https://biblioteca-digitala.ro / https://unibuc.ro

```
   pierd[j] = d[2][j] - d[0][j];        // pierdere oraş l(3,j) în loc de l(1,j);
  }                                     // terminare schimbări pentru preferința lui i;
  else                                  // se face schimbare pentru preferința lui j;
  {pierd[i] = d[2][i] - d[0][i];        // pierdere oraş l(3, i) în loc de l(1, i);
   if(l[2][i] == cap[b])                // sunt legate oraşele l(2, i) cu l(3, i)·
    pierd[i] = d[3][i] - d[0][i];       // pierdere oraş l(4, i) în loc de l(1, i);
   if(l[2][j] == cap[a])                // sunt legate oraşele l(1, i) şi l(3,j);
    pierd[j] = d[3][j] - d[1][j];       // pierdere oraş l(4,j) în loc de l(2,j);
   else                                 // nu sunt legate oraşele l(1, i) şi l(3,j);
    pierd[j] = d[2][j] - d[1][j];       // pierdere oraş l(3,j) în loc de l(2,j);
   pref[j] = l[1][j];                   // preferatul lui j este oraşul l(2,j);
  }                                     // terminare schimbări pentru preferința lui j;
else                                    // l(2, i) acelaşi cu l(2,j);
 if(l[1][i] == cap[a])                  // oraşele l(1, i) şi l(2, i) legate;
  if(pierd[j] < d[2][i] - d[0][i])      // j cu pierdere mai mică decât i;
  {if(l[2][j] == cap[a])                // sunt legate oraşele l(1, i) şi l(3,j);
    pierd[j] = d[3][j] - d[1][j];       // pierdere oraş l(4,j) în loc de l(2,j);
   else                                 // nu sunt legate oraşele l(1, i) şi l(3,j);
    pierd[j] = d[2][j] - d[1][j];       // pierdere oraş l(3,j) în loc de l(2,j)·
   pref[j] = l[1][j];                   // preferatul lui j este oraşul l(2,j);
  }                        // terminare calcule j cu pierdere mai mică decât i;
  else                                  // i cu pierdere mai mică decât j;
  {pierd[i] = d[3][i] - d[2][i]; pref[i] = l[2][i];}          // schimbare în i;
 else                                   // oraşele l(1, i) şi l(2, i) nu sunt legate;
  if(l[1][j] == cap[a])                 // sunt legate oraşele l(1, i) şi l(2,j);
   if(pierd[i] < d[2][j] - d[0][j])     // pierdere în i mai mică decât în j;
   {if(l[2][i] == cap[a])               // oraşele l(1, i) şi l(3, i) legate;
     pierd[i] = d[3][i] - d[1][i];      // pierdere oraş l(4, i) în loc de l(2, i);
    else                                // oraşele l(1, i) şi l(3, i) nu sunt legate;
     pierd[i] = d[2][i] - d[1][i];      // pierdere oraş l(3, i) în loc de l(2, i);
    pref[i] = l[1][i];                  // schimbare preferință i pentru l(2, i);
   }                       // terminare calcule pierdere în i mai mică decât în j;
   else                                 // pierdere în j mai mică decât în i;
   {pierd[j] = d[3][j] - d[2][j]; pref[j] = l[2][j];}          // schimbare j;
  else                                  // oraşele l(1, i) şi l(2,j) nu sunt legate;
   if(pierd[j] > pierd[i])              // se face schimbare pentru preferința lui i;
```

https://biblioteca-digitala.ro / https://unibuc.ro

```
    {if(l[2][i] == cap[a])           // oraşele l(1, i) şi l(3, i) legate;
      pierd[i] = d[3][i] - d[1][i];  // pierdere oraş l(4, i) în loc de l(2, i);
     else                            // oraşele l(1, i) şi l(3, i) nu sunt legate;
      pierd[i] = d[2][i] - d[1][i];  // pierdere oraş l(3, i) în loc de l(2, i);
     pref[i] = l[1][i];              // schimbare preferinţă i pentru l(2, i);
    }                        // terminare calcule pierdere în i mai mică decât în j;
    else                             // se face schimbare pentru preferinţa lui j;
    {if(l[2][j] == cap[a])           // sunt legate oraşele l(1, i) şi l(3, j);
      pierd[j] = d[3][j] - d[1][j];  // pierdere oraş l(4, j) în loc de l(2, j);
     else                            // nu sunt legate oraşele l(1, i) şi l(3, j);
      pierd[j] = d[2][j] - d[1][j];  // pierdere oraş l(3, j) în loc de l(2, j);
     pref[j] = l[1][j];              // preferatul lui j este oraşul l(2, j);
    }                                // terminare calcule l(1, i) acelaşi cu l(1, j);
  else                               // oraşele l(1, i) şi l(1, j) diferite;
   if(l[1][i] == l[0][j])            // oraşul l(2, i) acelaşi cu oraşul l(1, j);
    if(l[0][i] == l[1][j])           // oraşul l(1, i) acelaşi cu oraşul l(2, j);
     if(l[1][i] == cap[a])           // oraşele l(1, i) şi l(2, i) sunt legate;
      if(d[2][i] - d[0][i] < d[2][j] - d[0][j])     // pierderea din i mai mică;
      {pierd[i] = d[3][i] - d[2][i]; pref[i] = l[2][i];}   // schimbare în i;
      else                                          // pierderea din j mai mică;
      {pierd[j] = d[3][j] - d[2][j]; pref[j] = l[2][j];}   // schimbare în j;
     else                            // oraşele l(1, i) şi l(2, i) nu sunt legate;
     {pierd[i] = d[2][i] - d[0][i];  // pierdere oraş l(3, i) în loc de l(1, i);
      if(l[2][i] == cap[b])          // oraşele l(2, i) şi l(3, i) sunt legate;
       pierd[i] = d[3][i] - d[0][i]; // pierdere oraş l(4, i) în loc de l(1, i);
      if(l[2][j] == cap[k])          // oraşele l(2, j) şi l(3, j) sunt legate;
       pierd[j] = d[3][j] - d[0][j]; // pierdere oraş l(4, j) în loc de l(1, j);
      else                           // oraşele l(2, j) şi l(3, j) nu sunt legate;
       pierd[j] = d[2][j] - d[0][j]; // pierdere oraş l(3, j) în loc de l(1, j);
     }                // terminare calcule pentru oraşele l(1, i) şi l(2, i) nelegate
    else                             // oraşul l(1, i) diferit de oraşul l(2, j);
     if(l[1][i] == cap[a])           // oraşele l(1, i) şi l(2, i) sunt legate;
      if(d[2][i] - d[0][i] < pierd[j])              // pierderea din i mai mică;
      {pierd[i] = d[3][i] - d[2][i]; pref[i] = l[2][i];}   // schimbare în i;
      else                                          // pierderea din j mai mică;
      {pierd[j] = d[2][j] - d[1][j]; pref[j] = l[1][j];}   // schimbare în j;
```

https://biblioteca-digitala.ro / https://unibuc.ro

```
    else                                      // oraşele l(1, i) şi l(2, i) nu sunt legate;
     if(l[2][i] == cap[b])                    // oraşele l(2, i) şi l(3, i) sunt legate;
      pierd[i] = d[3][i] - d[0][i];           // pierdere oraş l(4, i) în loc de l(1, i);
     else                                     // oraşele l(2, i) şi l(3, i) nu sunt legate;
      pierd[i] = d[2][i] - d[0][i];           // pierdere oraş l(3, i) în loc de l(1, i);
   else                                       // oraşul l(2, i) diferit de oraşul l(1, j);
    if(l[0][i] == l[1][j])                    // oraşul l(1, i) acelaşi cu oraşul l(2, j);
     if(l[0][j] == cap[k])                    // oraşele l(1, j) şi l(2, j) sunt legate;
      if(d[2][j] - d[0][j] > pierd[i])                      // pierderea din i mai mică;
      {pierd[i] = d[2][i] - d[1][i]; pref[i] = l[1][i];}    // schimbare în i;
      else                                                  // pierderea din j mai mică;
      {pierd[j] = d[3][j] - d[2][j]; pref[j] = l[2][j];}    // schimbare în j;
     else                                     // oraşele l(1, j) şi l(2, j) nu sunt legate;
      if(l[2][j] == cap[k])                   // oraşele l(2, j) şi l(3, j) sunt legate;
       pierd[j] = d[3][j] - d[0][j];          // pierdere oraş l(4, j) în loc de l(1, j);
      else                                    // oraşele l(2, j) şi l(3, j) nu sunt legate;
       pierd[j] = d[2][j] - d[0][j];          // pierdere oraş l(3, j) în loc de l(1, j);
    else                                      // oraşul l(1, i) diferit de oraşul l(2, j);
     if(l[1][i] == l[1][j])                   // oraşul l(2, i) acelaşi cu oraşul l(2, j);
      if(l[1][i] == cap[a])                   // oraşele l(1, i) şi l(2, i) sunt legate,
       if(l[0][i] == l[2][j])                 // oraşul l(1, i) acelaşi cu oraşul l(3, j);
        pierd[j] = d[3][j] - d[0][j];         // pierdere oraş l(4, j) în loc de l(1, j);
       else                                   // oraşele l(1, i) şi l(3, j) sunt diferite;
        pierd[j] = d[2][j] - d[0][j];         // pierdere oraş l(3, j) în loc de l(1, j);
      else                                    // oraşele l(1, i) şi l(2, i) nu sunt legate;
       if(l[0][j] == cap[k])                  // oraşele l(1, j) şi l(1, j) sunt legate;
        if(l[2][i] == l[0][j])                // oraşele l(3, i) şi l(1, j) coincid;
         pierd[i] = d[3][i] - d[0][i];        // pierdere oraş l(4, i) în loc de l(1, i);
        else                                  // oraşele l(3, i) şi l(1, j) diferite;
         pierd[i] = d[2][i] - d[0][i];        // pierdere oraş l(3, i) în loc de l(1, i);
  }                         // terminare tratare coordonare cele două capete i şi j;
 }                          // terminare calcule pentru oraş i cu o legătură deja stabilită;
}                                             // terminare ciclul pentru parcurgere oraşe;
}                           // terminare funcţie de calcul pierderi şi preferinţe d  egare.
```

https://biblioteca-digitala.ro / https://unibuc.ro

4.1.4.3. Folosirea distanțelor ajustate

La stabilirea pierderilor, în loc să se țină seama de situația unui singur oraş, se pot avea în vedere perechile de oraşe. Aceasta deoarece, dacă oraşul j este cel mai apropiat de oraşul i, s-ar putea ca oraşul i să nu fie cel mai apropiat de oraşul j, existând un alt oraş k mai aproape de j. Aceasta implică, în cazul legării lui i cu j, să fie favorabil pentru i dar nu pentru j.

Pentru stăpânirea fenomenului menționat, se poate utiliza noțiunea de *distanță ajustată* dintre două oraşe care este dată de expresia:

$$da(i,j) = 2\, d(i,j) - d1(i) - d1(j),$$

unde $d1(i)$, respectiv $d1(j)$, reprezintă distanțele oraşului i, respectiv j, la cel mai apropiat oraş de i, respectiv de j şi $d(i, j)$ este distanța dintre i şi j. Utilizând distanța ajustată în locul distanței obişnuite în calculul pierderilor simple sau compuse, se pot construi alte euristici numite *cu pierderi simple ajustate*, respectiv *cu pierderi compuse ajustate*, în a doua dintre ele fiind considerate construcțiile ce evită inconsistența logică. De multe ori, euristicile ce folosesc distanța ajustată dau rezultate mai bune decât euristicile cu distanțe obişnuite.

4.1.4.4. Euristici mai rapide

Euristicile bazate pe pierderi pot fi uşor modificate pentru a reduce timpul necesar pentru calculul pierderilor în modul următor. În loc să se recalculeze pierderile la fiecare realizare a unei legături, se pot folosi aceleaşi pierde , până la apariția realizării unei legături pentru care pierderea este influențată de legăturile realizate între timp, caz în care este necesară recalcularea pierderilor. Mai precis, este necesară recalcularea în momentul când se face tentativa de a lega oraşul i cu j dacă:

1. Modificările efecuate afectează cel puțin una din valorile *pierd*(i) sau *pierd*(j) produse eventual de celelalte capete cu care ele sunt legate.
2. Cel puțin unul din cele două oraşe este candidat la o legătură realizată sau nu.
3. Cel puțin unul din cele două oraşe a avut legatătură cu un nod capăt care a participat la o legătură de la ultima recalculare a pierderilor.

Eficiența euristicilor obținute este comparabilă cu cea a celor din care provin din punct de vedere al lungimii turului determinat. Avantajul este că numărul de calcule scade simțitor, mai ales în cazul unui număr mare de oraşe, de foarte multe ori fiind necesară numai recalcularea a 2-4 pierderi la majoritatea legăturilor. În schimb apare un efort ceva mai mare la programare.

https://biblioteca-digitala.ro / https://unibuc.ro

4.1.5. Metoda ciclurilor crescătoare

Algoritmul următor se bazează pe ideea construirii unor cicluri cu un număr din ce în ce mai mare de oraşe până când sunt cuprinse toate cele n oraşe.

Algoritmul 4.5. (*Cicluri crescătoare*)

1. [Iniţializare] Se alege o pereche oarecare de oraşe şi se listează arbitrar în vederea generării permutării aciclice (i_1, i_2) de lungime doi; se face $k = 2$.
2. [Terminare] Dacă $k = n$, atunci STOP.
3. [Alegere] Se alege un oraş h dintre cele nealese încă şi, pentru $j = 1, 2, ..., k$, se calculează $d_j = a_{i_j,h} + a_{h,i_{j+1}} - a_{i_j,i_{j+1}}$, unde prin definiţie $i_{k+1} = i_1$ dacă $j = k$.
4. [Localizare] Fie j^* o valoare a lui j pentru care d_{j^*} este minimul cantităţilor calculate la pasul 3.
5. [Inserare] Se reetichetează i_j cu i_{j+1} pentru $j = j^*, ..., k$ şi se etichetează h cu i_{j^*}.
6. [Ciclare] Se face $k = k + 1$ şi se trece la pasul 2. ■

Prin $a_{i,j}$ am notat distanţa de la oraşul i la oraşul j. Ordinea de alegere a oraşelor din pasul 3 poate să influenţeze calitatea soluţiei finale.

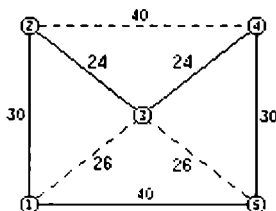


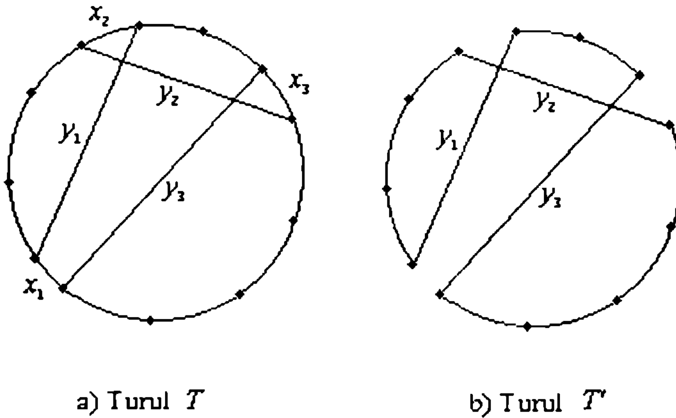
Fig. 4.4.

Exemplul 4.4. Se consideră graful din figura 4.4, în care turul optimal 1 2 3 4 5 este desenat cu linii pline. Acest tur are lungimea 148. Un alt tur identificat de algoritmul 4.5. este 1 2 4 5 3 de lungime 152. ■

4.1.6. Rectificarea turului

Fie S mulţimea tuturor muchiilor (legăturilor), în număr de n $(n - 1) / 2$, dintre cele n localităţi, T o submulţime de n muchii ce formează un tur de lungime $f(T)$ şi T' un tur de lungime $f(T') < f(T)$. Să presupunem că T şi T' diferă, ca mulţimi de n muchii, prin k muchii. În cadrul algoritmului de bază, se încearcă transformarea lui T în T' pe calea identificării succesive, secvenţiale, a k perechi de muchii ce urmează a fi schimbate între T şi $S - T$. Prin urmare, se încearcă identificarea a două mulţimi de muchii $x = \{x_1, ..., x_k\}$ şi $y = \{y_1, ..., y_k\}$ astfel încât, dacă muchiile din x sunt eliminate şi înlocuite cu muchiile din y, se obţine un tur de cost mai mic.

https://biblioteca-digitala.ro / https://unibuc.ro



a) Turul T b) Turul T'

Fig. 4.5.

Un exemplu ilustrativ este arătat pe figura 4.5. pentru k = 3. Numerotările muchiilor satisfac condiția că o pereche de muchii x_i şi y_i sau y_i şi x_{i+1} au un nod comun, cu $x_{k+1} = x_1$. În mod frecvent, numerotarea precedentă poate fi efectuată şi drept rezultat trecerea de la T la T' se realizează în mod secvențial. Un exemplu în care numerotarea precedentă nu poate fi efectuată este arătat în figura 4.6.

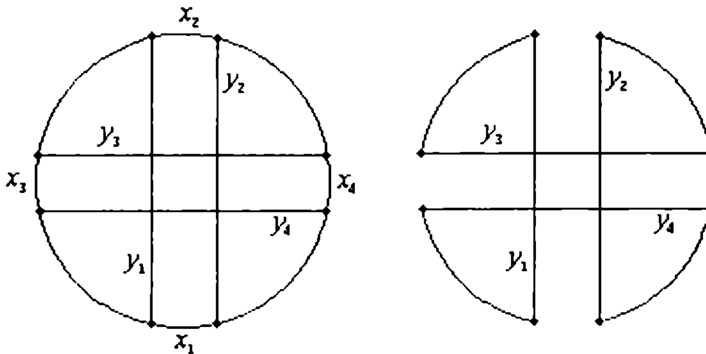


Fig. 4.6.

Presupunând că enumerarea muchiilor, în modul descris mai înainte, poate fi efectuată, succesiunea schimbărilor căutate este x_1, y_1; x_2, y_2; ... etc. Problema constă în identificarea unei secvențe care reduce pe T la T' cu condiția $f(T') < f(T)$, unde f reprezintă costul turului, urmată de iterarea procesului pentru T' până când nu se mai poate realiza o reducere.

Fie $|x_i|$ şi $|y_i|$ lungimile muchiilor x_i şi y_i şi $g_i = |x_i| - |y_i|$, câştigul obţinut prin schimbarea lui x_i cu y_i. Chiar dacă unele valori g_i sunt negative, dacă $f(T') < f(T)$, $\sum_{1}^{k} g_i = f(T) - f(T') > 0$. O parte din procedura propusă se bazează pe următoarea proprietate:

Lema 4.1. Dacă o secvență de numere au o sumă pozitivă, atunci există o permutare a acestor numere astfel încât *fiecare sumă parțială* este pozitivă.

Demonstrație: Fie k cel mai mare indice pentru care $g_1 + g_2 + ...+ g_{k-1}$ este minimă. Dacă $k \le j \le n$, atunci:

$$g_k + ... + g_j = (g_1 + ... + g_j) - (g_1 + ... + g_{k-1}) > 0.$$

Dacă $1 \le j < k$, atunci:

$$g_k + ... + g_n + g_1 + ... + g_j \ge g_k + ... + g_n + g_1 + ... + g_{k-1} > 0. \blacksquare$$

În concluzie, deoarece se caută secvențe de valori g_i ce au o sumă pozitivă, ***este suficient să se considere numai secvențe de câștiguri a căror sumă parțială este totdeauna pozitivă.*** Acest criteriu de câștig permite reducerea esențială a **numărului** de secvențe ce urmează a fi examinate.

Algoritmul euristic următor folosește notațiile din figura 4.7.

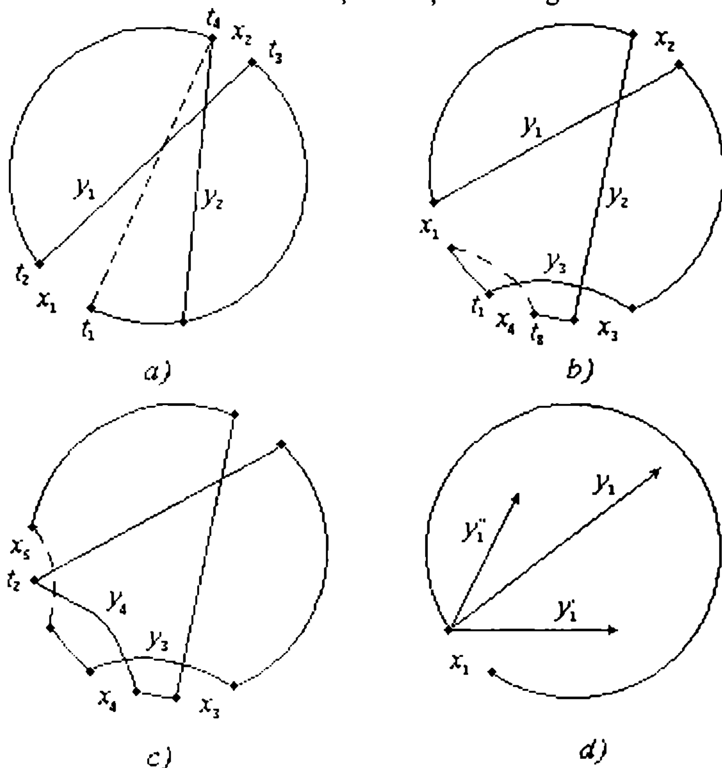


Fig. 4.7.

Algoritmul 4.6. (*Rectificare*)

1. [Inițializare] Se generează aleator un tur inițial T.
2. [Start rectificare] Fie $G^* = 0$ (G^* reprezintă cea mai bună ameliorare obținută anterior). Se alege un nod t_1 și o muchie x_1 din T adiacentă lui t_1. Se face $i = 1$.

https://biblioteca-digitala.ro / https://unibuc.ro

3. [Identificare pereche] Plecând de la cealaltă extremitate t_2 a lui x_1, se identifică o muchie y_1 conectată la t_3 cu $g_1 > 0$. Dacă nu există un asemenea y_1, se trece la pasul 6a. (Aceasta este prima aplicație a criteriului de câștig.).
4. [Extindere] Se face $i = i + 1$. Se caută un x_i care unește nodul t_{2i-1} cu nodul t_{2i} și un y_i care unește nodul t_{2i} cu nodul t_{2i+1} după cum urmează:
   a. Se identifică x_i astfel încât, dacă t_{2i} este conectat la t_1, configurația obținută este un tur. În acest fel, pentru un y_{i-1} dat, x_i este determinat în mod unic.
   b. Fie y_i o legătură accesibilă la nodul final t_{2i} al muchiei x_i și satisfăcând condițiile c., d. și e.. Dacă nu există un asemenea y_i, se trece la pasul 5.
   c. Pentru a garanta că mulțimile x și y sunt disjuncte, x_i nu poate fi o legătură anterior realizată (adică $y_j, j < i$) și, în mod similar, y_i nu poate fi o legătură anterior eliminată.
   d. Se calculează $G_i = \sum_{1}^{i} g_j > 0$ (Criteriul câștigului).
   e. În vederea asigurării criteriului realizabilității din a. pentru $i + 1$, muchia y_i aleasă trebuie să asigure eliminarea unei muchii x_{i+1}.
   f. Înainte ca y_i să fie construit, se testează dacă încheierea căutării prin conectarea lui t_{2i} la t_1 dă un câștig mai bun decât cel mai bun observat anterior (criteriul realizabilității a fost satisfăcut pentru $i \geq 2$, deci conexiunea lui t_{2i} la t_1 furnizează un tur). Fie y_i^* legătura între t_{2i} și t_1 și $g_i^* = |y_i^*| - |x_i|$. Dacă $G_{i-1} + g_i^* > G^*$, atunci se face $G^* = G_{i-1} + g_i^*$ și $k = i$.
5. [Reluare extindere] Se încheie construirea lui x_i și y_i din pașii 2 - 4 dacă nici o muchie nu mai satisface condițiile 4.c.-4.e. sau dacă $G_i \leq G^*$. Dacă $G^* > 0$, atunci se consideră turul T' cu $f(T') = f(T) - G^*$ și se repetă întregul proces începând cu pasul 2, folosind pe T' drept tur inițial.
6. [Întoarcere] Dacă $G^* = 0$, atunci se utilizează un backtracking limitat după cum urmează:
   a. Se repetă pașii 4 și 5 și se identifică muchii y_2 în ordinea crescătoare a lungimii cât timp este satisfăcut criteriul câștigului $g_1 + g_2 > 0$. Dacă apare o îmbunătățire, atunci se trece la pasul 2.
   b. Dacă toate căutările lui y_2 în pasul 4.b. nu furnizează un profit, tunci se reia de la pasul 4.a. și se încearcă altă alegere pentru x_2.
   c. Dacă nu s-a obținut o ameliorare, atunci se reia pasul 3 examinând muchiile y_1 în ordinea crescătoare a lungimilor (vezi figura 4.7.d).
   d. Dacă și muchiile y_1 au fost parcurse fără un profit, atunci se reia pasul 2 pentru alte muchii x_1.

https://biblioteca-digitala.ro / https://unibuc.ro

e. Dacă nu s-a identificat o nouă muchie x_1, atunci se selectează un nou t_1 şi se reia de la pasul 2.

7. Procedura se încheie după ce toate cele n valori ale lui t_1 au fost prelucrate fără a furniza un profit. În continuare se pot considera alte tururi aleatoare în pasul 1. ■

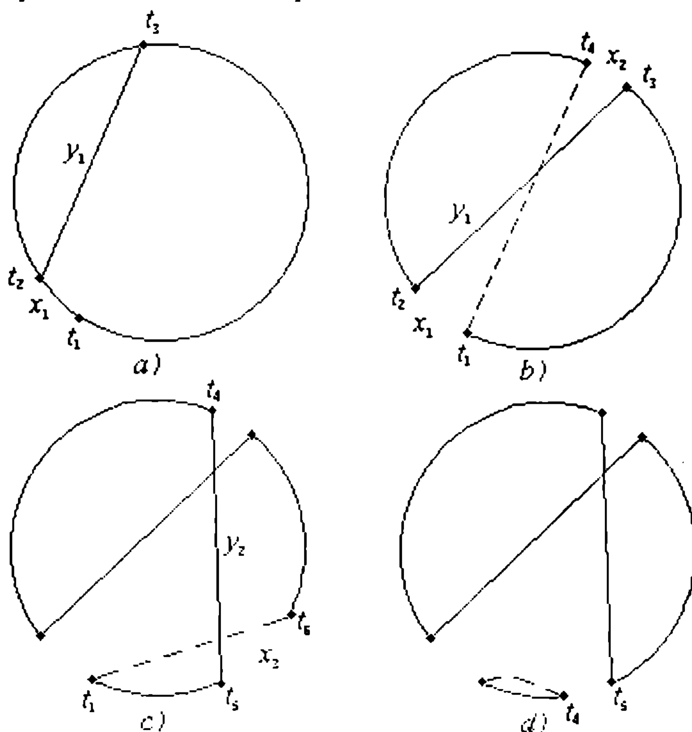


Fig. 4.8.

Dăm în continuare câteva informaţii suplimentare privind acest algoritm. În pasul 4.a. este aplicabil criteriul *realizabilităţii* care *garantează* că întotdeauna este posibilă obţinerea unui tur conectând pe t_{2i} cu t_1, pentru orice $i \geq 2$. Alegerea lui y_{i-1} în conformitate cu pasul 4.e. asigură existenţa întotdeauna a unui unic astfel de x_{i-1}. Un exemplu ilustrativ este arătat în figura 4.7.a). Evident că pentru asigurarea unei reduceri mari a costului, $|y_i|$ din pasul 4.b. trebuie să fie mic, aşa încât, în general, se preferă cel mai apropiat vecin. Figura 4.7.b) ilustrează cazul $i = 4$, cu observaţia că t_8 este unic. Un caz ilustrativ privind pasul 4.c. pentru $i = 5$ este arătat în figura 4.7.c), cu menţiunea că ambele muchii originale conectate la t_5 au fost eliminate. În pasul 4.f. G^* este întotdeauna cea mai bună ameliorare a lui T construit anterior şi constituie un standard de comparaţie. El are proprietăţile: $G^* \geq 0$ şi este monoton descrescător. Valoarea lui k defineşte mulţimile ce urmează a fi schimbate pentru a obţine G^*.

https://biblioteca-digitala.ro / https://unibuc.ro

De notat că backtracking-ul indicat în pasul 6 se efectuează *numai* dacă nu s-a obţinut un câştig şi *numai* de la nivelele 1 şi 2 ($i = 1$ şi $i = 2$).

Exemplul 4.3. Fie turul T din figura 4.8.a), două noduri adiacente t_1 şi t_2 şi x_1 muchia ce le uneşte. Fie t_3 cel mai apropiat nod de t_2; y_1 este muchia (t_2, t_3). Deoarece mulţimile x şi y trebuie să fie disjuncte, y_1 nu poate fi o muchie conectată la t_2. Se calculează $g = |x_1| - |y_1|$; dacă g nu este pozitiv, se efectuează o reluare (pasul 4.d.) şi fie t_2 un alt nod vecin al lui t_1. Acum $i = 2$ şi fie t_4 vecinul lui t_3 în modul ilustrat pe figura 4.8.b) şi x_2 muchia (t_3, t_4). Dacă y_2 este ales pentru a conecta pe t_4 cu t_1, atunci se obţine un tur şi dacă $g_1 + g_2 > 0$, atunci turul T poate fi îmbunătăţit schimbând pe x_1 şi x_2 respectiv cu y_1 şi y_2. Se memorează în G^*, îmbunătăţirea potenţială obţinută. Apoi $k = 2$ în pasul 4.f.. Se caută cel mai apropiat vecin t_5 al lui t_4 şi fie y_2 muchia (t_4, t_5). Din nou, t_5 nu poate fi unul dintre nodurile conectate anterior la t_4.

După cum se vede în figura 4.8.c), nu este posibilă decât o singură alegere pentru t_6 şi t_3 pentru obţinerea unui tur. Dacă x_3 este cealaltă muchie conectată la t_5, atunci se obţin două tururi distincte, arătate în figura 4.8.d).

Procesul de identificare a unui tur mai bun continuă până este satisfăcut criteriul de încheiere din pasul 5. ■

4.2. Problema restrictivă a comisvoiajorului

Punerea problemei. Fie un graf direcţionat complet $G = (V, A)$ cu mulţimea $V = \{v_1, v_2, ..., v_n\}$ de n noduri şi mulţimea A a arcelor (v_i, v_j), fiecare având costul $c_{i,j}$. Problema restrictivă a comisvoiajorului (*PRC*) constă în identificarea unui ciclu hamiltonian, definit ca mulţime ordonată de noduri

$$H = (v_{h_1}, v_{h_2}, ..., v_{h_n}, v_{h_{n+1}} \equiv v_{h_1}), \tag{4.1}$$

astfel încât:

$$z = \max\{c_{h_i, h_{i+1}} \mid 1 \le i \le n\} \tag{4.2}$$

să fie minim.

Fără a micşora generalitatea, se presupune că $n > 2$, $c_{i,j}$ sunt întregi pozitivi, pentru toate arcele $(v_i, v_j) \in A$, şi $c_{i,i} = +\infty$, pentru toate nodurile $v_i \in V$. ■

Problema *PRC* este NP-completă. Pentru obţinerea soluţiei se poate elabora relativ uşor un algoritm *branch and bound* cu căutare în lăţime.

Vom indica în continuare mai multe margini inferioare pentru *PRC*.

Problema asignării restrictive. Problema asignării restrictive (*PAR*) constă în identificarea unei permutări (a_i) a întregilor 1, 2, ..., n care minimizează pe $z^a = \max\{c_{i,a_i} \mid 1 \le i \le n\}$. Dacă (a_i) este un ciclu hamiltonian, atunci $z = z^a$. Dacă (a_i)

https://biblioteca-digitala.ro / https://unibuc.ro

defineşte m subtururi ($m > 1$) pe G, atunci z^u este o margine inferioară pentru z. Soluţia PAR necesită $O(n^3)$ operaţii [LAW76].

Drumuri restrictive. Fiind dat $G = (V,A)$, un drum restrictiv (DR) de la nodul v_i la nodul distinct v_j este un drum $P_{i,j} = (v_i \equiv v_{p_1}, v_{p_2}, \ldots, v_{p_k} \equiv v_j)$ astfel încât este minimizată expresia $z_{i,j}^P = \max\{c_{p_k, p_{k+1}} \mid 1 \le k \le t - 1\}$. Fie toate DR dintre toate perechile ordonate ale nodurilor din V. Atunci:

$$z^P = \max_{\substack{i=1,\ldots,n \\ j=1,\ldots,n, j\neq i}} \left\{z_{i,j}^P\right\} \tag{4.3}$$

este margine inferioară pentru z. Într-adevăr, din (4.1) şi (4.2) rezultă că ciclul hamiltonian care rezolvă PRC conţine un drum de la orice nod v_i la un alt nod v_j astfel încât costul maxim al arcelor sale nu depăşeşte pe z. Rezultă că $z_{i,j}^P \le z$, pentru toţi i, j. Un mod eficient pentru calculul lui z^P este dat de următoarea teoremă.

Teorema 4.1. Pentru orice nod $v_r \in V$, valoarea lui:

$$\bar{z} = \max_{j=1,\ldots,n, j\neq r} \left\{z_{r,j}^P, z_{j,r}^P\right\} \tag{4.4}$$

coincide cu valoarea lui z^P.

Demonstraţie. Din (4.3) şi (4.4) rezultă că $z^P \ge \bar{z}$. Urmează să se arate că $z^P \le \bar{z}$. Pentru fiecare pereche de noduri distincte v_s, $v_t \in V$,

a) dacă $s = r$ sau $t = r$, atunci din (4.4) rezultă că $z_{s,t} \le \bar{z}$;

b) dacă $s \neq r$ şi $t \neq r$, atunci un drum de la v_s la v_t poate fi obţinut unind un DR de la v_s la v_r cu un DR de la v_r la v_t. Din (4.4) rezultă că $z_{s,t}^P \le \max\{z_{s,r}^P, z_{r,t}^P\} \le \bar{z}$. ■

Dacă pentru fiecare nod $v_i \in V$ se definesc

$$z_i^P = \max\{z_{i,j}^P \mid 1 \le j \le n, j \neq i\},$$

$$\hat{z}_i^P = \max\{z_{j,i}^P \mid 1 \le j \le n, j \neq i\},$$

atunci, din teorema 4.1., rezultă că z^P poate fi obţinută dacă pentru orice nod $v_r \in V$ se calculează $z^P = \max\{z_r^P, \hat{z}_r^P\}$.

Se poate arată că z^P poate fi calculat astfel în $O(n^2)$ operaţii.

Înlănţuirea nodurilor. Fie nodul oarecare v_i şi valorile $c_{i,t}=\min\{c_{i,j} \mid 1 \le j \le n\}$, $c_{u,i} = \min\{c_{j,i} \mid 1 \le j \le n\}$. Deoarece în orice circuit hamiltonian un arc trebuie să iasă din v_i şi un altul trebuie să intre în v_i, rezultă că $b_i = \max\{c_{i,t}, c_{u,i}\}$ este evident o margine inferioară pentru z. Mai mult, dacă $v_t \equiv v_u$, caz fals în circuit hamiltonian, se poate îmbunătăţi marginea inferioară calculând pe $c_{i,t'} = \min\{c_{i,j} \mid 1 \le j \le n$ şi $j \neq t\}$ şi pe $c_{u',i} = \min\{c_{j,i} \mid 1 \le j \le n$ şi $j \neq u\}$. O margine mai bună este dată de:

$$b_i' = \begin{cases} b_i & \text{dacă } v_t \neq v_u, \\ \max\left\{b_i, \min\left\{c_{i,t'}, c_{u',i}\right\}\right\} & \text{dacă } v_t = v_u. \end{cases}$$

https://biblioteca-digitala.ro / https://unibuc.ro

Deoarece acest rezultat are loc pentru orice $v_i \in V$, rezultă că $z^v = \max_{v_i \in V}\{b_i'\}$ este o margine inferioară pentru z. Calculul lui b_i' necesită evident O(n) operații, așa încât z^v poate fi calculat în O(n^2) operații.

Înlănțuirea subtururilor. Dacă problema *PAR* a fost rezolvată, atunci o margine inferioară poate fi calculată într-un mod analog cu cel descris anterior. Fie $S = \{S_1, S_2, ..., S_m\}$ partiția mulțimii V definită de m subtururi obținute drept rezultat al rezolvării *PAR*, cu $m > 1$, și fie subturul ce corespunde lui S_k. Deoarece în orice circuit hamiltonian măcar un arc trebuie să iasă din S_k și măcar unul trebuie să intre în S_k, o margine inferioară pentru z este $d_k = \max\{c_{s,t}, c_{u,q}\}$, unde $c_{s,t} = \min\{c_{i,j} \mid v_i \in S_k, v_j \in V\text{-}S_k\}$ și $c_{u,q} = \min\{c_{j,i} \mid v_i \in S_k, v_j \in V - S_k\}$. În acest caz se impune de asemenea ca $v_s \neq v_q$ și $v_t \neq v_u$, adică se caută o pereche de arce, unul care intră și altul care iese, fără o extremitate comună. O margine mai bună este dată de:

$$d_k' = \min\left\{\max\left\{c_{s',t'}, c_{u',q'}\right\} \middle| v_{q'}, v_{s'} \in S_k, v_{q'} \neq v_{s'}; v_{t'}, v_{u'} \in V - S_k, v_{t'} \neq v_{u'}\right\}.$$

Valoarea lui d_k' se poate calcula pentru toate subtururile din S, așa că marginea inferioară pentru z este dată de:

$$z^S(S) = \max\{d_k' \mid 1 \leq k \leq m\}.$$

Se poate construi o procedură care calculează pe $z^S(S)$ în O($n^2 \log n$) operații.

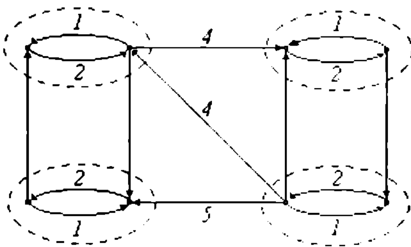


Fig. 4.9.

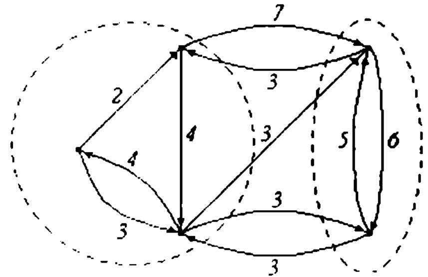


Fig. 4.10.

O margine mai bună poate fi obținută dacă numărul de subtururi furnizat de *PAR* este redus în mod euristic pe calea conectării perechilor de subtururi prin arce a căror cost nu-i mai mare decât orice margine inferioară. În cazul grafului din figura 4.9., în care liniile punctate notează soluția *PAR*, marginile inferioare sunt z^u=2, z^v=3, z^p=4. Pentru cele patru subtururi furnizate de *PAR*, procedura descrisă furnizează marginea z^s=3. Dacă însă S este modificată pe calea conectării celor două subtururi din stânga și din dreapta, pe calea înlocuirii arcelor de cost 1 cu arce de cost 3, rezultă z^s=5.

Trebuie menționat că nici una dintre marginile prezentate nu le domină pe celelalte. În graful din figura 4.9., $z^u < z^v < z^p < z^S(S)$ în timp ce în graful din figura 4.10., $z^u = 6 > z^v = 5 > z^p = 4 > z^S(S) = 3$.

https://biblioteca-digitala.ro / https://unibuc.ro

Algoritmul propus pentru *PRC* foloseşte o schemă de ramificare prezentată în [CAR80]. La fiecare nod al arborelui de decizie, algoritmul parcurge procedura de mărginire prezentată mai jos. Această procedură se bazează pe marginile anterioare şi căutarea euristică a unui circuit hamiltonian.

Fiind date graful $G = (V, A)$ şi o valoare $\bar{z}$ a marginii inferioare, se defineşte submulţimea A: $A(\bar{z})=\left\{(v_i,v_j)\middle|(v_i,v_j)\in A \text{ şi } c_{i,j}\le\bar{z}\right\}$. În continuare, se aplică o procedură euristică pentru identificarea unui circuit hamiltonian în graful parţial $\overline{G}=(V,A(\bar{z}))$. În particular, se poate folosi algoritmul enumerativ din [MAR83] care, fiind parcurs pe calea limitării numărului de iteraţii, are trei ieşiri posibile:

1) a fost găsit un circuit hamiltonian,
2) nu există nici un circuit hamiltonian şi
3) nu a fost găsit un circuit hamiltonian.

Procedura de mărginire. Fie z^* valoarea curentă a unei soluţii optimale.

Faza 1. Fie $\bar{z} = \max\{z^{r}, z^{p}, z^{a}\}$ şi se memorează în S soluţia corespunzătoare lui z^a. Dacă $\bar{z} \ge z^*$ sau S este un circuit hamiltonian, atunci STOP.

Faza 2. Se execută procedura euristică pentru găsirea unui circuit hamiltonian în $\overline{G}=(V,A(\bar{z}))$. Dacă a fost găsit un asemenea circuit, atunci STOP. Dacă nu există un circuit hamiltonian, atunci se pune $\bar{z}=\min\left\{c_{i,j}\middle|(v_i,v_j)\in A-A(\bar{z})\right\}$ şi dacă $\bar{z}\ge z^*$, atunci STOP. Se modifică S în vederea reducerii numărului de subtururi. Dacă S este un circuit hamiltonian, atunci STOP. Se pune $\bar{z}=\max\left\{\bar{z},z^S(S)\right\}$. Dacă $\bar{z}\ge z^*$, atunci STOP. Dacă execuţia fazei 2 a îmbunătăţit marginea inferioară $\bar{z}$, atunci se iterează faza 2, altfel STOP.

Dacă $\bar{z}\ge z^*$ la încheierea procedurii de mărginire, atunci algoritmul trece în backtracking. Altfel, dacă S este un circuit hamiltonian, atunci soluţia optimală este reactualizată şi se trece în backtracking. Dacă S nu este un circuit hamiltonian, atunci se trece la subturexcepţia lui S. ■

4.3. Problema comisvoiajorului cu *k* persoane

Fie un graf complet $G = (V, E)$ conţinând n noduri $v_1, v_2, ..., v_n$. Pentru un k întreg pozitiv, un k-tur H reprezintă o mulţime de k subtururi notate $H_1, ..., H_k$, unde:

1. H_i este un ciclu simplu conţinând cel puţin 3 arce, pentru $i = 1, 2, ..., k$.
2. H_i conţine nodul 1, pentru $i = 1, 2, ..., k$.
3. Pentru fiecare $v \in V\setminus\{v_1\}$ există un singur subtur H_i ce trece prin v.

https://biblioteca-digitala.ro / https://unibuc.ro

Se presupune că cei k comisvoiajori pleacă fiecare din nodul 1 şi vizitează mulţimea de noduri, care cuprinde toate nodurile din $V \setminus \{v_1\}$. Funcţia distanţă $d: E \rightarrow R$ satisface condiţia:

$$d(u, v) + d(v, w) \geq d(u, w), \text{ pentru toţi } u, v, w \in V.$$

Un k-tur H are lungimea $d(H)=\sum_{i=1}^{k} d(H_i)$, unde pentru orice $S \subset E$, $d(S)= \sum_{e \in S} d(e)$.

Problema de rezolvat constă în identificarea unui k-tur de lungime minimă.

Euristica propusă reprezintă o generalizare naturală a euristicii Christofides şi are aproape aceeaşi performanţă.

Algoritmul 4.7. (*k-tururi*)

1. [Arbore parţial] Se identifică un arbore maximal T_0 al lui G de lungime minimă printre arborii ce au $2k$ arce incidente cu nodul v_1. (În lucrarea [GLO7.. se arată că această problemă poate fi rezolvată într-un timp $O(|V^2|)$).
2. [Împerechere] Se determină mulţimea de noduri $X_0 \subseteq T$ de grad impar. Se realizează o împerechere M_0 perfectă de lungime minimă în subgraful G indus de X_0 ($|X|$ este mereu par şi problema împerecherii este rezolvată în timp de $O(|X|^3)$ [LAW76]).
3. [Ciclu eulerian] Graful $G_0 = (V, T_0 \cup M_0)$ este conex şi fiecare nod al său are grad par. Se construieşte un ciclu eulerian EC_0 al lui G_0, adică un ciclu ce conţine o singură dată fiecare arc al lui G_0. (EC_0 poate fi construit în timp liniar).
4. [Reducere] Ciclul EC_0 este în continuare redus la un k-tur H_0. Fie U mulţimea nodurilor adiacente la v_1 în T_0. Presupunem că EC_0 cuprinde secvenţa de noduri $v_1 = w_1, w_2, ..., w_m = v_1$. Parcurgând această secvenţă, un nod w_i se elimină dacă:
   a. $w_i \neq v_i$ şi w_i a mai apărut anterior,
   b. $w_i \in U$ şi $v_1 \notin \{w_{i-1}, w_{i+1}\}$. ■

La încheierea pasului 4, secvenţa de noduri obţinută defineşte un k-tur H_0. Eliminările din a. asigură vizitarea o singură dată a unui nod $v \neq v_1$ şi elim irile din b. asigură prezenţa în orice subtur a măcar două noduri diferite de v_1.

Exemplul 4.5. În figura 4.11.a) este reprezentat subgraful unui graf complet G, arcele subgrafului având lungimea 1 şi arcele nefigurate (v, w) o lungime $d(v, w)$ egală cu cea mai scurtă distanţă de la v la w în subgraf. Astfel, distanţa între nodurile x şi y este egală cu 7. Arborele T_0 al grafului G este arătat cu linii pline în figura 4.11.b). Completând arborele T_0 cu cele două arce punctate, se obţine soluţia generată de euristica descrisă, pentru cazul $k = 2$. Ţinând seama că lungimile celor două arce punctate sunt egale cu 10, rezultă că suma lungimilor celor două subtururi este egală cu $2 \cdot (10 + 2 \cdot 9) = 56$. Soluţia optimală H^* este arătată pe figura 4.11.c), suma lungimilor celor două subturuti fiind în acest caz egală cu $2 \cdot (2 \cdot 9 + 4) = 44$, unde 4 reprezintă lungimile arcelor (1, 2) respectiv (3, 4). ■

https://biblioteca-digitala.ro / https://unibuc.ro

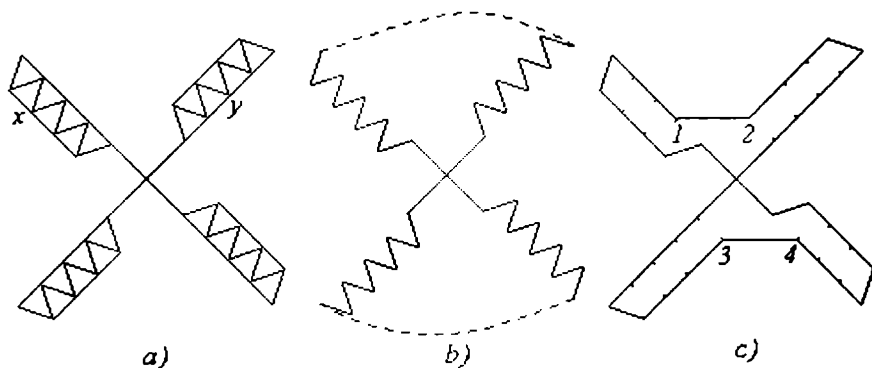


Fig. 4.11.

Se poate demonstra că:

$$d(H_0) \leq 3/2\ d(H^*),$$

considerând o generalizare a exemplului descris mai înainte. În exemplul 4.4., $d(H_0) = 56 < 3/2 \cdot 44 = 66 = 3/2\ d(H^*)$.

4.4. Localizarea stației de deservire în problema comisvoiajorului

Punerea problemei. Se consideră o rețea $G(N, L)$ unde N este o mulțime de n noduri și L mulțimea legăturilor (drumurilor). O singură unitate de deservire este localizată într-un centru X al rețelei. În fiecare zi se înregistrează până la un termen t_0 mulțimea cererilor ce urmează să fie satisfăcute. Se presupune că din fiecare nod i poate apare zilnic cel mult o cerere, cu probabilitatea h_i. Se notează cu k numărul de cereri înregistrate până la momentul t_0. Prin *listă* se înțelege numărul de noduri înregistrate, într-un catalog, în cazul $k > 0$. Numărul total de liste posibile dintr-o anumită zi este 2^n-1. Se notează cu E mulțimea listelor posibile și cu R o listă dată din E.

Fie $T(R \cup X)$ turul optimal, adică turul cel mai scurt în problema comisvoiajorului, ce începe din centrul de deservire și trece prin fiecare nod din R *măcar o dată* și se încheie în centrul de deservire.

$$T(R \cup X) = X \to \pi(1) \to \pi(2) \to \dots \to \pi(j) \to \dots \to \pi(k) \to X,$$

unde $\pi(j)$ este al j-lea nod din $T(R \cup X)$. Fie apoi $L(R \cup X)$ lungimea turului $T(R \cup X)$:

$$L(R \cup X) = d(X, \pi(1)) + \sum_{j=1}^{k-1} d(\pi(j), \pi(j+1)) + d(\pi(k), X),$$

unde $d(z, y)$ este cea mai mică distanță (cel mai scurt drum) între nodurile $z, y \in G$. ■

Fie P_R probabilitatea ca unitatea de deservire să recepționeze lista R pentru un $k > 0$ dat și

https://biblioteca-digitala.ro / https://unibuc.ro

$$P_W = 1 - \prod_{i \in N}(1 - h_i)$$

probabilitatea ca lista să nu fie vidă. Rezultă că:

$$P_R = (1 / P_W)[\prod_{j \in R} h_j][\prod_{j \notin R}(1 - h_j)] = K \prod_{j \in R} h_j / (1 - h_j),$$

unde:

$$K = (1 / P_W) \prod_{j \in N}(1 - h_j).$$

Problema de rezolvat constă în identificarea unei locații X^* din $G(N, L)$ care să minimizeze lungimea $g(X)$ a turului de deservire:

$$X^* = \arg \min_X g(X) \equiv \arg \min_X \sum_{R \in E} P_R L(R \cup X),$$

unde cu $\arg \min_X$ s-a notat *argumentul X ce minimizează* și:

$$g(X) = \sum_{R \in E} P_R L(R \cup X),$$

$L(R \cup X)$ fiind lungimea turului optimal.

Aproximarea lungimii turului optimal. Deoarece timpul necesar calculului turului optimal are $O(2^{|R|})$ este de dorit folosirea unei aproximări a acestui [illegible]. Pentru aceasta se propune următoarea euristică:

1. Se identifică un tur optimal pentru submulțimea K de noduri de unde au fost efectuate cereri.
2. Se calculează $\min_{j \in R} d(X, j) \equiv d(X, F_R(X))$ care este cea mai mică distanță dintre X și cel mai apropiat nod de X din R, notat cu $F_R(X)$. Se conectează turul obținut la pasul 1 cu nodul X prin intermediul a două arce de lungime $d(X, F_R(X))$. Lungimea turului aproximativ este:

$$A(R \cup X) = L(R) + 2\, d(X, F_R(X)). \tag{4.5}$$

Ideea principală a euristicii constă în identificarea în locul lui X^* a unui vîrf $\tilde{X}$ care satisface condiția:

$$\tilde{X} = \arg \min_{X \in N} f(X) \equiv \arg \min_{X \in N} \sum_{R \in E} P_R A(R \cup X),$$

unde $f(X) = \sum_{R \in E} P_R A(R \cup X)$.

Utilizând (4.5), $f(X)$ se poate scrie sub forma:

$$f(X) = \sum_{R \in E} P_R A(R \cup X) = 2 \sum_{R \in E} P_R d(X, F_R(X)) + \sum_{R \in E} P_R L(R). \tag{4.6}$$

Deoarece ultimul termen din (4.6) este o constantă independentă de X, rezultă că:

$$\tilde{X} = \arg \min_{X \in N} \sum_{R \in E} P_R d(X, F_R(X)).$$

În continuare se arată că pentru un X dat, $\sum_{R \in E} P_R d(X, F_R(X))$ se poate calcula într-un mod eficient. Pentru fiecare $X \in N$ nodurile $j_1, j_2, \ldots, j_n$ din N se renumerotează

https://biblioteca-digitala.ro / https://unibuc.ro

astfel încât $d(X, j_k) \le d(X, j_l)$, pentru $l \ge k$ (unde nodul $X \equiv j_1$). Se defineşte E_{X,j_k}, $k \ne 1$, prin expresia:

$$E_{X,j_k} = \{R \in E; j_1, j_2, ..., j_{k-1} \notin R, j_k \in R\}.$$

Din definiţiile lui $d(X, F_R(X))$ şi $E(X, j_k)$ rezultă că:

$$d(X, F_R(X)) = d(X, j_k) \text{ dacă şi numai dacă } R \in E_{X,j_k}.$$

Deoarece atunci când $X = F_R(X)$, $P_R d(X, F_R(X)) = 0$, rezultă:

$$\sum_{R\in E} P_R\, d(X, F_R(X)) = \sum_{k=2}^{n} d(X, j_k) \sum_{R\in E_{X,j_k}} P_R.$$

În lema ce urmează este indicat modul în care se poate calcula $\sum_{R\in E_{X,j_k}} P_R$.

Lema 4.2. $\sum_{R\in E_{X,j_k}} P_R = h_{j_k} / P_W \prod_{m=1}^{k-1}(1 - h_{j_m})$, pentru $k \ge 2$.

Demonstraţie. Fie $\widetilde{R}$ o listă aleasă la întâmplare; atunci:

$$\sum_{R\in E_{X,j_k}} P_R = \text{Prob}\{\widetilde{R} \in E_{X,j_k} \mid k > 0\} = \text{Prob}\{j_1, j_2, ..., j_{k-1} \in \widetilde{R}, j_k \notin \widetilde{R} \mid k > 0\} =$$

$$= 1 / P_W (1 - h_{j_1})(1 - h_{j_2}) \dots (1 - h_{j_{k-1}})\, h_{j_k}. ■$$

Rezultă că:

$$\sum_{R\in E} P_R\, d(X, F_R(X)) = 1 / P_W \sum_{k=2}^{n} d(X, j_k)\, h_{j_k} \prod_{m=1}^{k-1}(1 - h_{j_m}). \quad (4.7)$$

Analiza celui mai defavorabil caz. În euristica ce urmează se calculează $\widetilde{X}$ în loc de X^* astfel: pentru fiecare nod $X \in N$ se evaluează expresia (4.7) şi se selectează $\widetilde{X}$ ca fiind nodul pentru care se realizează cea mai mică valoare.

Algoritmul 4.8. (Aproximare lungime tur)

1. [Drumuri minime] Se calculează matricea D a celor mai scurte distanţe.
2. [Iniţializare] Se face $m = \infty$.
3. [Ciclare noduri] Pentru orice nod $X \in N$ se execută paşii 4 - 7, apoi STOP.
4. [Renumerotare] Se renumerotează nodurile din N, $j_1, j_2, ..., j_n$ astfel încât $d(X, j_k) \le d(X, j_l)$, pentru toţi $l \ge k$ (nodul $X \equiv j_1$).
5. [Calcul ponderi] Puntru $k = 2, 3, ..., n$ se calculează $U_k = \prod_{m=1}^{k-1}(1 - h_{j_m})$.
6. [Ponderea totală] Se calculează $V = \sum_{k=2}^{n} d(X, j_k)\, U_k\, h_{j_k}$.
7. [Rectificare] Dacă $V < m$, atunci se face $m = V$ şi $\widetilde{X} = X$. ■

Pasul 1 are complexitatea $O(n^3)$ ([FLO62]). Paşi 4, 5, 6 şi 7 sunt parcurşi de n ori; fiecare execuţie a pasului 4 are complexitatea $O(n \log_2 n)$ şi paşii 5 şi 6 au complexitatea $O(n)$. Complexitatea totală este de $O(n^3)$. Dacă matricea D este dată, atunci complexitatea este $O(n^2 \log_2 n)$.

https://biblioteca-digitala.ro / https://unibuc.ro

Se poate demonstra proprietatea următoare:

Lema 4.3. $A(R \cup X) \leq 3/2\, L(R \cup X)$, pentru orice R. ■

Se poate vedea că marginea din lemă poate fi atinsă ca în exemplul următor.

Exemplul 4.6. Se consideră exemplul unei rețele cu nodurile {1, 2, 3} şi arcele (1,2), (2,3) şi (3,1) de lungimi 1, 2, respectiv1 (vezi figura 4.12.). Fie R = {2,3} şi X = 1. Într-adevăr, $L(R \cup X) = 1 + 2 + 1 = 4$ şi $A(R \cup X) = 1 + 2 + 2 + 1 = 6$. ■

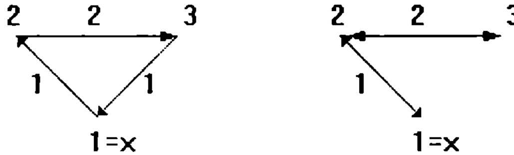


Fig. 4.12.

Lema 4.4. $(g(\widetilde{X}) - g(X^*)) / g(X^*) \leq 0.5$.

Demonstrație. Din definițiile lui $f(X)$ şi $g(X)$ rezultă că $g(\widetilde{X}) \leq f(\widetilde{X})$. Din definiția lui $\widetilde{X}$ rezultă $f(\widetilde{X}) \leq f(X^*)$. În virtutea lemei 4.3, $f(\widetilde{X}) \leq 3/2\, g(X^*)$. Din inegalitățile precedente se deduce:

$$g(\widetilde{X}) \leq f(\widetilde{X}) \leq f(X^*) \leq 3/2\, g(X^*),$$

de unde:

$$(g(\widetilde{X}) - g(X^*)) / g(X^*) \leq (3/2\, g(X^*) - g(X^*)) / g(X^*) = 0.5. ■$$

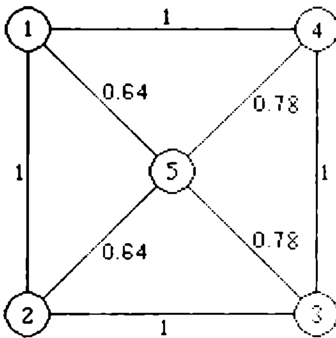


Fig. 4.13.

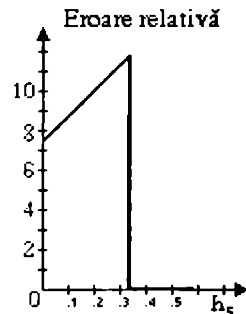


Fig. 4.14.

Exemplul 4.7. Fie graful din figura 4.13. în care $h_1 = h_2 = h_3 = h_4 = 0.5$ şi h_5 variază între 0 şi 1. S-au făcut experimentări pentru determinarea erorii relative maxime prin aplicarea euristicii. Eroarea relativă maximă determinată a fost egală cu aproximativ 11.6% şi a fost obținută în urma testării euristicii pentru o valoare a lui h_5 egală aproximativ cu 0.34. Pentru valori mai mici ale lui h_5 eroarea creşte liniar. Pentru valori mai mari ale lui h_5 eroarea relativă este nulă (vezi figura 4.14). ■

https://biblioteca-digitala.ro / https://unibuc.ro

4.5. Margini pentru problema transportului cu vehicule de capacitate limitată

Punerea problemei. Fie $X = \{x_1, x_2, ..., x_n\}$ mulţimea clienţilor, puncte în plan euclidian, unde clientul x_i este la distanţa r_i de originea O, depozitul. Distanţa maximă $\max_i\{r_i\}$ este notată $r_{\max}$, distanţa totală $\sum_{i=1}^{n} r_i$ cu $\sum r_i$ şi distanţa medie $\left(\sum_{i=1}^{n} r_i\right)/n$ cu $\bar{r}$.

Fiecare client urmează să fie vizitat de un vehicul. Al j-lea vehicul pleacă de la depozit şi se întoarce după ce a vizitat submulţimea $X_j \subseteq X$; mulţimea de puncte vizitate de acest vehicul este notată cu $X_j^0 = X_j \cup \{O\}$. Capacitatea fiecărui vehicul este q şi nici un vehicul nu vizitează mai mult decât q clienţi ($|X_j| \le q$).

Problema constă în identificarea unei mulţimi de drumuri astfel încât lungimea lor totală să fie cât mai mică; această soluţie optimală se notează cu $R^*(X)$. Un rol esenţial în cele ce urmează îl are cel mai scurt drum al comisvoiajorului ce vizitează fiecare consumator o singură dată şi a cărei lungime este notată cu $T^*(X)$. ■

4.5.1. Margini inferioare şi superioare

Teorema 4.2. $\max\left\{2\frac{n}{q}\bar{r},\ T^*(X)\right\} \le R^*(X) \le 2\left\lceil\frac{n}{q}\right\rceil\bar{r} + \left(1 - \frac{1}{q}\right) T^*(X)$.

Demonstraţia rezultă din următoarele două leme. ■

Lema 4.5. $R^*(X) \ge \max\{2(n/q)\,\bar{r},\ T^*(X)\}$.

Demonstraţie. Fie X_j mulţimea clienţilor vizitaţi de vehiculul j în soluţia optimală. Evident că:

$$T^*\left(X_j^0\right) \ge 2\max_{x_i \in X_j}\{r_i\} \ge 2\frac{\sum_{x_i \in X_j} r_i}{\left|X_j\right|} \ge \frac{2}{q}\sum_{x_i \in X_j} r_i,$$

de unde:

$$R^*(X) = \sum_j T^*\left(X_j^0\right) \ge \frac{2}{q}\sum r_i = 2\frac{n}{q}\bar{r}.$$

Inegalitatea $R^*(X) \ge T^*(X)$ este o consecinţă imediată a inegalităţii triunghiului. ■

Lema 4.6. $R^*(X) \le 2\lceil n/q\rceil\bar{r} + (1 - \lceil n/q\rceil/n)\,T^*(X)$.

Demonstraţie. Demonstraţia este constructivă, în sensul că se foloseşte o euristică pentru construirea unei soluţii a problemei transportului a cărei valoare este cel mult egală cu marginea superioară.

Fiind dat un cel mai scurt tur al comisvoiajorului, acest tur se împarte în $l = \lceil n/q\rceil$ segmente disjuncte astfel încât fiecare segment nu conţine mai mult decât q

https://biblioteca-digitala.ro / https://unibuc.ro

clienţi după care capetele segmentelor se conectează la depozit. În cadrul acestei construcţii sunt eliminate l arce din turul comisvoiajorului. Această euristică este notată *PTO* (*partiţionarea turului optimal*). Soluţia obţinută poate fi îmbunătăţită pe calea alegerii unei orientări arbitrare a turului comisvoiajorului, urmată de repetarea construcţiei de mai sus prin deplasarea capetelor celor l drumuri 1, 2, ... cu până la n - 1 poziţii în direcţia orientării turului. Urmează alegerea celei mai bune dintre soluţiile obţinute.

Pentru analiza acestei euristici, notată *IPTO* (*iterarea partiţionării turului optimal*), se observă că lungimea totală a n soluţii este egală cu:

$$2l \sum r_i + (n-l)\ T^*(X). \tag{4.8}$$

Primul termen reflectă faptul că fiecare client este vizitat pentru prima dată şi ultima dată de fiecare din cele l vehicule. Al doilea termen reflectă faptul că fiecare arc al turului este exclus de l ori şi prin urmare este prezent de n - l ori.

Evident că valoarea $R^{IPTO}(X)$ a soluţiei optimale este mai mică decât media expresiei (4.8), deci:

$$R^*(X) \le R^{IPTO}(X) \le 2l\bar{r} + (1 - l/n)\,T^*(X) = 2\lceil n/q \rceil \bar{r} + \left(1 - \lceil n/q \rceil / n\right) T^*(X). \blacksquare$$

4.5.2. Margini superioare pentru problema comisvoiajorului

Prima margine pentru $T^*(X)$ este exprimată în termeni de aria A şi perimetrul p al regiunii ce conţine pe X.

Teorema 4.3. Dacă X este conţinută într-o regiune plană conexă de arie A şi perimetru (finit) p, atunci:

$$T^*(X) \le \sqrt{2\,nA} + 3/2p. \blacksquare$$

Înainte de a demonstra teorema 4.3., se deduce un rezultat mai simplu pentru cazul când regiunea ce conţine pe X este dreptunghiulară.

Lema 4.7. Dacă X este conţinută într-un dreptunghi cu laturile a şi b, atunci:

$$T^*(X) \le \sqrt{2(n-1)ab} + 2(a+b).$$

Demonstraţie. Dreptunghiul se împarte în h benzi paralele cu latura a, presupusă orizontală (vezi figura 4.15.). Folosind marginile orizontale ale benzilor şi laturile verticale ale dreptunghiului, se construiesc două drumuri alternative prin dreptunghi. Fiecare dintre aceste drumuri este extins la un drum hamiltonian prin X, adăugând o legătură verticală dublă între fiecare punct x_i şi punctul de pe cel mai apropiat drum hamiltonian. Suma lungimilor acestor două drumuri hamiltoniene este:

$$(h+1)a + \left(2b - 2\frac{b}{h}\right) + 2n\,\frac{b}{h}.$$

https://biblioteca-digitala.ro / https://unibuc.ro

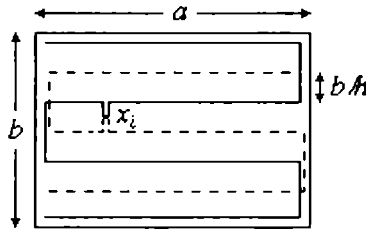


Fig. 4.15.

Pentru a extinde fiecare dintre cele două drumuri hamiltoniene, în vederea obţinerii a două tururi ale comisvoiajorului, trebuie adăugată o lungime de cel mult $2b$ - $2b$ / h în direcţie verticală şi una de cel mult $2a$ în direcţie orizontală. Rezultă că suma lungimilor celor două tururi este egală cu cel mult:

$$3a + 4b + ah + (2n - 4)\, b / h$$

şi, prin urmare:

$$T^*(X) \leq 3/2a + 2b + 1/2ah + (n-2)b/h. \tag{4.9}$$

Partea din dreapta a condiţiei (4.9) este minimă pentru $h = \sqrt{2(n-2)b/a}$. ■

Demonstrarea teoremei 4.3. Primul pas constă în interconectarea regiunii cu un sistem de linii orizontale paralele la distanţa h între ele. Lungimea totală a acestei corzi nu depăşeşte valoarea A / h. Pentru a demonstra această afirmaţie, fie o translatare a sistemului iniţial cu o lungime uniform distribuită între 0 şi h. Lungimea totală medie a corzilor astfel generate este A / h şi, prin urmare, această valoare trebuie să fie realizată pentru o anumită poziţie a sistemului.

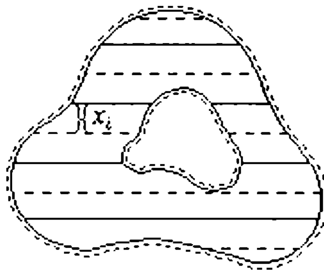


Fig. 4.16.

Presupunând liniile orientate ascendent, mulţimea lor se partiţionează în linii numerotate impar şi par. Adăugând frontierele regiunii la cele două mulţimi, rezultă două grafuri conexe (vezi figura 4.16.). Se poate arăta că ambele grafuri pot fi făcute euleriene, adăugând arce de lungime totală egală cu cel mult p / 2. Pentru aceasta se observă că, în fiecare graf, frontiera regiunii este constituită dintr-un număr finit de cicluri. Ignorând tangenţa, rezultă că o linie intersectează un ciclu de un număr par de

https://biblioteca-digitala.ro / https://unibuc.ro

ori astfel încât fiecare ciclu conține un număr par de puncte terminale ale corzilor și, prin urmare, un număr par de arce. Graful este transformat într-unul eulerian pe calea duplicării oricărui alt arc al ciclului. Pentru fiecare ciclu se poate alege una din cele două mulțimi de arce ce urmează a fi duplicate. Alegând mulțimea cea mai favorabilă, lungimea totală adăugată nu depășește $p / 2$.

În pasul următor, fiecare punct din X este conectat la ambele grafuri prin câte două arce, lungimea totală a celor patru arce fiind egală cu $2h$. Ambele grafuri rămân euleriene, lungimea arcelor lor nedepășind valoarea:

$$A / h + 3p + 2nh.$$

Rezultă că există un drum eulerian și, prin urmare, un tur de lungime ce nu depășește $A / 2h + 3p / 2 + nh$. Pentru $h = \sqrt{A / 2n}$ se obține marginea menționată în enunț. ■

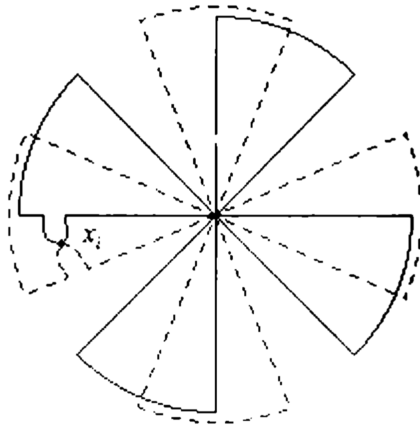


Fig. 4.17.

Teorema 4.4. $T^*(X) \leq 2\sqrt{\pi n r_{max} \bar{r}} + (2+\pi) r_{max}$, unde r_{max} reprezintă raza unui cerc în care este conținut X.

Demonstrație. Primul pas constă în împărțirea cercului de rază r_{max} în $4h$ sectoare egale. Ca și în cazul lemei 4.7. și a teoremei 4.3., frontierele sectoarelor și circumferința ciclului se folosesc două drumuri închise de forma unor flori, ce traversează cercul (vezi figura 4.17., în care $h = 4$). Fiecare drum este transformat într-un tur al comisvoiajorului printr-o conexiune dublă de lungime minimă la fiecare punct x_i; suma lungimilor acestor conexiuni duble este mai mică decât $2\,(2\pi / 4h)\, r_i = \pi r_i / h$. Rezultă că suma tururilor este mai mică decât:

$$\pi n \bar{r} \frac{1}{h} + 4 h r_{max} + 2\pi r_{max}$$

și în concluzie:

https://biblioteca-digitala.ro / https://unibuc.ro

$$T^*(X) \leq \pi\, n\, \bar{r} \frac{1}{2h} + 2\, h\, r_{\max} + \pi\, r_{\max}. \tag{4.10}$$

Alegând pentru h valoarea pentru care este minimizată expresia din dreapta în (4.10) şi rotunjind-o superior, adică luând $h = \left\lceil \sqrt{\pi\, n\, \bar{r} / 4 r_{\max}} \right\rceil$ se obţine rezultatul din enunţul teoremei. ■

Exemplul 4.8. Un exemplu ilustrativ pentru euristica *IPTO*, în care $n = 5$, $l = 2$ şi $q = 3$ este prezentat în figurile 4.18.a) - e). Se vede uşor că fiecare arc este prezent de $n - l = 5 - 2 = 3$ ori şi fiecare arc r_i este prezent de câte $2l = 4$ ori. ■

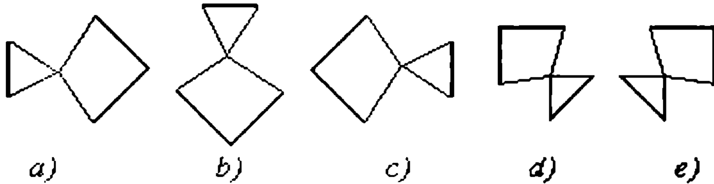


Fig. 4.18.

Observaţie. Expresia rotunjită a lui h are valoarea $\sqrt{\pi\, n\, \bar{r} / 4 r_{\max}} + \varepsilon$, unde $0<\varepsilon<1$. Introducând această expresie în partea din dreapta din (4.10) rezultă expresia:

$$\frac{\pi\, n\, \bar{r}}{\sqrt{\pi\, n\, \bar{r} / 4 r_{\max}} + \varepsilon} + 2 r_{\max}\left(\sqrt{\pi\, n\, \bar{r} / 4 r_{\max}} + \varepsilon\right) + \pi\, r_{\max},$$

care are o valoare mai mică decât:

$$\frac{\pi\, n\, \bar{r}}{\sqrt{\pi\, n\, \bar{r} / 4 r_{\max}} + \varepsilon} + 2 r_{\max}\left(\sqrt{\pi\, n\, \bar{r} / 4 r_{\max}} + \varepsilon\right) + \pi\, r_{\max} = 2\sqrt{\pi\, n\, \bar{r}\, r_{\max}} + (2+\pi)\, r_{\max}.$$

4.6. Problema orientării

Punerea problemei. Fiind date n puncte într-un spaţiu euclidian pl. , fiecare având *scorul* $s(i) \geq 0$, cu observaţia că $s(1) = s(n) = 0$, se cere identificarea unui drum de scor maxim ce începe în punctul 1 şi se încheie în punctul n, durata parcurgerii drumului fiind cel mult *TMAX*. Problema enunţată este considerată ca reprezentând o generalizare a problemei comisvoiajorului şi este rezolvată în contextul concursurilor de orientare turistică. ■

Euristica propusă. Ideea de bază a euristicii propuse constă în deplasarea succesivă dintr-un centru de greutate într-un alt centru de greutate. Fiecare centru de greutate are o mulţime de noduri (puncte) asociate şi un drum corespunzător. Progresarea continuă până când este satisfăcută o regulă de încheiere. Euristica este constituită din următorii trei paşi: 1. Construire drum, 2. Îmbunătăţire drum, 3. Calcul centru de greutate.

https://biblioteca-digitala.ro / https://unibuc.ro

Pasul 1. Obiectivul acestui pas constă în generarea unui drum inițial ce începe în nodul 1 şi se încheie în nodul n şi are un scor relativ mare, durata parcurgerii nedepăşind *TMAX*. Procedura utilizată se bazează pe un criteriu de gradare ponderată (*weighted ranking*). Mai precis, inserțiile se efectuează pe o combinație liniară de trei gradări. Aceste trei gradări sunt: scorul $SR(I)$, distanța la centrul de greutate $CR(I)$ şi suma distanțelor la două focare ale unei elipse $ER(I)$. Pentru un nod nesituat pe drumul ce trebuie generat, gradul ponderat este definit de:

$$WR(I) = \alpha\, SR(I) + \beta\, CR(I) + \gamma\, ER(I),$$

unde $\alpha + \beta + \gamma = 1$. Nodului cu cel mai mare scor i se atribuie rangul 1 şi ranguri crescătoare pentru cazul când scorurile descresc. În cazul a două scoruri egale se atribuie ranguri egale. În cazul rangului $ER(I)$, drept focare ale elipsei se consideră nodul inițial şi cel final. Nodului cu sumă minimală a distanțelor la cele două focare i se asignează rangul 1. Nodurilor cu valori crescătoare ale sumei distanțelor la cele două focare li se atribuie ranguri mai mari. Inițial, $CR(I)$ se calculează pentru mulțimea tuturor nodurilor din rețea. Nodului cel mai apropiat de $CR(I)$ i se atribuie rangul 1 şi ranguri crescătoare pe măsură ce distanțele cresc. După fiecare inserție a unui nod, centrul de greutate se recalculează pentru nodurile curente de pe drumul generat. La fiecare pas se identifică nodul cu $WR(I)$ minim, drept nod ce urmează a fi inserat în drum. În continuare se aplică succesiv procedura de inserție de cost minimal, cât timp nu e depăşit *TMAX*.

Pasul 2. Drumul precedent este modificat pe calea schimbării între ele a câte două noduri (algoritmul *2-OPT*), în vederea obținerii unui drum mai scurt pe mulțime de noduri respectivă. Urmează inserarea unui număr cât mai mare de noduri astfel încât *TMAX* să nu fie depăşit. Fie L drumul obținut.

Pasul 3. Se calculează centrul de greutate al drumului L, $g = (\bar{x}, \bar{y})$, unde:

$$\bar{x} = \sum_{i \in L} s(i)x(i) \;/\; \sum_{i \in L} s(i),\quad \bar{y} = \sum_{i \in L} s(i)y(i) \;/\; \sum_{i \in L} s(i).$$

Fie $a(i)=t(i,g)$ pentru $i=1,\dots,n$, unde $t(i,g)$ este timpul necesar pentru a parcurge drumul de la i la g. În continuare se identifică drumul de la 1 la n, după cum urmează:

a. Se calculează raportul $s(i) / a(i)$ pentru toți i.
b. Se adaugă noduri în ordinea descrescătoare a $s(i) / a(i)$, folosind inserții de cost minimal, până când nu mai poate fi adăugat nici un nod fără a depăşi *TMAX*.
c. Se foloseşte pasul 2. pentru a ajusta drumul obținut. ■

Fie L_1' drumul rezultat. Pasul 3. se reia succesiv până când în şirul $L_1', L_2', \dots$ al drumurilor obținute apare un ciclu, adică drumurile L_p' şi L_q' sunt identice pentru $q>p$. În încheiere, se alege drumul de scor maxim din mulțimea $\{L, L_1', \dots, L_q'\}$. Reluarea

https://biblioteca-digitala.ro / https://unibuc.ro

pasului 3. este limitată de condiția $q \leq 10$. În experimentările efectuate, valoarea lui q nu a depășit 5.

Observații. 1. În lucrarea [WRE72] se utilizează euristica împărțirii regiunii geografice în sectoare definite de cercuri concentrice.

2. În lucrarea [TSI84] se utilizează o euristică bazată pe tehnica Monte Carlo.

4.7. O aplicație a problemei comisvoiajorului

Punerea problemei. Fiind date paletele unei turbine de forma unei coroane circulare, se cere identificarea succesiunii plasării lor astfel încât ariile dintre două palete succesive să fie distribuite cât mai uniform. Dacă se notează cu c_{ij} aria dintre paletele i și j, atunci aria totală T are expresia:

$$T = \sum_i \sum_j c_{ij} \, x_{ij}, \qquad (4.11)$$

unde:

$$x_{ij} = \begin{cases} 1 & \text{dacă paleta } j \text{ este plasată în sens orar față de paleta } i \\ 0 & \text{altfel. } \blacksquare \end{cases}$$

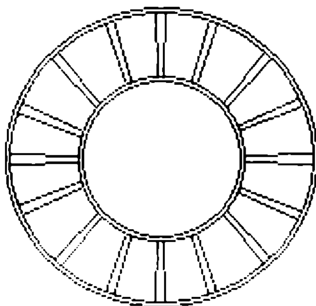


Fig. 4.19.

Problema este dificilă datorită faptului că nu există o metodă eficientă pentru identificarea tuturor valorilor c_{ij}. O metodă care necesită numai n măsurători constă în identificarea pentru fiecare paletă a două valori A_i și B_i, astfel încât:

$$c_{ij} = A_i + B_j, \text{ pentru toți } i \text{ și } j.$$

O proprietate importantă, care facilitează formularea și rezolvarea problemei studiate, constă în faptul că aria totală T, definită prin (4.11), nu depinde de ordinea plasării paletelor și are valoarea:

$$T = \sum_{i=1}^{n} \left(A_i + B_i\right).$$

https://biblioteca-digitala.ro / https://unibuc.ro

Folosind valoarea medie:

$$\bar{d} = 1/n \sum_i (A_i + B_i),$$

rezultă că o distribuire uniformă are loc dacă:

$$c_{ij} - \bar{d} = 0, \text{ pentru toți } i \text{ și } j \text{ cu } x_{ij} = 1.$$

Deoarece realizarea condiției precedente este foarte dificilă din puncte de vedere tehnologic, o soluție aproximativă acceptabilă constă în plasarea paletelor astfel încât să se minimizeze suma pătratelor abaterilor valorilor c_{ij} de $\bar{d}$. În felul acesta problema așezării optime a paletelor se poate enunța sub forma următoarei probleme a comisvoiajorului:

Problema P1. Să se minimizeze valoarea expresiei:

$$U(X) = \sum_i \sum_j \left(\bar{d} - \left(A_i + B_j\right)\right)^2 x_{ij}, \text{ unde } X = \{x_{ij}\}, \quad (4.12)$$

astfel încât să fie satisfăcute condițiile:

$$\sum_j x_{ij} = 1, \text{ pentru toți } i, \quad (4.13)$$

$$\sum_i x_{ij} = 1, \text{ pentru toți } j, \quad (4.14)$$

$$x_{ij} \in \{0, 1\}, \text{ pentru toți } i \text{ și } j, \quad (4.15)$$

permutarea indicilor i să nu conțină nici-un subciclu. ■ (4.16)

O formulare cu o structură de cost mai ușor tratabilă se poate obține în virtutea următoarei proprietăți:

Teorema 4.5. Pentru orice $X = \{x_{ij}\}$ ce satisface condițiile (4.13), (4.14) și (4.15) are loc relația:

$$U(X) = -n\,\bar{d}^2 + \sum_i \left(A_i^2 + B_i^2\right) + 2 \sum_i \sum_j \left(A_i B_j\right) x_{ij}. \quad (4.17)$$

Demonstrație. Fiecare termen din (4.12) se poate retranscrie astfel:

$$(\bar{d} - (A_i + B_j))^2 x_{ij} = (\bar{d}^2 - 2(A_i + B_j)\,\bar{d} + A_i^2 + B_j^2)\, x_{ij} + 2(A_i B_j)\, x_{ij}.$$

Deoarece pentru fiecare i, $x_{ij} = 1$ pentru o singură valoare j și pentru fiecare j, $x_{ij} = 1$ pentru o singură valoare i, rezultă:

$$\sum_i \sum_j (\bar{d}^2 - 2(A_i + B_j)\,\bar{d} + A_i^2 + B_j^2)\, x_{ij} = \sum_i (\bar{d}^2 - 2(A_i + B_i)\,\bar{d} + A_i^2 + B_i^2)$$

și, ținând seama de definiția lui $\bar{d}$, rezultă (4.17). ■

În virtutea teoremei 4.5., problema P1 poate fi reformulată astfel:

Problema P2. Să se minimizeze:

$$h(X) = \sum_i \sum_j (A_i B_j)\, x_i x_j, \quad (4.18)$$

astfel încât să fie satisfăcute condițiile (4.13), (4.14), (4.15) și (4.16). ■

https://biblioteca-digitala.ro / https://unibuc.ro

În continuare problema este studiată ținând seama de faptul că $c_{ij} = A_i B_j$ pot fi interpretate ca elementele unei matrice produs.

Se presupune că paletele sunt ordonate descrescător după valorile A_i, adică au loc inegalitățile $A_1 \geq A_2 \geq \ldots \geq A_n$. Primul pas al algoritmului propus constă în identificarea unei soluții optimale ce satisface (4.18), (4.13), (4.14) și (4.15). Se poate demonstra că o soluție optimală a problemei P2, fără condiția (4.16), se poate obține considerând $x_{ij} = 1$ atunci când B_j este a i-a cea mai mică valoare B, în cazul a două valori egale alegerea este arbitrară. Astfel soluția optimală menționată se poate obține ordonând valorile B nedescrescător și atribuind variabilelor x_{ij} valoarea 1.

Pentru descrierea în continuare a euristicii se folosește noțiunea de permutare. Orice vector $X = (x_{ij})$ ce satisface (4.13), (4.14) și (4.15) poate fi interpretat ca fiind o permutare a întregilor 1, 2, ..., n. Fiind dată o permutare $\Phi = (\Phi(1), \Phi(2), \ldots \Phi(n))$, se efectuează atribuirea $\Phi(i) = j$ dacă $x_{ij} = 1$. Deoarece pentru fiecare i, $x_{ij} = 1$ pentru un singur j rezultă că funcția $\Phi(i) = j$ este bine definită. Fiind dat un X ce satisface (4.13), (4.14) și (4.15) și Φ reprezentarea ca permutare a lui X, rezultă că:

$$\sum_i \sum_j (A_i B_j) x_{ij} = \sum_i A_i B_{\Phi(i)}.$$

Această valoare este notată în continuare cu $c(\Phi)$ și este interpretată drept cost al permutării Φ.

Fie Φ^1 o permutare ce satisface condițiile (4.13), (4.14), (4.15) și (4.18), obținută în modul descris mai înainte. După cum s-a menționat, Φ^1 este un vector al indicilor paletelor, obținut pe calea asocierii lor cu valorile lui B parcurse în ordine nedescrescătoare.

Exemplul 4.9. Fie n=6 și $\{(A_i,B_i)$: i=1,2,...,6$\}$=$\{$(0.9,0.4), (0.8,0.1), (0.6,0.3), (0.5,0.9), (0.3,0.6), (0.1,0.8)$\}$. În acest caz, Φ^1=(2,3,1,5,6,4) deoarece $B_2 \leq B_3 \leq B_1 \leq B_5 \leq$ $\leq B_6 \leq B_4$. Urmează identificarea faptului dacă Φ^1 reprezintă un singur ciclu. Pentru aceasta se reprezintă sub formă de tabel (vezi tabelul 4.1.) corespondența $\Phi^1(i)=j$:

i:	1	2	3	4	5	6
$\Phi^1(i)$:	2	3	1	5	6	4

Tabelul 4.1.

Din examinarea tabelului 4.1. rezultă că Φ^1 conține două subcicluri și anume $1 \to 2 \to 3 \to 1$ și $4 \to 5 \to 6 \to 4$. În cazul când Φ^1 conține mai multe cicluri, urmează *conectarea* lor pe calea schimbării atribuirilor la doi indici ce figurează în cicluri distincte. Astfel, dacă indicii j și k figurează în cicluri distincte, aceste două cicluri pot fi conectate prin j și k schimbând asignarea Φ^1 în Φ^2, unde $\Phi^1(i) = \Phi^2(i)$, pentru toți $i \neq j$ și $i \neq k$, $\Phi^2(j) = \Phi^1(k)$ și $\Phi^2(k) = \Phi^1(j)$. În exemplul considerat, dacă

https://biblioteca-digitala.ro / https://unibuc.ro

se consideră $j = 3$ şi $k = 4$, atunci $\Phi^2 = (2, 3, 5, 1, 6, 4)$ şi reprezintă ciclul $1 \to 2 \to 3 \to 5 \to 6 \to 4 \to 1$ definit de corespondenţa din tabelul 4.2. ■

i:	1	2	3	4	5	6
$\Phi^2(i)$:	2	3	5	1	6	4

Tabelul 4.2.

În cazuri mai complexe cu $k \geq 3$ subcicluri, după prima operaţie de conectare, Φ^2 conţine k - 1 cicluri. În continuare se identifică succesiv $\Phi^3, ..., \Phi^k$, unde Φ^k conţine un singur ciclu.

Creşterea costului c datorită conectării a două cicluri prin j şi k se calculează ţinând seama că la trecerea de la Φ^l la Φ^{l+1} au loc relaţiile $\Phi^{l+1}(i) = \Phi^l(i)$ pentru $i \neq j$ şi $i \neq k$, $\Phi^{l+1}(j) = \Phi^l(k)$ şi $\Phi^{l+1}(k) = \Phi^l(j)$. Rezultă că:

$$c(\Phi^{l+1}) - c(\Phi^l) = \sum_i A_i\ B_{\Phi^{l+1}(i)} - \sum_i A_i\ B_{\Phi^l(i)} =$$
$$= A_j\ B_{\Phi^{l+1}(j)} + A_k\ B_{\Phi^{l+1}(k)} - A_j\ B_{\Phi^l(j)} - A_k\ B_{\Phi^l(k)} = (A_j - A_k)\,(B_{\Phi^l(k)} - B_{\Phi^l(j)}). \tag{4.19}$$

Creşterea costului definită de (4.19) se notează cu $\delta_{\Phi^l,j,k}$.

Trebuie menţionat faptul că fiind date două subcicluri conţinând p, respectiv, q indici, numărul total de perechi de indici ce urmează a fi testate în vederea conectării este egal cu $p\ q$. În cadrul euristicii propuse se consideră numai perechile j, k, unde $k = j + 1$. Astfel, în exemplul de mai sus, se efectuează numai conectarea definită de $j = 3$ şi $k = 4$. O asemenea conectare se numeşte *adiacentă*. Deoarece în cadrul euristicii se consideră numai conectările menţionate, notaţia $\delta_{\Phi^l,j,j+1}$ se poate renota sub forma mai simplă $\delta_{\Phi^l,j}$. La etapa l a algoritmului, dacă Φ^l conţine două sau mai multe subcicluri, indicii i şi i+1 se identifică astfel încât:

$$\delta_{\Phi^l,i} = \min\{\delta_{\Phi^l,j} : j \text{ şi } j + 1 \text{ se află în submulţimi distincte}\}. \tag{4.20}$$

Valoarea minimă definită de (4.20) se notează cu δ^l.

Algoritmul 4.7. (*UNIFLOW*)

1. [Iniţializare] Se obţine o asignare optimală Φ^1 şi se identifică cele k subcicluri definite de Φ^1. Se face $l = 1$.
2. [Terminare] Dacă $l = k$, atunci STOP.
3. [Reducere] Se calculează δ^l şi se identifică un i care satisface condiţia $\delta_{\Phi^l,i} = \delta^l$. Se modifică Φ^l în Φ^{l+1} punând $\Phi^{l+1}(i) = \Phi^l(i + 1)$, $\Phi^{l+1}(i + 1) = \Phi^l(i)$ şi $\Phi^{l+1}(j) = \Phi^l(j)$, pentru toţi $j \neq i$, şi $j \neq i$+1. Se creşte l cu 1 şi se trece la pasul 2. ■

Algoritmul *UNIFLOW* se încheie după pasul 1 dacă Φ^1 defineşte un singur subciclu, adică un ciclu al întregilor de la 1 la n. Dacă Φ^1 defineşte k subcicluri, se identifică k - 1 conectări şi k - 1 valori $(\delta^1, \delta^2, ..., \delta^{k-1})$.

https://biblioteca-digitala.ro / https://unibuc.ro

5 PLANIFICAREA ACTIVITĂȚILOR

În acest capitol abordăm o problemă cu foarte multe aplicații, cunoscută în literatura de specialitate sub mai multe denumiri, unele cu anumite particularități, dintre care menționăm pe cele de *planificarea activităților*, *metode de ordonanțare*, *metoda drumului critic*, *programare dinamică* și altele. Complexitatea și varietatea acestor probleme permite folosirea unor metode exacte de rezolvare numai în cazuri cu totul particulare. Cele mai multe soluții sunt metode euristice. Unele din probleme au fost abordate în paragrafele 1.1.2.3.1. și 1.1.2.4.1.

5.1. Modelarea problemelor de planificarea activităților

Punerea problemei. Programarea activităților este planificarea la execuție a unei mulțimi de activități prin atribuirea de resurse materiale și temporale diferitelor activități și fixarea momentelor la care sunt procesate toate aceste activități cu o eficiență cât mai bună posibil. ■

Aplicații posibile: toate domeniile din economie, turism, transporturi, educație, cultură, servicii, justiție, statistică și multe altele.

Datele luate în considerație în problemele de programarea activităților privesc *activitățile*, *constrângerile potențiale*, *resursele* și *funcția obiectiv*. Constrângerile potențiale cuprind atât *constrângeri de succesiune* cât și *constrângeri de localizare temporală*. Resursele pot să genereze *constrângeri disjunctive*, cum ar fi faptul că două activități care folosesc același procesor nu se pot executa simultan, sau *constrângeri cumulative*, de exemplu, în cazul în care trei procesoare ar trebui să execute patru activități, ele nu pot să execute trei dintre cele patru activități în același timp fără să fi decis anterior care este activitatea care va fi întârziată.

Programarea activităților se face în așa fel încât să fie optimizat un anumit obiectiv care va fi, după caz, minimizarea duratei totale, respectarea termenelor din

https://biblioteca-digitala.ro / https://unibuc.ro

comenzi, omogenizarea numărului ce măsoară mâna de lucru necesară sau eventual minimizarea unui cost. Principalele obiective sunt: utilizarea eficace a resurselor, o întârziere a execuției activităților cât mai mică posibil și respectarea termenelor prescrise pentru terminarea fiecărei activități în parte.

Spunem că o problemă de programarea activităților este o problemă *statică* sau *determinată* dacă mulțimea activităților de executat este dată de la început. În caz contrar spunem că acestă problemă este o problemă *dinamică* sau *nedeterminată* sau *stohastică*. În acest caz, activitățile evoluează în timp de o manieră nedeterminată.

5.1.1. Notații utilizate

Vom nota cu A mulțimea activităților și cu n numărul activităților de executat; durata activității i este t_i; execuția nu poate să înceapă înainte de r_i care este momentul de disponibilitate sau de începere cel mai devreme și trebuie să se termine înainte de d_i care este momentul cel mai târziu sau termenul de abandon (*deadline*).

Vom nota cu s_i momentul de începere a execuției activității i și cu c_i momentul terminării execuției. Dacă activitatea i nu este interuptibilă, are loc relația:

$$c_i = s_i + t_i.$$

O condiție necesară pentru a se putea realiza o planificare a unor activități este:

$$r_i \leq s_i < c_i \leq d_i, \text{ pentru orice } i = 1, 2, ..., n.$$

În unele cazuri, se pot considera ponderi w_i pentru activități, acestea putând fi de exemplu penalități de întârziere dacă $c_i > d_i$. Mai general, se poate asocia activității i o funcție $g_i(u)$ care este în cele mai multe dintre cazuri crescătoare și reprezintă costul asociat terminării execuției activității i la momentul u.

Două activități din A care nu sunt legate între ele prin relații de precedență se numesc *activități independente*. Constrângerile de precedență cele mai generale între două activități i și j, numite *constrângeri potențiale*, se pot scrie sub forma $s_j - s_i \geq a_{ij}$. În particular, dacă $a_{ij} = t_i$, spunem că *i precede* pe j. Mulțimea constrângerilor potențiale se poate reprezenta printr-un graf direcționat ponderat $G = (A, R)$, unde A este mulțimea activităților și R este mulțimea constrângerilor potențiale din care s-au eliminat toate arcele tranzitive ce reprezintă constrângeri redondante. Acest graf este cunoscut sub numele de *diagramă Hasse*.

5.1.2. Modul de execuție

Vom spune că o activitate este *fracționabilă* sau că este *interuptibilă* sau că este *preemptivă* dacă activitatea se poate executa pe bucăți, la momente diferite și eventual pe procesoare diferite. Activitatea i poate fi *întreruptă* de o activitate j la momentul t

https://biblioteca-digitala.ro / https://unibuc.ro

dacă i este în execuție înainte de t și folosește o resursă alocată lui j la momentul t, aceasta producând întreruperea execuției activității i.

Pentru o *problemă repetitivă* se pleacă de la o mulțime de activități *A de bază* pentru care se definesc o mulțime de constrângeri ce privesc resursele și o mulțime de constrângeri *interne* de succesiune. Fiecare activitate de bază i dă naștere la o infinitate de activități *derivate* $(i, 1)$, $(i, 2)$, ..., (i, h), ... care sunt executate cu respectarea *constrângerile interne*, de tipul activitatea (i, h) precede activitatea (j, h), și, de asemenea, o mulțime de *constrângeri externe* definite prin triplete de forma (i, j, k), $i \in A$, $j \in A$ și $k \in N^*$ care arată că, pentru orice h, activitatea derivată $(j, h+k)$ se execută după executarea activității derivate (i, h). Acest tip de probleme se întâlnesc în producțiile de serie din industrie.

Pentru o *problemă de atelier* activitățile, considerate ca operații elementeare, sunt grupate în entități numite *lucrări* (engl. *job*). Un atelier conține m mașini distincte și fiecare lucrare este o mulțime de m operații elementare, fiecare dintre ele executându-se pe câte o mașină diferită de mașinile ce execută celelalte operații ale aceleiași lucrări. Dacă operațiile elementare sunt independente și unele dintre lucrări nu conțin toate cele m tipuri de activități, atunci se spune că este vorba de o *problemă open-shop*. Dacă cele m operații elementare ale unei lucrări sunt ordonate total, nu neapărat aceeași ordonare pentru toate lucrările, atunci se spune că este vorba de o *problemă job-shop*. Dacă operațiile elementare ale lucrărilor sunt ordonate total fiind utilizate procesoarele în aceeași ordine pentru toate lucrările, atunci spunem că este vorba de o *problemă flow-shop* sau *prelucrare în flux*.

5.1.3. Resurse

Se spune despre o resursă că este *refolosibilă* dacă, după ce a fost alocată la o activitate, ea redevine disponibilă la terminarea acestei activități pentru alte activități. De exemplu, mașinile, procesoarele, fișierele, personalul sunt resurse refolosibile. O resursă este *consumabilă* dacă, după ce a fost alocată la o activitate, nu mai este disponibilă pentru activitățile care mai sunt de executat. Așa pot fi considerate drept resurse consumabile banii, materiile prime, hârtia de scris etc. O anumită resursă poate să fie disponibilă numai la un anumit moment.

Mașinile pot să fie *identice*, activitățile respective putându-se efectua practic pe oricare dintre ele fără deosebire, sau *diferite*. În cazul mașinilor diferite se notează cu t_{ij} timpul de execuție al activității i pe mașina j. Deosebirea între mașini se face fie prin viteza de lucru, mașinile executând același fel de operații, fie prin tipurile de operații ce se pot executa, fie prin ambele criterii de deosebire.

https://biblioteca-digitala.ro / https://unibuc.ro

5.1.4. Funcţii obiectiv

Variabilele care intervin cel mai frecvent în exprimarea funcţiilor ce definesc eficacitatea unei programări sunt: momentul c_i al terminării execuţiei activităţii i, întârzierea activităţii i dată de $T_i = \max\{0, c_i - d_i\}$ şi indicatorul de întârziere notat U_i şi defint prin $U_i = 0$ dacă $c_i \leq d_i$ şi $U_i = 1$ altfel.

Criteriile sau *funcţiile obiectiv* cele mai utilizate sunt următoarele:

1. *Durata totală*, notată C_{max}, care este egală cu momentul de terminare al celei mai întârziate activităţi; expresia corespunzătoare este: $C_{max} = \max\limits_{i \in A} \{c_i\}$. În continuare, dacă nu se specifică altfel, acesta este criteriul principal pentru problemele de programarea activităţilor pe care le prezentăm.
2. *Respectarea termenelor.* Se poate urmări minimizarea celei mai mari întârzieri dată de expresia $T_{max} = \max\limits_{i \in A} \{T_i\}$ sau suma întârzierilor dată de expresia $\sum\limits_{i \in A} T_i$ sau suma ponderată a întârzierilor dată de expresia $\sum\limits_{i \in A} w_i T_i$ sau numărul de activităţi întârziate dat de expresia $\sum\limits_{i \in A} U_i$ sau, în sfârşit, numărul ponderat al activităţilor întârziate dat de expresia $\sum\limits_{i \in A} w_i U_i$.
3. *Minimizarea costului.* De exemplu, criteriul $\sum\limits_{i \in A} w_i c_i$ permite evaluarea stocurilor utilizate.
4. *Numărul întreruperilor.* Dacă activitatea i este întreruptă de n_i ori, atunci numărul total de întreruperi dat de expresia $\sum\limits_{i \in A} n_i$ poate să fie considerat drept un criteriu secundar esenţial.
5. *Criterii pentru probleme repetitive.* În acest caz se poate urmări maximizarea frecvenţei de execuţie a unei activităţi particulare sau minimizarea unei durate generalizate definită, pentru o soluţie dată, prin expresia $\lim\limits_{n \to \infty} \sup(D_n / n)$, unde D_n este momentul terminării celei de-a n-a subprobleme, adică $D_n = \max\limits_{i \in A} \{c^{n}\}$.

Se spune că un criteriu de minimizare $f(c_1, c_2, ..., c_n)$ este regulat dacă:

$$(c_1 \leq d_1, c_2 \leq d_2, ..., c_n \leq d_n) \Rightarrow f(c_1, c_2, ..., c_n) \leq f(d_1, d_2, ..., d_n).$$

Un criteriu regulat permite de foarte multe ori punerea în evidenţă a unor submulţimi de programări care conţin soluţia optimală, numite *submulţimi dominante.*

De exemplu, un caz particular în care este asigurată existenţa unei mulţimi dominante este specificat în teorema următoare:

Teorema 5.1. Pentru orice criteriu regulat corespunzător unei probleme cu un procesor, în cazul în care toate activităţile sunt disponibile la momentul 0, mulţimea

https://biblioteca-digitala.ro / https://unibuc.ro

programărilor pentru care maşina nu este liberă atâta timp cât nu au fost executate toate lucrările formează o mulțime dominantă. ■

Spunem că două criterii sunt *echivalente* dacă orice soluție optimală în raport cu primul criteriu este soluție optimală în raport cu al doilea criteriu şi reciproc.

Exemplul 5.1. Criteriul bazat pe media momentelor de terminare, notată cu $\bar{c}$, este echivalent cu cel bazat pe media întârzierilor algebrice, notată $\bar{L}$, dar criteriul bazat pe întârzierea absolută, notată T_{max}, nu este echivalent cu criteriul bazat pe întârzierea algebrică absolută, notată L_{max}. Se poate demonstra că o soluție optimală pentru L_{max} este soluție optimală şi pentru T_{max} dar nu şi reciproc, după cum se poate vedea pe exemple simple. ■

5.1.5. Reprezentarea soluțiilor

O metodă de reprezentare a unei soluții a planificării activităților este prin intermediul *diagramelor Gantt*. Figurile 5.1. şi 5.4. sunt exemple de diagrame Gantt. În aceste diagrame este inclusă o axă a timpului în raport cu care sunt indicate asignările diferitelor activități pentru fiecare dintre procesoare. În diagrame se pot efectua reprezentări în raport cu timpul şi modul de alocare a celorlalte resurse implicate în executarea activităților.

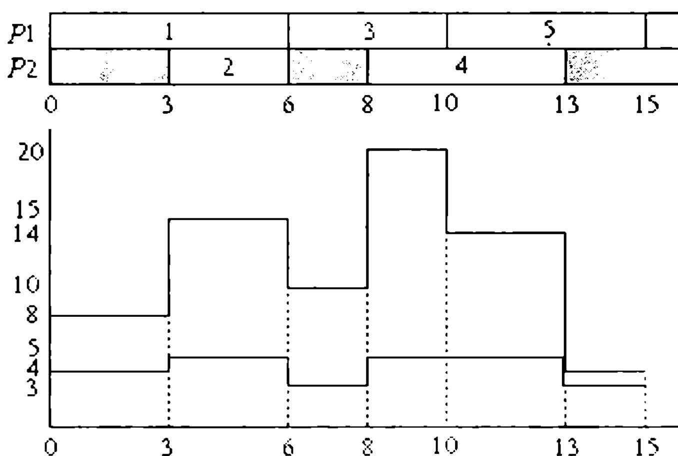


Fig.5.1.

Exemplul 5.2. În figura 5.1. se găseşte diagrama Gantt pentru programarea a cinci activități $A = \{1, 2, 3, 4, 5\}$, având respectiv duratele $t_1 = 6$, $t_2 = 3$, $t_3 = 4$, $t_4 = 5$, $t_5 = 5$, utilizând din resursa $R1$ câte 4, 1, 3, 2, 3 unități, din resursa $R2$ câte 8, 7, 10, 10, 4 unități şi începând execuția la momentele 0, 3, 6, 8, 10. Se pot utiliza două procesoare $P1$ şi $P2$. Graficul pentru $R2$ este deasupra celui pentru $R1$. ■

https://biblioteca-digitala.ro / https://unibuc.ro

5.1.6. Modelarea problemei principale de programare

Pentru o mulțime $A = \{1, 2, ..., n\}$ de activități care respectă constrângeri temporale de tip inegalități potențiale $s_j - s_i \geq a_{ij}$, unde s_i, respectiv s_j, este momentul de începere a activității i, respectiv j, şi a_{ij} este un număr real dat, trebuie determinată o mulțime $S = \{s_i \mid i \in A\}$ de momente pozitive care face minimă durata totală a programării $\max_{i \in A}(s_i + t_i)$ şi satisface inegalitățile potențiale.

Acestei probleme i se asociază un graf direcționat ponderat $G = (X, U)$, **numit** *graf de precedență*, unde $X = A \cup \{0, n + 1\}$, cu 0 o activitate fictivă de ***start*** **şi** $n + 1$ o activitate fictivă *finală*, ambele de timp nul şi U conține arcele $(0, i)$, $i \in A$, de valoare 0, arcele (i, j) asociate constrângerilor potențiale, de valoare a_{ij}, şi arcele $(i, n + 1)$, $i \in A$, de valoare t_i. Dacă există trei arce astfel încât $a_{ik} \leq a_{ij} + a_{jk}$, atunci arcul (i, k) poate fi eliminat fiind redondant.

Exemplul 5.3. Fie $A = \{1, 2, 3, 4, 5\}$, $t_1 = t_3 = t_5 = 1$, $t_2 = 3$, $t_4 = 2$ care sunt supuse la următoarele constrângeri temporale:

1. Activitatea 2 începe la momentul 3.
2. Activitățile 3 şi 4 să se suprapună pe cel puțin o unitate de timp.
3. Activitateà 4 nu poate să înceapă decât după terminarea activităților 1 şi .
4. Activitatea 5 nu poate să înceapă înainte de a începe activitatea 3.

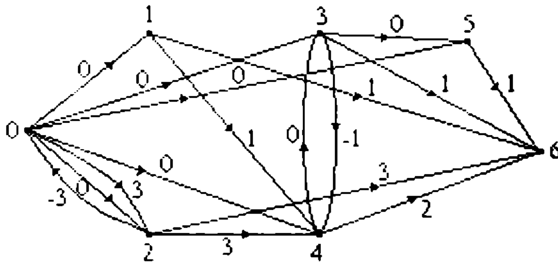


Fig. 5.2.

Prima constrângere se traduce prin $s_2 - s_0 \geq 3$ şi $s_0 - s_2 \geq -3$. Pentru a doua constrângere trebuie descris faptul că activitățile 3 şi 4 încep înainte de terminarea celeilalte cu cel puțin o unitate de timp, adică $s_3 \leq s_4 + t_4 - 1$ şi $s_4 \leq s_3 + t_3 - 1$. A treia constrângere se traduce prin $s_4 - s_2 \geq t_2$ şi $s_4 - s_1 \geq t_1$. Ultima constrângere duce la $s_5 \geq s_3$. Se ob ne astfel graful de precedență din figura 5.2. şi, după eliminarea arcelor inutile, se obține graful echivalent simplificat din figura 5.3. O soluție a problemei este $\{s_1=0, s_2=3, s_3=6, s_4=6, s_5=5\}$ şi diagrama Gantt corespunzătoare ei este dată în figura 5.4. Această soluție este optimală deoarece existența drumului (0,2,4,6) în graf implică $s_6 \geq 8$, valoarea găsită fiind chiar 8. ■

https://biblioteca-digitala.ro / https://unibuc.ro

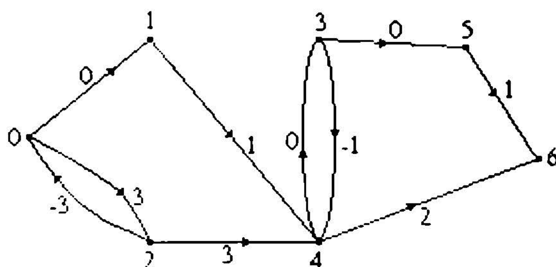


Fig. 5.3.

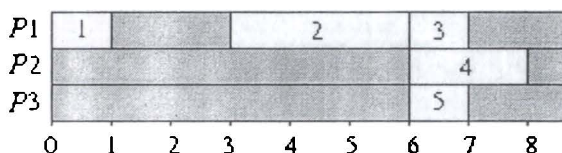


Fig. 5.4.

5.1.7. Modelarea cu rețele Petri

Reţelele Petri sunt utilizate pentru a modela comportamentul dinamic al sistemelor discrete. Ele permit reprezentarea pe un singur graf a evenimentelor generatoare de schimbare a stărilor, condițiile necesare pentru a face aceste schimbări de stare şi regulile de calcul a unei noi stări dacă se produce evenimentul corespunzător.

Reţelele Petri sunt alcătuite din două tipuri de obiecte: *locuri* și *tranziții*. Locurile sunt asociate cu variabilele de stare ale sistemelor şi tranziţiile reprezintă evenimente susceptibile de a modifica starea sistemului. Dacă o rețea Petri modelează o problemă de programarea activităților, locurile vor fi asociate de cele mai multe ori cu diferitele tipuri de resurse utilizate de activități și constrângerilor de succesiune între activități. Tranzițiile sunt asociate cu activitățile. Evenimentele, realizarea tranziţiilor, corespund executării activităţilor. Execuția unor activități nu este posibilă decât dacă ele dispun de resursele necesare şi dacă activităţile care le preced au fost terminate. Aceste *precondiţii* pentru executarea unei activități se traduc prin existenţa unor arce ponderate cu valori întregi care leagă unele locuri cu tranziţiile considerate. La terminarea execuției unei activități, pot să fie restituite sau pot să fie consumate sau pot să fie generate unele resurse. Aceste *postcondiţii* de terminare a activităţilor se traduc prin arce ponderate ce leagă tranziția considerată cu o submulțime de locuri.

Exemplul 5.4. Să considerăm mulțimea activităților $A=\{1,2,3,4\}$, cu timpii de execuţie $t_1=6$, $t_2=7$, $t_3=4$, $t_4=5$, care sunt supuse la următoarele constrângeri temporale:

1. Activitatea 1 precede activitățile 2 şi 3.
2. Activitatea 4 poate să înceapă de îndată ce s-a executat jumătate din activitatea 3.

https://biblioteca-digitala.ro / https://unibuc.ro

În figura 5.5. este dată rețeaua Petri temporizată asociată acestei probleme. Pe această figură, cercurile sunt stări și dreptunghiurile sunt tranziții. A fost introdusă tranziția $T_{3,1}$ pentru a marca jumătate din execuția activității 3, T_0 este tranziția inițială și T_5 este tranziția finală a rețelei Petri. ■

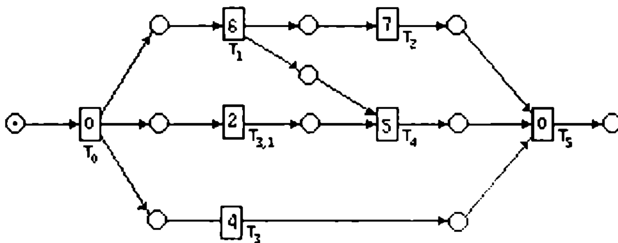


Fig. 5.5.

Exemplul 5.5. Să considerăm mulțimea activităților independente $A=\{1,2,3,4,5,6\}$, cu timpii de execuție $t_1=2$, $t_2=7$, $t_3=3$, $t_4=5$, $t_5=8$, $t_6=4$, care sunt executate pe trei procesoare identice. Figura 5.6. reprezintă rețeaua Petri pentru modelarea acestei probleme, pe arce fiind marcate resursele utilizate și respectiv produse de fiecare tranziție. Figura 5.7. prezintă diagrama Gantt pentru o soluție a problemei. Această soluție are timpul total 11 unități, pe când soluția optimă are timpul total de 10 unități fiind obținută, de exemplu, prin executarea activităților 1 și 5 pe primul procesor, a activităților 2 și 3 pe al doilea procesor și a activităților 4 și 6 pe al treilea procesor. ■

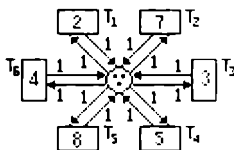


Fig. 5.6.

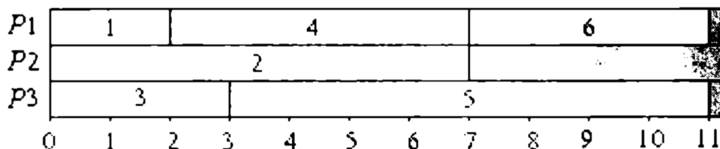


Fig. 5.7.

Exemplul 5.6. Să considerăm mulțimea activităților independente $A=\{1,2,3,4\}$, cu timpii de execuție $t_1=2$, $t_2=6$, $t_3=9$, $t_4=3$, care sunt executate pe două procesoare identice. Activitățile 1 și 4 sunt interuptibile și celelalte nu sunt interuptibile. Figura 5.8. reprezintă rețeaua Petri pentru modelarea acestei probleme și figura 5.9. prezintă diagrama Gantt pentru o soluție a problemei. Această soluție este chiar soluția optimală în raport cu minimizarea timpului total de lucru. ■

https://biblioteca-digitala.ro / https://unibuc.ro

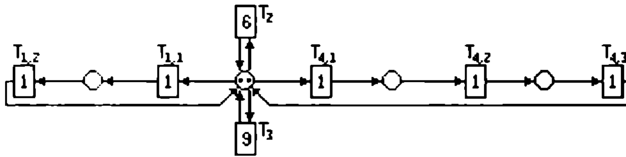


Fig. 5.8.

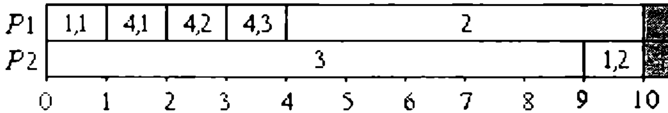


Fig. 5.9.

5.2. Metode polinomiale de rezolvare a problemei planificării activităților

În acest paragraf vom prezenta mai multe metode polinomiale de soluționare a problemei planificării activităților. Vom prezenta metode bazate pe drumul critic, metode bazate pe circuite critice și metode clasice de soluționare. Dintre metodele polinomiale fac parte și metodele ce folosesc programarea liniară. Aceste metode sunt prezentate în capitolul 6.

5.2.1. Algoritmi bazați pe metoda drumului critic

5.2.1.1. Mulțimi de potențiale pe un graf conjunctiv

Se numește *graf conjunctiv* un graf direcționat, ponderat $G = (A, U)$ care are un nod 0, numit *rădăcină*, și un nod $n + 1$, numit *antirădăcină*, astfel încât există un drum de valoare pozitivă de la rădăcină la orice alt nod și un drum de valoare pozitivă de la orice nod diferit de antirădăcină la aceasta.

Se numește *mulțime de potențiale* pe un graf conjunctiv $G = (A, U)$ orice aplicație s definită pe mulțimea A cu valori reale astfel încât $s_0 = 0$ și pentru orice arc conjunctiv (i, j) de valoare v_{ij} are loc relația $s_j - s_i \geq v_{ij}$. Mulțimea potențialelor se notează τ, deci $\tau = \{s_i \mid i \in A\}$.

Teorema 5.2. (de existență) O condiție necesară și suficientă pentru a exista o mulțime de potențiale pe un graf conjunctiv $G = (A, U)$ este aceea de a nu exista nici un circuit de valoare strict pozitivă. ■

Corolarii: 1. Dacă un graf conjunctiv este fără circuite, atunci există pentru el cel puțin o mulțime de potențiale.

https://biblioteca-digitala.ro / https://unibuc.ro

2. Dacă ponderile arcelor unui graf conjunctiv nu sunt negative, atunci există o mulţime de potenţiale dacă şi numai dacă toate circuitele sunt de valoare nulă.

Pentru un graf conjunctiv fără circuite de valori strict pozitive, vom nota cu $l(i,j)$ valoarea maximă a unui drum de la i la j; prin definiţie $l(i,i)=0$ pentru orice $i \in A$ şi $l(i,j) = -\infty$ dacă nu există drum de la i la j.

Lema 5.1. Dacă $\tau = \{s_i \mid i \in A\}$ este o mulţime de potenţiale şi l este definit ca mai sus, atunci $s_j - s_i \geq l(i, j)$, pentru orice $(i, j) \in U$. ■

Demonstraţia lemei rezultă uşor din definiţii. Cu ajutorul lemei 5.1. se poate demonstra următoarea proprietate:

Teorema 5.3. Pentru orice mulţime de potenţiale $\tau = \{s_i \mid i \in A\}$ are loc relaţia $r_i \leq s_i$, unde $r_i = l(0, i)$. ■

Teorema 5.3. arată că, pentru mulţimea de potenţiale $R = \{r_i \mid i \in A\}$, activităţile se execută cel mai devreme posibil. Se spune că R este *calată la stânga*.

În metoda potenţialelor, se caută o programare a activităţilor de durată totală minimă. O astfel de programare corespunde la o mulţime de potenţiale cu s_{n+1} minimală care este valoarea maximă a unui drum de la 0 la $n + 1$. Un astfel de drum se numeşte *drum critic* şi valoarea optimală a lui se notează cu s^*.

Teorema 5.4. Fie $F = \{f_i = \min[l(0, n + 1) - l(i, n + 1)] \mid i \in A\}$ o mulţime de potenţiale optimală. Dacă $\tau = \{s_i \mid i \in A\}$ este o altă mulţime de potenţiale optimală, atunci $s_i \leq f_i$, pentru orice $i \in A$. ■

Teorema 5.4. arată că F este o mulţime de potenţiale optimală în care orice activitate se execută cel mai târziu posibil. Se spune că F este *calată la dreapta*.

Teorema 5.5. Mulţimea $F(\delta) = \{f_i(\delta) = \min\{-l(i, 0), f_i + \delta\} \mid i \in A\}$ este o mulţime de potenţiale de durată $\min\{s^* + \delta, -l(n + 1, 0), f_i + \delta\}$. Ea este calată la dreapta dintre programările de aceeaşi durată. ■

5.2.1.2. Metoda potenţiale-activităţi

Pentru definirea metodei potenţiale-activităţi dăm următorul exemplu.

Exemplul 5.7. Fie mulţimea activităţilor $A = \{1,2,3,4,5,6,7\}$, cu duratele $t_1=3$, $t_2=7$, $t_3=4$, $t_4=6$, $t_5=5$, $t_6=3$, $t_7=2$, cu constrângerile: 1 precede pe 3, 1 şi 2 preced pe 4, 3 precede pe 5, 3 şi 4 preced pe 6 şi 6 precede pe 7. Se caută un mod de execuţie a celor 7 activităţi într-un timp cât mai scurt. În figura 5.10. este reprezentat graful de precedenţă asociat acestei probleme. Au fost adăugate, pe lângă arcele care decurg din relaţiile de precedenţă impuse, arce de la rădăcină la nodurile care nu au predecesori şi arce de la nodurile care nu au succesori la antirădăcină. ■

https://biblioteca-digitala.ro / https://unibuc.ro

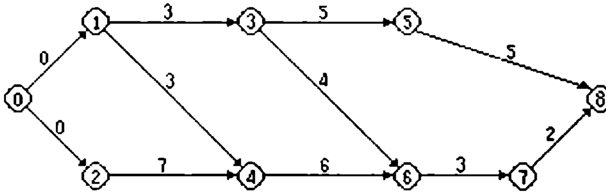


Fig. 5.10.

Pentru orice programare optimală $\tau = \{s_i \mid i \in A\}$ au loc relațiile $r_i \le s_i \le f_i$. Pentru un graf fără circuite se pot calcula programările cel mai devreme R și cel mai târziu F aplicând algoritmii următori cunoscuți sub numele de ***algoritmii lui Bellman.***

Algoritmul 5.1. (***Bellman***)

1. [Inițializare] Se face $r_0 = 0$ și se marchează nodul 0.
2. [Terminare] Dacă toate nodurile au fost marcate, atunci STOP.
3. [Testare circuite] Dacă nu există noduri nemarcate pentru care toți predecesorii sunt marcați, atunci STOP. (Graful are cel puțin un circuit).
4. [Alegere] Se alege un nod j nemarcat, care are toți predecesorii marcați.
5. [Marcare] Se face $r_j = \max\limits_{(i,j)\in U} (r_i + v_{ij})$ și se marchează nodul j.
6. [Ciclare] Se merge la pasul 2. ■

Algoritmul 5.2. (***Bellman***)

1. [Algoritm 5.1.] Se aplică algoritmul 5.1. pentru determinarea valorilor $r_0, ..., r_{n+1}$.
2. [Testare circuite] Dacă a fost detectat un circuit, adică algoritmul 5.1. s-a terminat în pasul 3., atunci STOP.
3. [Inițializare] Se face $f_{n+1} = r_{n+1}$ și se marchează nodul $n + 1$.
4. [Terminare] Dacă toate nodurile au fost marcate, atunci STOP.
5. [Alegere] Se alege un nod i nemarcat, pentru care toți succesorii au fost marcați.
6. [Marcare] Se face $f_i = \min\limits_{(i,j)\in U} (f_j - v_{ij})$ și se marchează nodul i.
7. [Ciclare] Se merge la pasul 4. ■

Exemplul 5.8. Aplicând algoritmii 5.1. și 5.2. pentru datele din exemplul 5.7. se obțin succesiv valorile: r_0=0, r_1=0, r_2=0, r_3=3, r_4=7, r_5=8, r_6=13, r_7=16, r_8=18, apoi f_8=18, f_7=16, f_6=13, f_5=13, f_4=7, f_3=8, f_2=0, f_1=4, f_0=0. ■

În raport cu valorile calculate aplicând algoritmii lui Bellman se pot defini diferite cantități care dau gradul de libertate al activităților. Astfel numim ***marje totală*** M_i a unei activități i timpul maxim de variație a începutului execuției acestei activități care nu influențează momentul de terminare a tuturor activităților. Aceasta este dată de formula $M_i = f_i - r_i$.

https://biblioteca-digitala.ro / https://unibuc.ro

Prin *marje liberă* m_i a unei activități i se înțelege timpul maxim de variație a începutului execuției acestei activități care nu influențează pentru succesorii ei începerea execuției cel mai devreme posibil. Aceasta este dată de formula $m_i = \min_{j\in U^+(i)} (r_j - r_i - v_{ij})$. Prin *marje sigură* μ_i a unei activități i se înțelege timpul maxim de variație, dacă există, de care se dispune pentru execuția activității i când predecesorii ei încep execuția cel mai târziu posibil și succesorii ei încep execuția cel mai devreme posibil. Aceasta este dată de formula:

$$\mu_i = \max\{0, \min_{j\in U^+(i)} (r_j - v_{ij}) - \max_{j\in U^-(i)} (f_j + v_{ji})\}.$$

În formulele anterioare, $U^+(i)$ reprezintă mulțimea succesorilor direcți ai lui i și $U(i)$ reprezintă mulțimea predecesorilor direcți ai lui i.

Drumurile de valoare maximală de la 0 la $n + 1$ au un rol central în problemele de programarea activităților și se numesc *drumuri critice*. Activitățile cuprinse în aceste drumuri au marja totală zero. Se spune despre aceste activități că sunt *activități critice*. Întârzierea unei activități critice cu o valoare δ produce întârzierea întregii programări cu aceeași valoare. La supravegherea realizării unei planificări, atenția principală trebuie acordată activităților critice.

i	0	1	2	3	4	5	6	7	8
M_i	0	4	0	5	0	5	0	0	0
m_i	0	0	0	0	0	5	0	0	-
μ_i	-	0	0	0	0	0	0	0	-

Tabelul 5.1.

Exemplul 5.9. Pentru datele din exemplul 5.7. se obțin valori pentru marja totală, marja liberă și marja sigură date în tabelul 5.1. Există un singur drum critic și anume (0, 2, 4, 6, 7, 8) de lungime 18. ■

5.2.1.3. Metode seriale de planificare folosind drumuri critice

În metodele seriale, euristicile permit clasificarea activităților în raport cu diferite relații de ordine de prioritate. Astfel activitățile pot să fie programate plecînd de la un moment inițial 0 și crescând timpul în funcție de activitățile deja programate. La momentul t se alege o sarcină de cea mai mare prioritate dintre sarcinile disponibile, adică acelea pentru care toți predecesorii au fost terminați și pentru care cererile de resurse sunt cel mult egale cu cantitățile disponibile la momentul t. Stabilirea ordinelor de prioritate se poate face static sau dinamic. În metodele statice, prioritățile nu se modifică în timpul planificărilor, fiind stabilite o singură dată la începutul planificării, pe când în cazul dinamic ordinele de prioritate se pot modifica

https://biblioteca-digitala.ro / https://unibuc.ro

în timp în funcţie de situaţie. Algoritmii de acest fel fac parte din clasa algoi :nilor de planificare pe bază de liste şi au structura următoare.

Algoritmul 5.3. (***pe bază de liste***)

1. [Iniţializare] Se face $B = \varnothing$ şi $t = 0$.
2. [Terminare] Dacă B conţine toate activităţile, atunci STOP.
3. [Disponibilitate] Se determină momentul t mai mare sau egal cu valoarea curentă la care devine disponibilă prima activitate dintre cele care nu aparţin lui B.
4. [Alegere] Se alege dintre activităţile disponibile la momentul t o activitate i de cea mai mare prioritate şi se face $s_i = t$ şi $B = B \cup \{i\}$.
5. [Ciclare] Se merge la pasul 2. ■

Dintre euristicile cele mai utilizate pentru a stabili reguli de prioritate amintim:

- ordonarea activităţilor crescător în raport cu începerea cel mai târziu dată de f_i;
- ordonarea activităţilor crescător în raport cu terminarea cel mai târziu dată de $f_i + t_i$, unde t_i este timpul de execuţie a activităţii i.

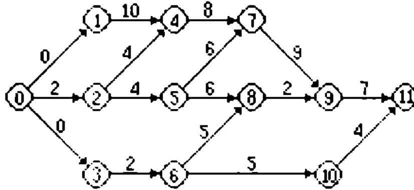


Fig. 5.11.

i	1	2	3	4	5	6	7	8	9	10
t_i	10	4	2	8	6	5	9	2	7	4
$R1$	3	3	1	1	1	2	3	2	1	2
$R2$	0	0	0	1	1	1	0	1	0	0
r_i	0	2	0	10	6	2	18	12	27	10
f_i	0	6	18	10	12	20	18	25	27	30

Tabelul 5.2.

Exemplul 5.10. În figura 5.11. sunt reprezentate constrângerile unei probleme de programare cu 10 activităţi (0 este rădăcina şi 11 este antirădăcina). Activit› le necesită la începutul execuţiei două tipuri de resurse, $R1$ cu cel mult 5 unităţi şi $R2$ cu cel mult o unitate. Timpii de execuţie, cererile de resurse $R1$ şi $R2$ şi momentele de începere a execuţiei cel mai devreme posibil, respectiv cel mai târziu posibil sunt date în tabelul 5.2. În figura 5.12. este dată diagrama Gantt corespunzătoare soluţiei obţinută prin aplicarea algoritmului 5.3. cu priorităţi pentru activităţi cu cele mai mici termene, pentru începerea cel mai târziu posibil. Sunt indicate, de asemenea, consumul din fiecare resursă pe fiecare perioadă de timp. Această soluţie nu este optimală având valoarea 44. În figura 5.13 este dată o soluţie optimală a problemei de valoare 40. ■

https://biblioteca-digitala.ro / https://unibuc.ro

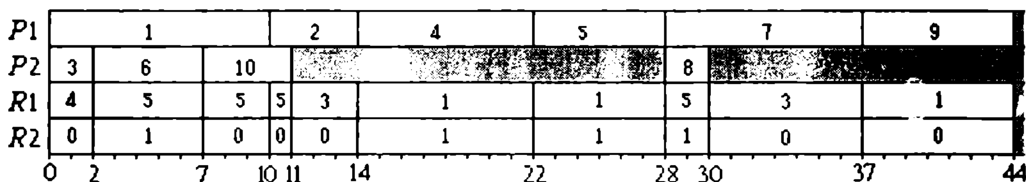


Fig. 5.12.

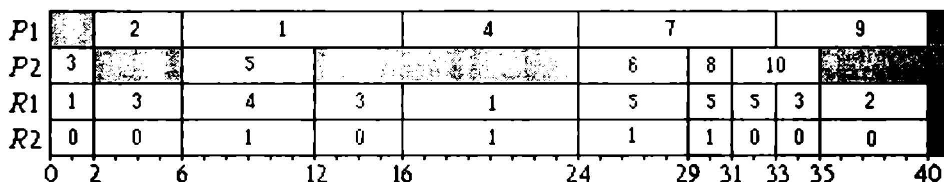


Fig. 5.13.

Pentru metodele seriale se pot da exemple în care nu se poate determina cât de departe este soluția găsită față de soluția optimală. De asemenea în aplicarea acestei metode pot să apară diferite anomalii. De exemplu, prin creşterea disponibilității resurselor se aşteaptă obținerea unor soluții mai bune dar rezultatul poate fi contrar aşteptărilor. Exemple de astfel de anomalii sunt prezentate în paragraful 5.2 2.6.

5.2.1.4. Euristica Schrage pentru un singur procesor

Punerea problemei. Fiind date n activități accesibile la timpii r_i, necesitând duratele de procesare t_i şi termenele d_i, $i = 1, 2, ..., n$ se cere planificarea ordinei de execuție a activităților pe un procesor astfel încât timpul de lucru să fie minimizat. ■

Algoritmul 5.4. (***Schrage***)

1. [Inițializare] Se face $i = 0$.
2. [Terminare] Dacă toate activitățile au fost planificate, atunci STOP.
3. [Disponibilitate] Se determină momentul U_i la care devine liber procesorul şi V_i cel mai mic moment de acces al unei activități neprocesate încă.
4. [Alegere] Se alege activitatea k din mulțimea $\{j \mid j$ este o activitate neprocesată şi $r_j \leq \max\{U_i, V_i\}\}$ cu termenul cât mai mare posibil. Dacă mai multe activități satisfac această condiție, se alege activitatea cu cel mai mare timp de execuție.
5. [Planificare] Se face $s_k = U_i$ şi se consideră k planificată.
6. [Ciclare] Se face $i = i + 1$ şi se merge la pasul 2. ■

Această euristică necesită O(n log n) paşi.

Fie $\sigma = (\sigma(1), \sigma(2), ..., \sigma(n))$ secvența generată în modul descris de algoritmul Schrage şi T_S timpul total. Atunci, după cum este ilustrat în figura 5.14. în care segmentele punctate corespund unor intervale de execuție sau de aşteptare a eliberării procesorului, are loc relația:

https://biblioteca-digitala.ro / https://unibuc.ro

$$T_S = r_{\sigma(i)} + \sum_{h=i}^{j} t_{\sigma(h)} + d_{\sigma(j)}, \tag{5.1}$$

unde $1 \le i \le j \le n$. Dacă există mai multe moduri de calcul pentru T_S, atunci se presupune că i şi j au valorile cele mai mici posibile. Deoarece i este cât mai mic posibil, rezultă fie că activitatea $\sigma(i)$ este prima în σ, fie că procesorul este neocupat imedi·,· înaintea executării activităţii $\sigma(i)$. În ambele cazuri, din modul de construire a lui σ rezultă că:

$$r_{\sigma(i)} \le r_{\sigma(h)}, \text{ pentru } h = i, i+1, ..., j. \tag{5.2}$$

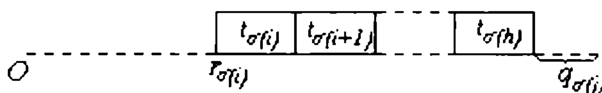


Fig. 5.14.

Dacă $d_{\sigma(j)} \le d_{\sigma(h)}$, pentru $h=i,i+1,...,j$, atunci din (5.1) şi (5.2) rezultă că σ este o secvenţă optimală. Altfel, se poate defini o activitate $\sigma(k)$ astfel încât $i \le k < j$ şi $d_{\sigma(j)} > d_{\sigma(k)}$ dar $d_{\sigma(h)} \ge d_{\sigma(j)}$, pentru $h=k+1,k+2,...,j$. Activitatea $\sigma(k)$ se numeşte în continuare *activitate de interferenţă* pentru secvenţa σ, deoarece ea poate întârzia momentele la care sunt furnizate activităţile $\sigma(k+1),\sigma(k+2),...,\sigma(j)$, ocupând procesorul atunci când măcar una din activităţile $\sigma(k+1),\sigma(k+2),...,\sigma(j)$ este furnizată spre procesare. Această activitate se numeşte *activitate critică*.

În continuare prezentăm o euristică modificată. Ideea de bază a euristicii modificate constă în aplicarea succesivă a algoritmului Schrage, cu restricţia ca activitatea de interferenţă să fie executată după activitatea critică în cadrul unei noi secvenţe. Până la urmă, oricare dintre activităţile cu timp mare de executare va înceta să fie activitate de interferenţă. Descrierea formală a euristicii este dată în continuare.

Algoritmul 5.5. (***Schrage modificat***)

1. [Iniţializare] Se face $t = 0$.
2. [Algoritm Schrage] Se aplică algoritmul Schrage obţinând timpul total $T_S^{(t)}$ şi secvenţa de planificare la execuţie $\sigma^{(t)}$.
3. [Terminare] Dacă $t = n - 1$ sau nu mai apare nici o interferenţă în programare, atunci se face $T_S = \min \{T_S^{(0)}, ..., T_S^{(t)}\}$ şi STOP.
4. [Restricţie suplimentară] Dacă $\sigma^{(t)}(k^{(t)})$ şi $\sigma^{(t)}(j^{(t)})$ sunt respectiv activităţile de interferenţă şi cea critică, se adaugă restricţia ca activitatea $\sigma^{(t)}(j^{(t)})$ să preceadă activitatea $\sigma^{(t)}(k^{(t)})$. (Dacă $j = \sigma^{(t)}(j^{(t)})$ şi $k = \sigma^{(t)}(k^{(t)})$, această restricţie este uşor de implementat făcând $r^k = r^j$).
5. [Ciclare] Se face $t = t + 1$ şi se merge la pasul 2. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Algoritmul modificat necesită $O(n^2 \log n)$ paşi.

Se poate demonstra că $T_{S'}/T^* < 3/2$, unde T^* este soluţia optimală. Se poate construi un exemplu care arată că marginea anterioară este cea mai bună pentru euristica propusă.

Exemplul 5.11. Fie un sistem de 3 activităţi cu $r_1=0$, $r_2=1$, $r_3=(p+1)/2$, $t_1=(p-1)/2$, $t_2=(p-1)/2$, $t_3=1$, $d_1=0$, $d_2=(p-3)/2$, $d_3=(p-1)/2$, unde p este un întreg şi $p\geq 3$. Algoritmul Schrage furnizează soluţia $\sigma^{(0)}=(1,2,3)$ cu $T_S^{(0)}=(3p-1)/2$. Activitatea 2 este activitatea de interferenţă şi activitatea 3 este critică. Restricţia ca activitatea 3 să preceadă activitatea 2 în $\sigma^{(1)}$ se obţine luând $r_2=r_3=(p+1)/2$. Reluarea algoritmului Schrage furnizează secvenţa $\sigma^{(1)}=(1,3,2)$, cu $T_S^{(1)}=(3p-1)/2$. La această etapă nu există nici o activitate de interferenţă astfel încât algoritmul se încheie cu $\sigma^{(1)}=(1,3,2)$. Secvenţa optimală este (2,3,1) cu timpul de operare $T^*=p+1$. Rezultă că raportul $T_{S'}/T^*=3/2-2/(p+1)$ poate fi făcut oricât de aproape de 3/2, pentru valori suficient de mari ale lui p. ■

5.2.1.5. Probleme cu resurse consumabile

Punerea problemei: Fie o mulţime cu n activităţi $A = \{1, 2, ..., n\}$ legate prin constrângeri potenţiale reprezentate printr-un graf conjunctiv $G = (X, U)$ care nu conţine arce de forma $(i, 0)$, care ar duce la imposibilitatea unei programări. Activitatea i necesită la începutul execuţiei ei la momentul s_i o cantitate a_i dintr-o resursă consumabilă. Disponibilitatea acestei resurse este dată de un grafic ce conţine momentele $u_1, u_2, ..., u_q$ în care sosesc respectiv cantităţile $b_1, b_2, ..., b_q$ din această resursă. Obiectivul este de a determina o programare a activităţilor de durată minimă în care să se ţină seama de disponibilitatea resursei. ■

Aplicaţii posibile: realizarea unei finanţări, cheltuielile unei familii, gestionarea memoriei centrale a unui calculator, rentabilizarea unor împrumuturi.

Programarea activităţilor este în acest caz o mulţime de potenţiale care satisfac condiţia de admisibilitate următoare:

$$D(t) = \sum_{i\in J(t)} a_i \leq \sum_{k\in K(t)} b_k \text{, pentru orice } t,$$

unde $J(t)$ este mulţimea activităţilor care au început înainte de momentul t şi $K(t)$ este mulţimea momentelor de sosire a resurselor precedente momentului t.

Pentru rezolvarea acestei probleme se poate aplica algoritmul următor, numit *algoritm de decalare*, în care se determină o mulţime de potenţiale calate la dreapta F, apoi se caută cea mai mică valoare $\hat{\delta}$ a lui δ astfel încât $F(\hat{\delta})$ să fie o programare.

https://biblioteca-digitala.ro / https://unibuc.ro

Algoritmul 5.6. (algoritm de decalare)

1. [Admisibilitate] Dacă $\sum_{i \in A} a_i > \sum_{k=1}^{q} b_k$, atunci STOP. (Nu există programare.)
2. [Calcul F] Se determină $F = \{f_i = l(0, n + 1) - l(i, n + 1) \mid i \in X\}$.
3. [Renumerotare] Se renumerotează activităţile în ordinea nedescrescătoare a valorilor f_i.
4. [Iniţializări] Se face $\hat{\delta}=0$, $k=1$, $D=0$, $S(u_1)=b_1$ şi $S(u_i)=S(u_{i-1})+b_i$, pentru $i=2,3,...,q$.
5. [Ciclare] Pentru $i = 1, 2, ..., n$ se aplică paşii 6 - 8, apoi STOP.
6. [Consum curent] Se face $D = D + a_i$.
7. [Moment disponibil] Atâta timp cât $S(u_k) < D$, se face $k = k+1$.
8. [Ajustare] Dacă $u_k - f_i > \hat{\delta}$, atunci se face $\hat{\delta} = u_k - f_i$, $\hat{i} = i$, $\hat{k} = k$. ■

Algoritmul de decalare determină o programare de durată minimală în $O(|U|+n\log n+q)$ operaţii dacă $G=(X,U)$ este fără circuite şi în $O(n|U|+n\log n+q)$ altfel.

Exemplul 5.12. Fie mulţimea activităţilor $A=\{1,2,3,4\}$ cu duratele de execuţie $t_1=7$, $t_2=8$, $t_3=9$ şi $t_4=6$. Activitatea 1 trebuie să preceadă activitatea 4 şi activitatea 2 trebuie să preceadă activităţile 3 şi 4. Toate activităţile necesită la începutul executării lor câte o cantitate $a_i=7$ de resurse. Cantităţile de resurse $b_1=b_2=b_3=10$ sunt disponibile la momentele $u_1=0$, $u_2=5$, $u_3=10$. Aplicând algoritmul de decalare, se calculează mulţimea de potenţiale calate la dreapta F: $f_0=0$, $f_1=4$, $f_2=0$, $f_3=8$, $f_4=11$, $f_5=17$. Apoi, pentru orice activitate i, se determină cea mai mică valoare u_k astfel încât $S(u_k)\geq D(f_i)$. Se determină $\hat{\delta}$, cea mai mică valoare astfel încât $S(f_i+\hat{\delta})\geq D(f_i)$, pentru orice i; aici $\hat{\delta}=2$. O programare de durată minimală este: $s_0=0$, $s_1=6$, $s_2=2$, $s_3=10$, $s_4=13$, $s_5=19$. ■

Teorema 5.6. (de existenţă) O condiţie necesară şi suficientă pentru a exista o programare este aceea ca graful $G = (X, U)$ să fie fără circuite de valori strict pozitive şi ca $\sum_{i \in A} a_i \leq \sum_{k=1}^{q} b_k$. ■

Fie f^* durata optimală a unei programări; u_k este un *moment critic* dacă pentru orice programare optimală τ există o sarcină i astfel încât: $s_i = u_k$; $s_i + l(i, n + 1) = f^*$; $S(u_k) \geq \sum_{j \in J} a_j > S(u_{k-1})$, unde $J = \{j \in A \mid l(j, n + 1) \geq l(i, n + 1)\}$. Activitatea i se numeşte în acest caz *activitate pivot*. Dacă $f^* = l(0, n + 1)$, atunci orice activitate pivot este şi activitate critică.

Teorema 5.7. (de criticitate) Dacă durata optimală f^* a unei programări nu este egală cu valoarea $l(0,n+1)$ a drumului critic, există un moment critic u_k şi o activitate pivot i. Într-o programare optimală, cel puţin una din activităţile pivot asociate momentului critic va începe la momentul u_k. În plus, dacă toate valorile $l(i,n+1)$ sunt distincte, activitatea pivot i este definită unic şi se execută la momentul critic u_k. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Exemplul 5.13. Pentru datele din exemplul 5.12. rezultă că $\hat{i} = 3$ este activitatea pivot şi $\hat{k} = 3$ este momentul critic. ■

5.2.1.6. Probleme cu termene de predare

Punerea problemei: Fie o mulţime cu n activităţi $A = \{1, 2, ..., n\}$ legate prin constrângeri potenţiale reprezentate printr-un graf conjunctiv $G = (X, U)$ care nu conţine arce de forma $(i, 0)$, care ar duce la imposibilitatea unei programări. Activitatea i necesită un timp de execuţie t_i, costul c_i, termenul de predare d_i şi la începutul execuţiei ei la momentul s_i o cantitate a_i dintr-o resursă consumabilă. Disponibilitatea acestei resurse este dată de un grafic ce conţine momentele u_1, u_2, ..., u_q în care sosesc respectiv cantităţile b_1, b_2, ..., b_q din această resursă. Obiectivul este de a determina o programare a activităţilor, dacă există, de durată minimă în care să se ţină seama de disponibilitatea resursei. ■

Aplicaţii posibile: realizarea unei finanţări, cheltuielile unei familii, gestionarea memoriei centrale a unui calculator, rentabilizarea unor împrumuturi.

Se numeşte *graf complet* graful $\overline{G}$ obţinut din G prin introducerea arcelor $(i, 0)$ ponderate cu $t_i - d_i$.

Înainte de a determina soluţia problemei sunt ajustate valorile d_i în aşa fel încât să fie îndeplinită condiţia $d_i \le \min_{k \in U^+(i)} (d_k - v_{ik} - t_k) + t_i$. Aceasta se poate face printr-un algoritm de determinare a drumurilor din graful G. Complexitatea acestui algoritm este $O(|U|)$, pentru un graf fără circuite, şi $O(n^3)$, pentru un graf cu circuite.

În acest caz, mulţimea potenţialelor calate la dreapta $F(\delta)$ este dată de:

$$F(\delta) = \{f_i(\delta) = \min(f_i + \delta, d_i - t_i) \mid i \in X\},$$

unde $f_i = l(0, n + 1) - l(i, n + 1)$. Pentru un δ suficient de mare, $F(\delta) = \{d_i - t_i \mid i \in X\}$. Prin abuz de notaţie, vom nota această mulţime de potenţiale cu $F(\infty)$.

Teorema 5.8. (de existenţă) O condiţie necesară şi suficientă pentru a exista o programare este ca graful complet $\overline{G} = (X, \overline{U})$ să fie fără circuite absorbante şi ca mulţimea potenţialelor $F(\infty) = \{d_i - t_i \mid i \in X\}$ să fie o planificare. ■

Se numeşte *latitudinea* unei activităţi i cantitatea $\delta_i = d_i - f_i - t_i$. Valoarea δ_i reprezintă cantitatea maximală cu care poate să întârzie activitatea fără să depăşească termenul său de predare. În algoritmul următor, numit *algoritmul dicotomic*, se numerotează activităţile în ordinea nedescrescătoare în raport cu δ_i şi se face o căutare dicotomică pentru a determina în ce interval $[\delta_i, \delta_{i+1})$ se găseşte δ astfel încât $F(\delta)$ să fie optimal.

Algoritmul 5.7. (***dicotomic***)

https://biblioteca-digitala.ro / https://unibuc.ro

1. [Calcul F] Se determină $F=\{f_i=l(0,n+1)-l(i,n+1) \mid i\in X\}$ şi $F(\infty) = \{d_i - t_i \mid i\in X\}$.
2. [Realizabil] Dacă $F(\infty)$ nu este admisibil, atunci STOP. (Nu există programare).
3. [Renumerotare] Se renumerotează activitățile în ordinea nedescrescătoare a valorilor δ_i, unde $\delta_i = d_i - f_i - t_i$.
4. [Inițializări] Se face $j = 0$, $k = n$, $i = [(k + j) / 2]$ şi $b = 0$.
5. [Algoritm decalare] Se aplică algoritmul de decalare pentru $F(0)$.
6. [Testare soluție] Dacă $\hat{\delta} \leq \delta_1$, atunci se face $\hat{\delta} = \delta_1$ şi $b = 0$.
7. [Terminare] Dacă $b = 1$, atunci STOP.
8. [Algoritm decalare] Se aplică algoritmul de decalare pentru $F(\delta_i)$.
9. [Mutare în jos] Dacă $\hat{\delta} = 0$, atunci se face $k = i$ şi $i = [(k + j) / 2]$.
10. [Mutare în sus] Dacă $\hat{\delta} + \delta_i > \delta_{i+1}$, atunci se face $j = i$ şi $i = [(k + j) / 2]$.
11. [Determinare interval] Dacă $\hat{\delta} > 0$ şi $\hat{\delta} + \delta_i \leq \delta_{i+1}$, atunci se face $b=1$ şi $\delta = \hat{\delta} + \delta_i$.
12. [Ciclare] Se merge la pasul 7. ■

Se poate demonstra că algoritmul dicotomic determină o programare ori de câte ori există o programare pentru o problemă dată. Complexitatea lui este $O(|U| + n \log n + q \log n)$, dacă graful G este fără circuite şi $O(|U| + q \log n)$, altfel.

5.2.1.7. Metoda PERT

În metoda *PERT* (***P**rogram **E**voluation and **R**esearch **T**echnique*) se poate folosi pentru determinarea drumurilor de valoare maximală un graf de precedență în care nodurile sunt asociate cu evenimente de tipul *terminarea activității i, începutul sarcinilor j şi k* şi arcele sunt asociate activităților. Activitățile pot să fie reale sau fictive. Activitățile fictive sunt utilizate pentru a reprezenta anumite constrângeri potențiale şi au, în general, durata 0.

Un inconvenient al acestei metode în raport cu metoda potențialelor este numărul dublu de noduri din graf. Aceasta deoarece fiecărei activități i îi corespunde un nod I_i pentru începutul execuției şi un nod F_i pentru terminarea ei. Un avantaj al acestei metode este o mai uşoară urmărire de către practicieni, fiecare activitate putând să fie reprezentată printr-un singur segment ce poate avea o lungime proporțională cu timpul de execuție. Schemele ce se obțin pentru graful asociat pot să arate sub forma unor diagrame Gantt.

Pentru reflectarea modului în care sunt reflectate pe graful G constrângerile potențiale vom da în continuare câteva exemple. Menționăm că rădăcina este notată I şi antirădăcina F.

https://biblioteca-digitala.ro / https://unibuc.ro

- Activitatea i precede activitatea j se traduce prin adăugarea arcului (F_i, I_j) de valoare 0.
- Activitatea i începe după r_i unități se traduce prin adăugarea arcului (I, I_i) de valoare r_i.
- Activitatea j poate să înceapă numai după ce s-a consumat 2/3 din timpul de execuție al activității i se traduce prin înlocuirea lui i prin două activități i' ce reprezintă prima parte din i de timp 2/3 din timpul lui i și i'' ce reprezintă ultima parte din i de timp 1/3 din timpul lui i. Apoi se înlocuiesc arcele de forma $(\bullet, I_i)$ cu arce $(\bullet, I_{i'})$ și arcele $(F_i, \bullet)$ cu $(F_{i''}, \bullet)$, de aceleași valori, și se adaugă arcele $(F_{i'}, I_{i''})$ de valoare 0 și $(F_{i'}, I_j)$ de valoare 0.
- Activitatea j începe la un interval t' după terminarea activității i se traduce prin includerea unui arc (F_i, I_j) de valoare t'.
- Activitatea i nu se poate termina după termenul d_i se traduce prin includerea unui arc (F_i, F) de valoare $-d_i$.
- Activitatea i să înceapă înainte de momentul t' se traduce prin includerea unui arc (I_i, I) de valoare $-t'$.

Graful obținut în urma transformărilor de tipul celor indicate mai sus se poate simplifica prin eliminarea arcelor redondante.

5.2.2. Metoda circuitelor critice

Punerea problemei: Se cere determinarea unei programări a mulțimii de activități $A = \{1, 2, ..., n\}$, fiecare activitate putându-se executa repetitiv, supuse la constrângeri interne date prin relații de precedență între activități și la constrângeri externe prin care se stabilesc relații între copiile activităților. ■

Aplicații posibile: fabricarea unor elemente de serie, executarea programelor.

Execuția repetitivă a activităților poate fi asimilată cu calculul de mai multe ori a unei funcții $F = (A, <)$ care stabilește ordinea de executare a câte unei copii a fiecărei activități. Vom nota cu $<$ mulțimea constrângerilor interne și cu $<'$ mulțimea finită de constrângeri externe. O constrângere externă este de forma (i, j, k), unde i și j sunt activități *părinți* și k este un număr întreg strict pozitiv, și exprimă faptul că, pentru orice n număr natural, execuția activității j pentru al $(n + k)$-lea calcul al funcției nu poate să înceapă înainte de terminarea execuției lui i pentru al n-lea calcul al funcției. Pentru a evita reentranța se face presupunerea că $(i, i, 1) \in <'$, pentru orice $i \in A$. Aceasta este cunoscută sub numele de *ipoteza de nereentranță*.

https://biblioteca-digitala.ro / https://unibuc.ro

Dacă se notează cu t_i^n momentul de începere a celei de-a n-a execuții a activității i, atunci a n-a execuție a lui F are loc între momentul α_n dat de expresia $\alpha_n=\min\{t_i^n|i\in A\}$ și momentul β_n dat de expresia $\beta_n=\max\{t_i^n|i\in A\}$. De cele mai multe ori se presupune că orice execuție a lui F se efectuează într-un timp finit. Aceasta se poate exprima prin existența unei constante K astfel încât $\beta_n-\alpha_n<K$. Se poate vedea ușor că, pentru fiecare $i\in A$, are loc inegalitatea $t_i^n \le t_i^{n+1}$ pentru orice n număr natural. La fel, șirurile $(\alpha_n)_n$ și $(\beta_n)_n$ sunt crescătoare. Ele sunt strict crescătoare dacă $t_i > 0$.

În continuare vom numi *activitate derivată* de ordinul n a activității părinte i perechea (i, n), pentru un n număr natural nenul.

Dacă F se calculează de un număr dat N de ori, problema se poate rezolva prin metodele prezentate anterior, considerând N copii identice ale grafului ociat și stabilind arce între activități din copii diferite în funcție de constrângerile externe.

5.2.2.1. Modelarea problemei centrale repetitive

Vom utiliza noțiunea de *graf de evenimente* definit prin $R=(G,p,E^0)$, unde $G=(T,P)$ este un graf direcționat conex în care T este mulțimea tranzițiilor și P este mulțimea locurilor; p: $T \to$ N asociază durate tranzițiilor și $E^0 = (M^0 = P \to$ N, $R^0 = T \to$ N) este starea inițială. Vom presupune că $t_i>0$ și că duratele reziduale sunt 0 la momentul 0. De aici rezultă că postmarcajul inițial este egal cu marcajul inițial.

Fie funcția $F = (A, <)$ cu constrângerile externe <‘. Vom asocia problemei centrale repetitive graful de evenimente temporizat următor: fiecărei activități i de durată t_i îi corespunde tranziția T_i de aceeași durată și de durată reziduală inițială nulă; fiecărei precedențe $i < j$ îi corespunde locul P_{ij} de marcă inițială nulă, cu unica intrare T_i și unica ieșire T_j; fiecărei constrângeri externe $(i, j, k) \in$ <‘ îi corespunde un loc P_{ij} de marcă inițială k, având o intrare T_i și o ieșire T_j. Execuțiile controlate i. inite ale grafului de evenimente $R = (G, p, E^0)$ sunt soluții pentru problema centrală repetitivă.

Exemplul 5.14. Pentru realizarea unei piese se efectuează activitățile {1,2,...,12} repartizate în trei ateliere A_1, A_2 și A_3. Limitările interne sunt date în figura 5.15. Constrângerile externe sunt implicate de restricția că fiecare atelier A_i poate lucra la cel mult K_i piese la un moment dat, unde $K_1=3$, $K_2=2$, $K_3=1$ și de ipoteza de nereentranță. Aceste constrângeri externe se traduc în satisfacerea următoarelor inegalități: $t_1^{n+3}-t_6^n \ge t_6$ (primul atelier poate lucra la cel mult 3 piese la un moment dat), $t_7^{n+2}-t_{10}^n \ge t_{10}$ (al doilea atelier poate lucra la cel mult 2 piese la un moment dat), $t_{11}^{n+1}-t_{12}^n \ge t_{12}$ (al treilea atelier poate lucra la cel mult o piesă la un moment dat),

https://biblioteca-digitala.ro / https://unibuc.ro

$t_i^{n+1}-t_i^n \ge t_i$, pentru orice $n > 0$ şi orice activitate i (piesele se pot lucra numai în ordinea numerelor asociate lor). Graful de evenimente corespunzător procedeului expus anterior este reprezentat în figura 5.16. ■

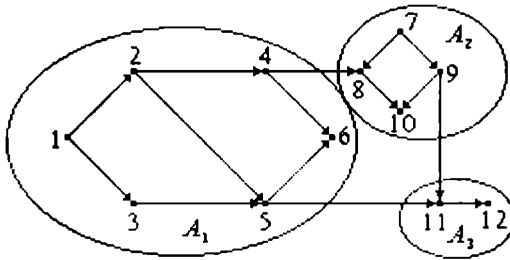


Fig. 5.15.

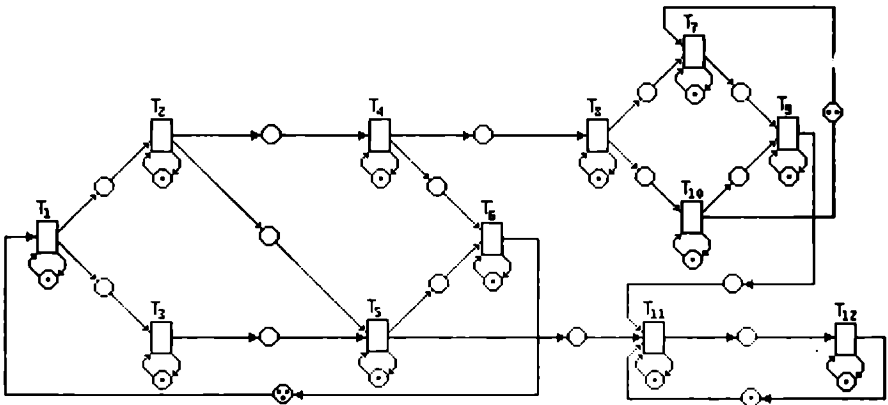


Fig. 5.16

Din graful de evenimente $R=(G,p,E^0)$ se poate construi un graf infinit, numit *graful dezvoltat*, $g(R)=(\tau,S,\nu)$, unde $\tau= T_0^0\cup\{T_i^n \mid T_i\in T, n\ge 1\}$, T_0^0 fiind nodul asociat activităţii fictive de început (rădăcina) şi T_i^n fiind nodul asociat activităţii derivate (i,n), $S=S_1\cup S_2$, cu $S_1=\{(T_0^0,T_j^{M_{ij}^0}) \mid (T_i,T_j)\in P$ şi $M_{ij}^0>0\}$, $S_2=\{(T_i^n,T_j^{n+M_{ij}^0}) \mid (T_i,T_j)\in P$ şi $n\ge 1\}$, M_{ij}^0 fiind marcajul iniţial al lui P_{ij}; $\nu(T_0^0,T_j^{M_{ij}^0})=0$, pentru orice arc din S_1 şi $\nu(T_i^n,T_j^{n+M_{ij}^0})=t_i$, pentru fiecare arc din S_2. Traversarea de indice $n+M_{ij}^0$ a lui T_j poate să înceapă după a n-a traversare a lui T_i.

Fie un element ω pe care îl adăugăm la mulţimea numerelor naturale obţinând mulţimea $\overline{\mathrm{N}} = \mathrm{N}\cup\{\omega\}$. Presupunem că ω are următoarele proprietăţi: $\omega > n$, pentru orice număr natural n, $\omega+n=\omega-n=\omega$, pentru orice număr natural n, $\omega+\omega=\omega-\omega=\omega$.

https://biblioteca-digitala.ro / https://unibuc.ro

O *mulţime de potenţiale* pe $g(R)$, graful extins al grafului de evenimente temporizate R, este o aplicaţie s a mulţimii τ a vârfurilor lui $g(R)$ cu valori reale cu următoarele proprietăţi: $s_0^0 = 0$; $s_i^n \in \overline{\mathrm{N}}$, pentru orice $n \geq 1$ şi orice $T_i \in T$: $s_j^q - s_i^p \geq v(T_i^p, T_j^q)$, pentru orice $(T_i^p, T_j^q) \in S$. Există o bijecţie între execuţiile controlate ale lui R şi mulţimile de potenţiale pe $g(R)$. Un graf de evenimente temporizate are o execuţie controlată infinită dacă şi numai dacă pentru orice circuit suma marcajelor iniţiale este strict pozitivă.

Soluţia optimală a problemei de programare repetitivă este execuţia controlată cea mai devreme. Relaţia de ordine între execuţii ordonate se defineşte în felul următor: dacă X şi Y sunt execuţii controlate având respectiv asociate mulţimile de potenţiale x şi y, atunci $X \leq Y$ dacă şi numai dacă $x_i^n \leq y_i^n$, pentru orice $T_i \in T$ şi orice $n \geq 1$.

Data cea mai devreme a celei de-a n-a execuţii a activităţii părinte i a unei probleme de programare repetitivă este valoarea maximală a unui drum de la T_0^0 la T_i^n în graful extins.

Notând cu γ_i^n momentul cel mai devreme a celei de-a n-a traversare a tranziţiei T_i, se obţine $\gamma_i^n = \omega$ dacă există în $g(R)$ un drum de valoare ∞ de la T_0 la T_i^n şi γ_i^n = valoarea maximă a unui drum de la T_0^0 la T_i^n.

Fie C mulţimea tranziţiilor ce aparţin circuitelor cu toate arcele marcate iniţial cu zero. Fie A^p mulţimea nodurilor lui $g_p(R)$ care rezultă din C, dată de expresia $A^p = \{T_i^p \mid T_i \in C\}$. Dacă τ^p este mulţimea nodurilor lui $g_p(R)$, atunci graful $g'_p(R)$ cu nodurile $\tau^p - A^p$ şi arcele corespunzătoare lor din $g_p(R)$ este fără circuite. Deci acest graf admite o numerotare a nodurilor astfel încât pentru orice arc (T_i^p, T_j^p) din $g'_p(R)$ are loc $i < j$. Această numerotare permite o ordonare a tuturor nodurilor din copii ce nu aparţin la circuite şi determinarea potenţialelor până la o limită M de copii date aplicând algoritmul următor:

Algoritmul 5.8. (*potmin*)

1. [Iniţializare] Se face $\gamma_0^0 = 0$ şi $\gamma_i^1 = \omega$, pentru orice $T_i^1 \in C^1$.
2. [Ciclare copii] Pentru $p = 1, 2, \ldots, M$ se aplică pasul 3, apoi STOP.
3. [Ciclare noduri] Pentru $i = 1, 2, \ldots, n$ se aplică pasul 4.
4. [Calcul potenţial] Dacă $T_i^p \in C^p$, atunci $\gamma_i^p = \omega$, altfel:

$$\gamma_i^p = \max_{\left(T_j^q, T_i^p\right) \text{ arc în } g(R)} \left\{\gamma_j^q + v\left(T_j^q, T_i^p\right)\right\}. \blacksquare$$

Complexitatea acestui algoritm este $O(n^2 M)$.

https://biblioteca-digitala.ro / https://unibuc.ro

5.2.3. Metode clasice

În acest paragraf vom prezenta metodele devenite clasice pentru programarea activităţilor. Am clasificat aceste metode în două categorii: metode de schimb şi metode pe bază de liste.

5.2.3.1. Metode de schimb

5.2.3.1.1. *Programarea pe un procesor*

Punerea problemei. Fiind date n activităţi disponibile la momentul 0, cu timpii de execuţie $t_1, t_2, ..., t_n$, cu ponderile $w_1, w_2, ..., w_n$ se cere programarea lor pe un procesor prin care să se minimizeze valoarea $\sum_{i=1}^{n} w_i c_i$, unde c_i este momentul terminării activităţii i. ■

Dacă activităţile sunt independente, programarea activităţilor în ordinea necrescătoare a valorilor $\rho_i = w_i / t_i$ conduce la o soluţie optimă.

5.2.3.1.1.1. Proprietăţi ale soluţiilor optime pentru activităţi dependente

Pentru o submulţime x de activităţi vom folosi următoarele notaţii: $t(x) = \sum_{i \in x} t_i$, $w(x) = \sum_{i \in x} w_i$, $\rho(x) = w(x) / t(x)$, $G(x)$ subgraful generat de x şi $\bar{x} = A - x$.

Se vede imediat că dacă s este o programare pentru A, atunci $s(x)$ este o programare pentru x.

Vom spune că o submulţime de activităţi x este *iniţială* dacă pentru orice $i \in x$ predecesorii lui i sunt elemente ale lui x.

Oricare ar fi soluţia problemei de programare $s = (i_1, i_2, ..., i_n)$, mulţimile $\{i_1, i_2, ..., i_k\}$ sunt iniţiale pentru orice $k = 1, 2, ..., n$.

Vom spune că o submulţime de activităţi y este *ρ-maximală* dacă este nevidă, $\rho(y) \geq \rho(x)$, pentru orice submulţime x iniţială, şi este minimală în raport cu incluziunea cu aceste proprietăţi.

Teorema 5.9. Dacă $s^* = (s^*(x), s^*(\bar{x}))$ este o programare optimală, atunci $s^*(x)$ este soluţie optimală pentru $G(x)$ şi $s^*(\bar{x})$ este soluţie optimală pentru $G(\bar{x})$.

Teorema 5.10. Dacă y este iniţială ρ-maximală, atunci există o programare optimală s^* astfel încât $s^* = (s^*(y), s^*(\bar{y}))$. În plus, y apare ca un şir neîntrerupt în orice soluţie optimală. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Lema 5.2. Dacă (x_1, x_2) este o partiție a lui x, atunci:

$$\rho(x) = (t(x_1) / t(x))\, \rho(x_1) + (t(x_2) / t(x))\, \rho(x_2). \blacksquare$$

Lema 5.3. Dacă $s=(s(x),s(x_1),s(x_2),s(y))$ și $s'=(s(x),s(x_2),s(x_1),s(y))$ sunt programări de valori $\varphi(s)$, respectiv $\varphi(s')$, atunci $\varphi(s)\leq\varphi(s')$ dacă și numai dacă $\rho(x_1)\geq\rho(x_2)$. ■

Lema 5.4. Dacă $(x_1, x_2, ..., x_r)$ este o partiție a lui x, atunci $\max\{\rho(x_1), \rho(x_2), ..., \rho(x_r)\} \geq \rho(x)$. Egalitatea are loc dacă și numai dacă $\rho(x_1) = \rho(x_2) = ... = \rho(x_r)$. ■

Teorema 5.11. Dacă s^* este o programare optimală, atunci există o mulțime inițială ρ-maximală de activități y astfel încât $s^* = (s^*(y), s^*(\bar{y}))$. ■

Din proprietățile precedente rezultă următorul procedeu de determinare a unei soluții optimale: se determină mulțimile inițiale ρ-maximale de activități y și se alege soluția $(s(y), s(\bar{y}))$ care are cea mai mică valoare.

5.2.3.1.1.2. Dependențe de tip antiarborescență

Reamintim că un *antiarbore* este un graf conex cu un nod numit *antirădăcină* care nu are nici un succesor, toate celelalte noduri având câte un succesor imediat. G este o *familie de antiarbori* dacă orice componentă conexă a lui G este un antiarbore. Vom nota cu $A(i)$ antiarborele generat de i.

Teorema 5.12. Mulțimile de activități inițiale ρ-maximale ale unei familii de antiarbori sunt $A(i)$, $i = 1, 2, ..., n$. ■

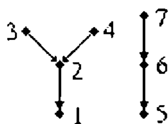


Fig. 5.17.

i	1	2	3	4	5	6	7
$A(i)$	{1, 2, 3, 4}	{2, 3, 4}	{3}	{4}	{5, 6, 7}	{6, 7}	{7}
$w(A(i))$	15	9	2	3	13	7	3
$t(A(i))$	7	6	1	3	5	4	2
$\rho(A(i))$	2 + 1/7	1 + 1/2	2	1	2 + 3/5	1 + 3/4	1 + 1/2

Tabelul 5.3.

Exemplul 5.15. Fie mulțimea de activități $A = \{1, 2, 3, 4, 5, 6, 7\}$ cu timpii de execuție (1, 2, 1, 3, 1, 2, 2), cu ponderile (6, 4, 2, 3, 6, 4, 3) și cu graful de precedență din figura 5.17. În tabelul 5.3. sunt sintetizate informațiile privind mulțimile de activități inițiale corespunzătoare. Rezultă că mulțimea $A(5)$ este ρ-maximală. Deci $s^* = (s^*(A(5)), s^*(\overline{A(5)}))$. Apoi se determină componentele acestei soluții după modelul $s^*(A(5)) = (s^*(A(5) - \{5\}), 5)$. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Algoritmul de determinare a soluţiei optimale este în acest caz:

Algoritmul 5.9. ***OPT*(*A*)**

1. [Terminare] Dacă $A = \varnothing$, atunci STOP.
2. [Prefix soluţie] Se planifică activităţile date de soluţia $OPT(A(1) - \{1\})$.
3. [Planificare centrală] Se planifică prima activitate din A.
4. [Sufix soluţie] Se planifică activităţile date de soluţia $OPT(\overline{A(1)})$. ■

Complexitatea acestui algoritm este $O(n^2)$.

Exemplul 5.16. Cu datele din exemplul 5.15., deoarece pentru $A(5)$ există o singură planificare, $s^*(A(5)) = (7, 6, 5)$. Cum $\overline{A(5)} = A(1)$, soluţia optimală este $(7, 6, 5, s^*(A(2)), 1)$. $A(3)$ este ρ-maximală pentru $A(2)$ şi, cum $\overline{A(3)} = \{4, 2\}$, rezultă soluţia optimală a problemei iniţiale (7, 6, 5, 3, 4, 2, 1). ■

5.2.3.1.1.3. Euristică Greedy pentru activităţi cu termene de predare

Punerea problemei. Presupunem executarea pe un procesor a n activităţi, activitatea i având timpul de execuţie t_i, ponderea w_i şi termenul de predare d_i, pentru $i = 1, 2, ..., n$. ■

În euristica propusă, se programează pe ultimul loc activitatea cu valoare maximă a raportului t_i / w_i şi care nu depăşeşte termenul limită de încheiere d_i. Procedura se reia până când s-a efectuat programarea tuturor activităţilor. Descrierea formală este dată în continuare.

Algoritmul 5.10. **(*Greedy*)**

1. [Iniţializări] Se face $k = n$ şi $T_n = \sum_{i=1}^{n} t_i$, $V_n = \{1, 2, ..., n\}$.
2. [Terminare] Se determină $E_k = \{j \in V_k \mid d_j \geq T_k\}$. Dacă $E_k = \varnothing$, atunci nu există o soluţie realizabilă şi STOP.
3. [Alegere] Se alege $r_k \in \{i \mid t_i / w_i = \max_{j \in E_k} \{t_j / w_j\}\}$ cu d_k cât mai mare. În cazul unor valori maxime egale ale rapoartelor şi ale limitelor de încheiere corespunzătoare lor d_k, se alege aleator una dintre valorile r_k.
4. [Modificări] Dacă $k = 1$, atunci STOP; altfel se face $T_{k-1} = T_k - t_{r_k}$, $V_{k-1} = V_k - \{r_k\}$, $k = k - 1$ şi se trece la pasul 2. ■

Această procedură are complexitatea temporală $O(n \log n)$ şi identifică o soluţie realizabilă, dacă aceasta există.

În vederea investigării eficienţei euristicii propuse, fie o clasă de probleme pentru care euristica furnizează o soluţie optimală. Se presupune că activităţile sunt

https://biblioteca-digitala.ro / https://unibuc.ro

indexate astfel încât $t_1 / w_1 \le t_2 / w_2 \le ... \le t_n / w_n$. Pentru o programare dată σ se afirmă că activitățile $\sigma(i)$, $\sigma(j)$ și $\sigma(k)$ sunt într-o configurație interzisă dacă $i < j < k$ și $\sigma(j) > \sigma(i) > \sigma(k)$. Dacă σ are trei activități, nu neapărat adiacente, într-o configurație interzisă, atunci activitatea situată în poziția mijlocie, activitatea $\sigma(j)$ din poziția j, are valoarea maximă a raportului t_r / w_r, activitatea din prima poziție, activitatea $\sigma(i)$, are valoarea imediat mai mică a raportului t_r / w_r și activitatea din ultima poziție, activitatea $\sigma(k)$, are cea mai mică valoare a raportului t_r / w_r.

Teorema 5.13. Fie σ^* o soluție optimală a problemei studiate. Dacă în σ^* nu există o configurație interzisă a trei activități din σ^*, atunci soluția σ^0, adică cea furnizată de euristică, este și ea optimală.

Demonstrație. Să presupunem că $\sigma^0 \neq \sigma^*$. Fie $s = \max\{k | \sigma^0(k) \neq \sigma^*(k)\}$. În virtutea pasului 3 al euristicii, $\sigma^*(s) < \sigma^0(s)$. Fie $i \in \{1,2,..,s-1\}$ astfel încât $\sigma^*(i) = \sigma^0(s)$. În virtutea ipotezei $\sigma^*(j) < \sigma^*(i)$, pentru toți $j \in \{i+1, i+2, ..., s-1\}$. O succesiune de interschimbări de perechi de intervale are drept rezultat o soluție cel puțin la fel de bună cu activitatea $\sigma^*(i)$ inserată în poziția s. Procedeul descris poate fi repetat până când $\sigma^0 = \sigma^*$. Deci σ^0 este optimală. ■

5.2.3.1.2. *Probleme de atelier cu două procesoare*

În problemele de programare cu m procesoare numite *flowshop* se presupun un număr de n lucrări, fiecare lucrare i constă din m activități $(i, 1)$, $(i, 2)$, ..., (i, m), activitatea (i, j) se execută timp de t_{ij} pe procesorul j și (i, j) precede $(i, j + 1)$, pentru $j = 1, 2, ..., m - 1$ și $i = 1, 2, ..., n$.

Pentru $m = 2$, vom nota $a_i = t_{i1}$ și $b_i = t_{i2}$, pentru $i = 1, 2, ..., n$. Fie relația ρ pe mulțimea lucrărilor definită prin $i \rho j$ dacă și numai dacă $\min(a_i, b_j) < \min(a_j, b_i)$.

Teorema 5.14. Relația ρ este o relație de ordine parțială strictă.

Pentru demonstrație a se vedea [BÂS89]. ■

Teorema 5.15. Orice ordine totală compatibilă cu ρ este o planificare optimală a lucrărilor executate pe două procesoare diferite.

Pentru demonstrație a se vedea [BÂS89]. ■

Proprietățile enunțate anterior permit obținerea unei planificări optimale aplicând următorul algoritm cunoscut sub numele de *algoritmul lui Johnson.*

Algoritmul 5.11. (***Johnson***)

1. [Inițializări] Se face $R = \{1, 2, ..., n\}$, $s = 1$ și $d = n$.
2. [Terminare] Dacă $R = \varnothing$, atunci STOP.

https://biblioteca-digitala.ro / https://unibuc.ro

3. [Alegere] Se alege o pereche (i, j) care corespunde celei mai mici valori [illegible]$_{ij}$, pentru $i \in R$ şi $j = 1, 2$. Dacă sunt mai multe, se alege oricare dintre ele.
4. [Programare] Dacă $j = 1$, atunci se planifică i pe poziţia s şi se face $s = s + 1$, altfel se planifică i pe poziţia d şi se face $d = d - 1$.
5. [Ciclare] Se face $R = R - \{i\}$ şi se merge la pasul 2. ■

Complexitatea algoritmului este $O(n^2)$.

Exemplul 5.17. Pentru n=5, a_1=3, b_1=6, a_2=5, b_2=2, a_3=1, b_3=2, a_4=6, b_4=6, a_5=7 şi b_5=5, prin aplicarea algoritmului precedent, mai întâi se face R={1,2,3,4,5}, s=1 şi d=5. Cum a_3=1 este cea mai mică valoare a unui timp, se plasează lucrarea 3 pe locul 1, se face s=2 şi R={1,2,4,5}. Pentru activităţile rămase b_2=2 este cea mai mică valoare a unui timp, deci se plasează lucrarea 2 pe locul 5, se face d=4 şi R={1,4,5}. Pentru activităţile rămase a_1=3 este cea mai mică valoare a unui timp, deci se plasează lucrarea 1 pe locul 2, se face s=3 şi R={4,5}. Pentru activităţile rămase b_5=5 este cel mai mic timp, deci se plasează lucrarea 5 pe locul 4, se face d=3 şi R={4}. În final se plasează lucrarea 4 pe locul 3 şi se obţine planificarea optimă (3,1,4,5,2). ■

5.2.3.1.3. *Euristici de planificare pe două maşini cu timpi de accesibilitate*

Fie n activităţi cu timp de accesibilitate r_i, $i = 1, 2, ..., n$ şi durate de procesare a_i şi b_i pe maşinile A şi respectiv B. Obiectivul constă în planificarea (ordonarea) activităţilor astfel încât timpul maxim de încheiere C_{max} să fie minimizat. Pentru rezolvarea problemei, se propun patru euristici pe care le prezentăm în continuare.

Prima euristică, numită *ARB*, constă în ordonarea arbitrară a activităţilor, după care C_{max}^{ARB} este evaluat în $O(n)$ paşi. În a doua euristică, numită R, activităţile sunt ordonate în ordinea nedescrescătoare a timpilor de accesibilitate şi în a treia euristică, numită J, ordonarea are loc în conformitate cu *regula Johnson*, ignorând timpii de accesibilitate, şi anume: activităţile cu $a_i \le b_i$ sunt ordonate primele în ordinea nedescrescătoare a duratelor a_i după care activităţile rămase (cu $a_i > b_i$) sunt ordonate în ordinea necrescătoare a duratelor b_i. În ambele cazuri, procesarea se efectuează în $O(n \log n)$ paşi. Dacă e adoptată euristica R, atunci nu există timp neocupat în mod nenecesar pe maşina A. A patra euristică, numită *RJ*, este o variantă a euristicii R în care se ţine seama de avantajul euristicii J de a elimina timpii neocupaţi în mod nenecesar: ori de câte ori trebuie aleasă o activitate dintr-o secvenţă care evită timpii neocupaţi în mod nenecesar, se alege acea activitate care urmează să fie procesată prima în conformitate cu regula lui Johnson. Această euristică are complexitatea $O(n \log n)$ şi este prezentată în continuare.

https://biblioteca-digitala.ro / https://unibuc.ro

Algoritmul 5.12. (Euristica *RJ*)

1. [Inițializări] Se face $k = 0$ și $T = \min_{i \in S}\{r_i\}$, unde S este mulțimea activităților.
2. [Alegeri] Se determină mulțimea $S' = \{i \mid i \in S, r_i \leq T, a_i \leq b_i\}$ și mulțimea $S'' = \{i \mid i \in S, r_i \leq T, a_i > b_i\}$. Dacă $S' \neq \varnothing$, se alege o activitate i din S' cu a_i cât mai mic; altfel se alege o activitate i din S'' cu b_i cât mai mare.
3. [Plasare] Se face $k = k + 1$, se plasează activitatea i în poziția k, se face $T = T + a_i$ și $S = S - \{i\}$.
4. [Terminare sau ciclare] Dacă $S = \varnothing$, atunci STOP (s-a obținut o programare); altfel se face $T = \max\{T, \min_{i \in S}\{r_i\}\}$ și se trece la pasul 2. ■

Dacă o euristică H generează o secvență $(\sigma(1), ..., \sigma(n))$, atunci timpul de execuție maxim al euristicii H este:

$$C_{\max}^{H} = r_{\sigma(u)} + \sum_{i=u}^{v} a_{\sigma(i)} + \sum_{i=v}^{n} b_{\sigma(i)}, \tag{5.3}$$

pentru anumiți $u, v \in \{1, 2, ..., n\}$, unde $u \leq v$, și u este ales cât mai mic posibil.

În continuare se prezintă performanțele celor patru euristici în cel mai rău caz. Fie $C_{\max}^{*}$ valoarea minimă a timpului maxim de încheiere a unei procesări.

Teorema 5.16. $C_{\max}^{ARB}/C_{\max}^{*}<3$, $C_{\max}^{R}/C_{\max}^{*}<2$, $C_{\max}^{J}/C_{\max}^{*}<2$ și $C_{\max}^{RJ}/C_{\max}^{*}<2$; aceste margini sunt cele mai bune posibil.

Demonstrație. De fiecare dată, se presupune că secvența generată este $(\sigma(1), \sigma(2), ..., \sigma(n))$ și timpul maxim de execuție este dat de (5.3). Evident că $C_{\max}^{*}$ este mai mare decât orice timp de accesibilitate, adică:

$$C_{\max}^{*} > r_{\sigma(u)}. \tag{5.4}$$

Timpul necesar pentru procesarea activităților $\{\sigma(u), \sigma(u+1), ..., \sigma(v)\}$ pe mașina A satisface inegalitatea:

$$C_{\max}^{*} > \sum_{i=u}^{v} a_{\sigma(i)}. \tag{5.5}$$

În mod analog, timpul necesar pentru procesarea activităților $\{\sigma(u), \sigma(u+1), ..., \sigma(v)\}$ pe mașina B satisface inegalitatea:

$$C_{\max}^{*} > \sum_{i=u}^{v} b_{\sigma(i)}. \tag{5.6}$$

Pentru euristica ARB, însumând (5.4), (5.5) și (5.6) și ținând seama de (5.3) rezultă $3C_{\max}^{*} > C_{\max}^{ARB}$. Pentru euristicile R și RJ, valoarea minimă a timpului de accesibilitate a activităților $\{\sigma(u), \sigma(u+1), ..., \sigma(v)\}$ este $r_{\sigma(u)}$. Rezultă că:

$$C_{\max}^{*} > r_{\sigma(u)} + \sum_{i=u}^{v} a_{\sigma(i)}. \tag{5.7}$$

Adunând (5.6) cu (5.7) și ținând seama de (5.3), rezultă $2C_{\max}^{*}>C_{\max}^{R}$ și $2C_{\max}^{*}>C_{\max}^{RJ}$.

https://biblioteca-digitala.ro / https://unibuc.ro

În euristica J, activitățile din $\{\sigma(u), \sigma(u+1), ..., \sigma(v)\}$ sunt ordonate în conformitate cu regula lui Johnson. Timpul maxim de execuție, ignorând timpii de accesibilitate, satisface inegalitatea:

$$C^*_{\max} > \sum_{i=u}^{v} a_{\sigma(i)} + \sum_{i=u}^{v} b_{\sigma(i)}. \tag{5.8}$$

Adunând (5.4) și (5.8) și ținând seama de (5.3), rezultă $2C^*_{\max} > C^J_{\max}$. **Faptul că** limitele date sunt cele mai mici posibile rezultă din următorul exemplu. ■

Exemplul 5.18. Fie problema cu trei lucrări specificată cu datele **din tabelul** 5.4., unde $0 < 8k < K$.

i	1	2	3
r_i	k	0	K-3k
a_i	2k	K-6k	k
b_i	K-6k	k	2k

Tabelul 5.4.

Pentru secvența (1,2,3) rezultă timpul de execuție $K=C^*_{\max}$. Într-adevăr, lucrarea 3 are timpul de accesibilitate K-3k și timpul de execuție este $a_3+b_3=3k$, deci încheierea procesării lucrărilor nu poate fi mai mică decât $K-3k+3k=K$. Dacă lucrările sunt ordonate arbitrar în ordinea (3,2,1) se obține $C^{ARB}_{\max}=3K-12k$, deci $C^{ARB}_{\max}/C^*_{\max}$ = **3-12*k*/*K*** și are o valoare oricât de apropiată de 3. Pentru euristicile R și RJ se obține secvența (2,1,3) pentru care $C^R_{\max}=C^{RJ}_{\max}=2K-8k$, deci $C^R_{\max}/C^*_{\max}=C^{RJ}_{\max}/C^*_{\max}=2-8k/K$ și are o valoare oricât de apropiată de 2. Euristica J generează secvența (3,1,2) cu $C^J_{\max}=2K-5k$, deci $C^J_{\max}/C^*_{\max}=2-5k/K$ și are o valoare oricât de apropiată de 2. ■

Pentru euristica RJ există două cazuri pentru care abaterea maximă de la valoarea optimală este mai mică decât 0.5.

Lema 5.5. Dacă $a_{\sigma(i)} \le b_{\sigma(i)}$, pentru $i = u, u+1, ..., v$ sau dacă $a_{\sigma(i)} \ge b_{\sigma(i)}$, pentru $i = v, v+1, ..., u$, atunci $C^{RJ}_{\max} / C^*_{\max} < 3/2$.

Demonstrație. Pentru secvența $\{\sigma(u), \sigma(u+1), ..., \sigma(u)\}$ timpii de procesare ai lucrărilor pe mașinile A și B satisfac respectiv condițiile:

$$C^*_{\max} > r_{\sigma(u)} + \sum_{i=u}^{u} a_{\sigma(i)} \text{ și} \tag{5.9}$$

$$C^*_{\max} > r_{\sigma(u)} + \sum_{i=u}^{u} b_{\sigma(i)}. \tag{5.10}$$

Scăzând (5.10) din (5.3), rezultă că:

$$C^{RJ}_{\max} - C^*_{\max} < \sum_{i=u}^{v} a_{\sigma(i)} - \sum_{i=u}^{v-1} b_{\sigma(i)}. \tag{5.11}$$

Dacă $a_{\sigma(i)} \le b_{\sigma(i)}$, pentru $i = u, u+1, ..., v$, atunci din (5.11) rezultă că:

https://biblioteca-digitala.ro / https://unibuc.ro

$$C^{RJ}_{\max} - C^{*}_{\max} < a_{\sigma(v)} \leq 1/2\ (a_{\sigma(v)} + b_{\sigma(v)}) \leq 1/2\ C^{*}_{\max},$$

deci $C^{RJ}_{\max}\ /\ C^{*}_{\max} < 3/2$. Scăzând (5.9) din (5.3) rezultă că:

$$C^{RJ}_{\max} - C^{*}_{\max} < \sum_{i=v}^{n} b_{\sigma(i)} - \sum_{i=v+1}^{n} a_{\sigma(i)}.$$

Dacă $a_{\sigma(i)} \geq b_{\sigma(i)}$, pentru $i = v, v + 1, ..., u$, atunci din (5.11) rezultă că:

$$C^{RJ}_{\max} - C^{*}_{\max} < b_{\sigma(v)} \leq 1/2\ (a_{\sigma(v)} + b_{\sigma(v)}) \leq 1/2\ C^{*}_{\max},$$

deci $C^{RJ}_{\max}\ /\ C^{*}_{\max} < 3/2$. ■

Există o euristică îmbunătățită pentru care $C^{RJ}_{\max}\ /\ C^{*}_{\max} < 5/3$.

Exemplul 5.19. Cu datele din exemplul 5.18., pentru succesiunea **(2, 1, 3),** calculele se efectuează după cum urmează:

$u = 1, v = 1$: $C^{H}_{\max} = 0 + \sum_{i=1}^{1} a_{\sigma(i)} + \sum_{i=1}^{3} b_{\sigma(i)} = 0 + K - 6k + K - 6k + 2k + k = 2K - 9k$,

$u = 1, v = 2$: $C^{H}_{\max} = 0 + \sum_{i=1}^{2} a_{\sigma(i)} + \sum_{i=2}^{3} b_{\sigma(i)} = 0 + K - 6k + 2k + K - 6k + 2k = 2K - 8k$,

$u = 1, v = 3$: $C^{H}_{\max} = 0 + \sum_{i=1}^{3} a_{\sigma(i)} + \sum_{i=3}^{3} b_{\sigma(i)} = 0 + K - 6k + 2k + k + 2k = K - k$,

maximul având loc pentru $u = 1, v = 2$. Programarea este arătată în figura 5. . ■

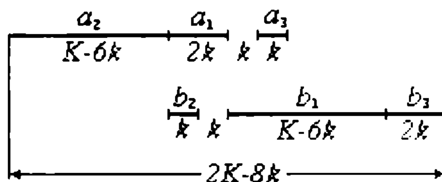


Figura 5.18.

5.2.3.1.4. *Algoritm euristic pentru procesare pe mai multe procesoare*

Punerea problemei. Fiind date n lucrări ce urmează a fi executate pe m procesoare, ocupate în aceeași ordine, timpul de execuție a lucrării i pe procesorul j fiind t_{ij}, $i = 1, 2, ..., n$; $j = 1, 2, ..., m$, se cere determinarea ordinii în care cele n lucrări trebuiesc executate pe cele m mașini astfel încât timpul total de lucru să fie minim.

Fie σ o secvență parțială conținând $k < n$ lucrări, o lucrare a și σa rezultatul concatenării lui a cu σ. Atunci, folosind ipotezele menționate în [BAK69], relația recursivă pentru timpul de execuție a secvenței parțiale σa de lungime $k + 1$ pe mașina j, notat $T(\sigma a, j)$, este:

$$T(\sigma a, j) = \max\ [T(\sigma, j);\ T(\sigma a, j - 1)] + t_{aj}, \tag{5.12}$$

unde:

$$T(\varnothing, j) = T(\sigma, 0) = 0, \text{ pentru toți } \sigma \text{ și } j.$$

https://biblioteca-digitala.ro / https://unibuc.ro

Problema planificării constă în minimizarea lui $T(\sigma\ a,\ j)$, unde a parcurge toate lucrările şi σ parcurge toate secvenţele posibile de n-1 lucrări diferite de lucrarea a. ■

Problema planificării lucrărilor este interpretată drept un caz complex de sortare a n obiecte astfel încât să se asigure minimizarea funcţiei $f(t_{11}, t_{12}, \ldots, t_{nm})$ a timpului de lucru. Dificultatea problemei echivalente de sortare constă în faptul că o asemenea funcţie nu poate fi definită fără cunoaşterea secvenţelor anterioare şi posterioare ale fiecărei lucrări. După cum se arată în lucrările [DUD64] şi [PAG61], pentru cazul a două procesoare, funcţia $f(i)$ asociată lucrării i are forma:

$$f(i) = A / \min(t_{i1}, t_{i2}), \text{ unde } A = 1 \text{ dacă } t_{i2} \le t_{i1} \text{ şi } A = -1 \text{ altfel.}$$

Pentru cazul a trei procesoare, studiat de Johnson [JOH54], atunci când:

$$t_{a2} \le \min_{1 \le b \le n} (t_{b1}, t_{b3}),$$

funcţia $f(i)$ are forma:

$$f(i) = A / \min_{1 \le j \le 2} (t_{ij} + t_{ij+1}), \text{ unde } A = 1 \text{ dacă } t_{i3} \le t_{i1} \text{ şi } A = -1 \text{ altfel.}$$

Problema planificării, în cazul a două sau trei procesoare, este echivalentă cu problema sortării a n numere astfel încât $f(a_1) \le f(a_2) \le \ldots \le f(a_n)$, unde a_i reprezintă lucrarea de pe locul i. Pentru cazul a mai mult de trei procesoare se propune o generalizare a funcţiei $f(i)$ definită astfel:

$$f(i) = A / \min_{1 \le j \le m-1} (t_{ij} + t_{ij+1}), \text{ unde } A = 1 \text{ dacă } t_{im} \le t_{i1} \text{ şi } A = -1 \text{ altfel.} \quad (5.13)$$

Algoritmul euristic propus este:

Algoritmul 5.13. (***Johnson generalizat***)

1. [Calcul funcţie] Pentru fiecare lucrare i se calculează $f(i)$ definit prin (5.13).
2. [Ordonare] Se ordonează lucrările în ordinea crescătoare a valorilor $f(i)$, în cazul egalităţii $f(i) = f(i + 1)$, acordând prioritate lucrării pentru care suma timpilor de lucru pe toate procesoarele este mai mică.
3. [Calcul timp] Se calculează timpul necesar efectuării lucrărilor utilizând (5.12). ■

Exemplul 5.20. Pentru ilustrarea euristicii propuse, se consideră următorul exemplu cu patru lucrări şi cinci procesoare timpii de lucru fiind daţi în tabelul 5.5.

i \ j	1	2	3	4	5
1	4	3	7	2	8
2	3	7	2	8	5
3	1	2	4	3	7
4	3	4	3	7	2

Tabelul 5.5.

https://biblioteca-digitala.ro / https://unibuc.ro

Pentru acest exemplu, valorile $f(i)$ sunt: $f(1) = -1/7$, $f(2) = -1/9$, $f(3) = -1/3$, $f(4) = 1/7$. Rezultă secvența 3124 a lucrărilor. Utilizând relația (5.12) rezultă că timpul de lucru este egal cu 34 de unități. Soluția optimală este definită de secvența 3412 și necesită un timp de lucru egal cu 33 de unități. ■

Pe baza testărilor efectuate pe mai multe exemple, rezultă că euristica propusă furnizează soluții aproape optimale sau optimale în majoritatea cazurilor.

Se poate demonstra că euristica propusă de Palmer [PAL65] se bazează pe analogia, menționată de Page [PAG61], dintre problemele de sortare și planificare. Dacă t_{ij} este timpul necesar lucrării i pe procesorul j, $i = 1, 2, ..., n$, $j = 1, 2, ..., m$, Palmer a propus ordonarea lucrărilor în ordinea descrescătoare a $s(i)$, unde:

$$s(i) = 1/2\ (-(m-1)\ t_{i1} - (m-3)\ t_{i2} + ... + (m-3)\ t_{i,m-1} + (m-1)\ t_{im}) = 1/2\ \sum_{j=1}^{m}(2j-m-1)\ t_{ij}.$$

Pentru a considera problema planificării drept o problemă de sortare, Page a propus o funcție $f(i)$ astfel încât lucrările să fie parcurse în ordinea crescătoare a valorilor acestei funcții. Funcția propusă este:

$$f(i) = -2\ s(i) = \sum_{j=1}^{m}(m-2j+1)\ t_{ij}.$$

Exemplul 5.21. Figura 5.19. ilustrează soluția furnizată de euristică pentru datele din exemplul 5.20.

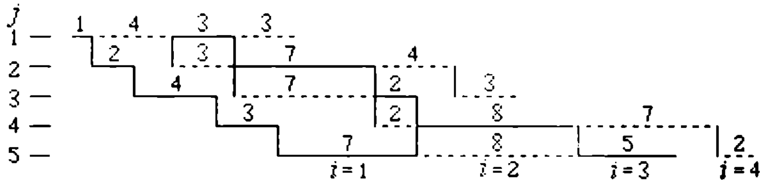


Fig. 5.19.

Segmentele ce corespund valorilor t_{ij} pentru $i = 1$ și $i = 3$ sunt desenate cu linii pline și celelalte, pentru $i = 2$ și $i = 4$, sunt desenate cu linii punctate. ■

5.2.3.1.5. *Euristici bazate pe indicele de înclinare*

Punerea problemei. Fiind date n lucrări și m procesoare, se cere identificarea ordinei în care urmează să fie parcurse cele n lucrări în vederea minimizării timpului de execuție. Timpul de ocupare a procesorului j de lucrarea i se notează cu t_{ij}. ■

Dacă ordonarea lucrărilor pe fiecare procesor este aceeași, soluția optimă se obține pentru una din cele $n!$ ordonări inițiale, a căror enumerare necesită un timp exponențial, fapt ce justifică utilizarea metodelor euristice de rezolvare a problemei.

Două margini inferioare, obținute pe calea inspectării diferitelor situații, sunt date în continuare, dar este puțin probabilă atingerea lor pentru valori mari ale lui n și m.

https://biblioteca-digitala.ro / https://unibuc.ro

a. Timpul total nu este mai mic decât: (timpul total pe procesorul j) + (timpul pe procesoarele 1 până la j - 1 a lucrării i_1) + (timpul pe procesoarele j + 1 până la n a lucrării i_2), pentru toți j și în ipoteza că i_1 și i_2 reprezintă prima și ultima lucrare, fiind diferite între ele și fiind alese în așa fel încât să minimizeze suma ultimilor doi termeni. Se presupune, de asemenea, că nu există intervale de timp neocupate în etapele lui i_1 ce preced începutul lucrului pe procesorul j, în timpul lucrului procesorului j și în etapele lui i_2 ce succed încheierii lucrului pe procesorul j.

b. Timpul total nu este mai mic decât: (timpul total pentru lucrarea i)+(suma timpilor minimali pe procesorul 1 și procesorul m consumați pentru celelalt lucrări), pentru toate valorile lui i. Această margine presupune că nu există întreruperi în procesarea lucrării i; lucrările cu timp de lucru mai mic pe procesorul 1 decât pe procesorul m sunt considerate înaintea lucrării i și celelalte după i, astfel încât timpul consumat înaintea și după procesarea lucrării i este redus la minimum.

Dacă unele dintre t_{ij} sunt nule, atunci lucrările cu $t_{i_1 j}$ și $t_{i_2 j}$ nule trebuiesc excluse în cazul marginii a.. Trebuie de asemenea avută în vedere posibilitatea schimbării ordinei lucrărilor ce afectează marginea b..

Definițiile celor două margini inferioare sunt ilustrate pe figurile 5.20.a) și b).

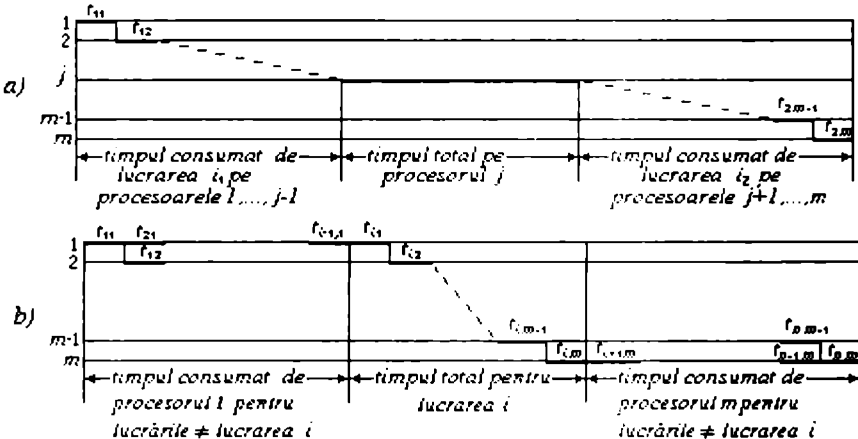


Fig. 5.20.

Euristica propusă utilizează noțiunea de *indice de înclinare* (*slope index*) în care principiul de bază este: "Se acordă prioritate activităților ce au cea mai mare tendință de trecere de la timpii de lucru mici la timpii de lucru mari în ordinea încărcării pe procesoare". Pentru fiecare activitate i se poate utiliza un ***indice de înclinare*** definit prin:

$$S_i = -(m-1)\,t_{i1} - (m-3)\,t_{i2} - \ldots + (m-3)\,t_{i,m-1} + (m-1)\,t_{im}.$$

https://biblioteca-digitala.ro / https://unibuc.ro

În această formulă ponderile asociate timpilor de lucru pe procesoare sunt determinate prin metoda celor mai mici pătrate plecând de la o aproximare liniară a timpilor în raport cu indicele procesorului. Activitățile sunt procesate în ordinea descrescătoare a indicelui S_i.

Această aproximare favorizează evident apropierea de marginile a. și b. de mai sus. O argumentare neriguroasă este ilustrată pe figura 5.21. Blocurile 1,2,3 reprezintă, respectiv, lucrări cu timp de procesare crescător, uniform și descrescător. Ele pot fi cuplate mai economic, din punct de vedere al timpului de lucru total, în ordinea 1,2,3.

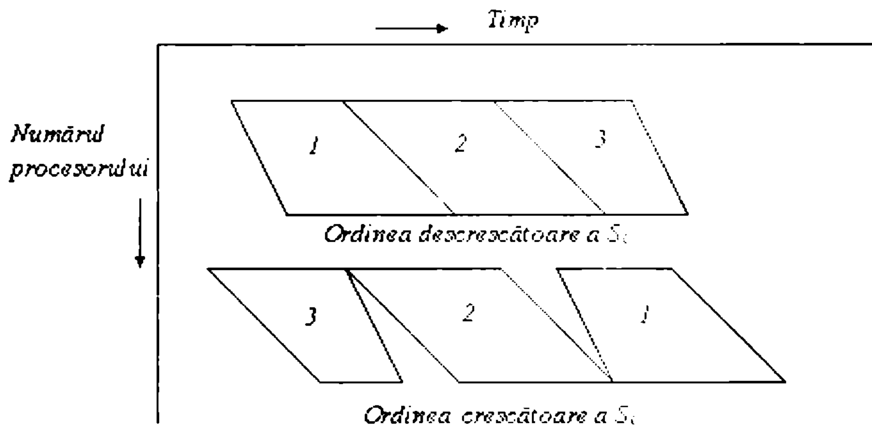


Fig. 5.21.

Rezultatele obținute în urma comparării soluțiilor obținute cu ajutorul euristicii propuse și cele optimale arată că diferența timpilor de lucru este cel mult de 10% din timpul optimal.

Rezultate mai bune se pot obține considerând indicele de înclinare ce utilizează ponderi asociate timpilor de lucru pe procesoare care sunt determinate prin metoda celor mai mici pătrate plecând de la o aproximare parabolică a timpilor în raport cu indicele procesorului. Se obține astfel pentru S_i expresia următoare:

$$S_i = \sum_{j=1}^{m} (6j^2 - 6j(m+1) + m + 2)t_{ij}.$$

Avantajul *împachetării* parabolice în ordinea descrescătoare a valorilor S_i este evident.

5.2.3.1.6. *Planificarea pe procesoare cu viteze diferite*

Punerea problemei. Se studiază problema procesării a n activități independente pe m procesoare de viteze diferite. Procesoarele sunt nepreferențiale, în sensul că nu există noțiunea de procesor rapid sau lent. ∎

https://biblioteca-digitala.ro / https://unibuc.ro

Un sistem de n activități și m procesoare este o matrice $n\times m$, notată μ, cu elemente din mulțimea $R^+\cup\{\infty\}$, pentru $n,m\geq 1$, astfel încât pentru fiecare i există un j astfel încât $\mu(i,j)\neq\infty$, $i=1,2,...,n$. Valoarea $\mu(i,j)$ este timpul consumat de activitatea i pe procesorul j, pentru $i=1,2,...,n$, $j=1,2,...,m$. O *funcție de asignare* $A:\{1,2,...,n\}\rightarrow\{1,2,...,m\}$ satisface condițiile $\mu(i,A(i))\neq\infty$, pentru $i=1,2,...,n$. Funcția $s:\{1,2,...,n\}\rightarrow R$, numită *funcție de start*, satisface condițiile:

a. Pentru orice j = 1, 2, ..., m cel mult o activitate este executată la un moment oarecare pe procesorul j.
b. Pentru orice j = 1, 2, ..., m, dacă $0 \leq x < \sum_{i:A(i)=j}\mu(i,j)$ măcar o activitate a fost executată la momentul x pe procesorul j.

Intuitiv, prima condiție exprimă faptul că toate activitățile asignate unui procesor sunt executate secvențial și a doua condiție previne perioadele neocupate pe procesori.

O activitate i este executată pe procesorul j la momentul x dacă $j=A(i)$ și $s(i)\leq x<s(i)+\mu(i,j)$. *Timpul de încheiere* al activității i este definit prin:

$$f(i) = s(i) + \mu(i, A(i)).$$

Matricea μ asociază m durate posibile pentru fiecare activitate. *Cea mai bună durată* pentru activitatea i, notată $b(i)$ este $b(i) = \min_j\{\mu(i,j)\}$. *Eficiența* procesorului j pentru activitatea i este definită astfel: $ef(i, j) = b(i) / \mu(i, j)$. *Timpul de încheiere a unei asignări A*, notat $f(A)$ este definit de:

$$f(A) = \max_{1\leq j\leq m}\sum_{i:A(i)=j}\mu(i,j).$$

Algoritmul descris în continuare generează o asignare A și o funcție de start s pentru un sistem μ de sarcini. În cadrul algoritmului se construiește câte o listă separată a sarcinilor pentru fiecare procesor, în vederea reflectării eficienței lui pentru diferite activități. Dacă un procesor nu este destul de eficient pentru următoarea activitate din listă, atunci el este desactivat.

Algoritmul 5.14. (***Asignare***)

1. [Calcul b] Pentru i = 1, 2, ..., n, se calculează $b(i) = \min_{1\leq j\leq m}\mu(i,j)$.
2. [Calcul ef] Pentru i = 1, 2, ..., n, j = 1, 2, ..., m, se calculează $ef(i,j) = b(i) / \mu(i,j)$.
3. [Inițializări] Pentru j=1,2,...,m se construiește o listă de sarcini i=1,2,...,n sortate în ordine nedescrescătoare a valorilor $ef(i,j)$. Se face sum_j=0, pentru j=1,2,...,m (sum_j este timpul de încheiere pe procesorul j a activităților atribuite până la un anumit moment). Se declară toate procesoarele ca fiind "active" și toate activitățile ca "neasignate".
4. [Terminare] Dacă toate activitățile sunt asignate, atunci STOP.
5. [Alegere procesor] Se alege un procesor j astfel încât sum_j este minimă pentru procesoarele active.

https://biblioteca-digitala.ro / https://unibuc.ro

6. [Alegere activitate] Se alege următoarea activitate neasignată din lista j a activităților. Dacă nu mai exită o astfel de activitate sau dacă $ef(i, j) < 1 / \sqrt{m}$, atunci se marchează j ca inactiv, altfel se definește $A(i) = j$, se notează i ca fiind asignată, se definește $s(i) = sum_j$ și se face $sum_j = sum_j + \mu(i, j)$.
7. [Ciclare] Se trece la pasul 4. ■

Se poate demonstra că eficiența algoritmului 5.14. este dată de relația:

$$f(A) / f(B) \leq 2.5 \sqrt{m} + 1 + 1 / \left(2\sqrt{m}\right),$$

unde $f(B)$ este timpul de încheiere a unui algoritm optimal.

În continuare vom prezenta alte variante îmbunătățite ale algoritmului 5.14.

Algoritmul 5.15. (Asignare îmbunătățită)

În pasul 3 al Algoritmului 5.14 se face precizarea că, dacă $ef(i, j) = ef(k, j)$, atunci i este procesată înainte de k dacă $\mu(i, j) \geq \mu(k, j)$; restul rămâne neschimbat. ■

Se poate demonstra că eficiența algoritmului 5.15. este dată de relația:

$$f(A) / f(B) \leq \left(1 + \sqrt{2}\right)\sqrt{m} + 2 + 1 / \left(\sqrt{8}\sqrt{m}\right).$$

Algoritmul 5.16. ***(Asignare performantă)***

1. [Soluție inițială] Se determină o asignare A_1 și funcția de start s utilizând Algoritmul 5.14.
2. [Reformulare] Fie u activitatea cu ultim timp de start în s și $U = \{i: f(i) > s(u)\}$.
3. [Optimizare] Se determină o asignare optimală pentru mulțimea U, considerată ca sistem de activități, încercând toate asignările posibile. Se asignează fiecare activitate din U procesorului ce îi este atribuit în această asignare optimală, activitățile din U fiind asignate a fi procesate după activitățile ce nu aparțin lui U care au fost deja asignate. ■

Se poate demonstra că eficiența algoritmului 5.15. este dată de relația:

$$f(A) / f(B) \leq 1.5 \sqrt{m} + 2 + 1 / \left(2\sqrt{m}\right).$$

Algoritmul 5.17. (Asignare cu subintervale)

Fie parametrul $\delta = 1/m^r$ pentru un r dat. În pasul 3 din Algoritmul 5.14. se înlocuiește sortarea activităților după eficiență cu următorii pași: Se împarte intervalul [0,1] în k subintervale $[0, 1/\sqrt{m})$, $[1/\sqrt{m}, (1+\delta)/\sqrt{m})$, $[(1+\delta)/\sqrt{m}, (1+2\delta)/\sqrt{m})$, ..., $[(1+k\delta)/\sqrt{m}, 1)$, unde $k \approx \left(\sqrt{m} / \delta\right) = m^{r+1/2}$. Se consideră fiecare dintre eficiențe $ef(i,j)$ drept un element al unui subinterval. Pentru fiecare procesor j se sortează activitățile în ordinea necrescătoare a subintervalelor la care aparțin. ■

Se poate demonstra că eficiența algoritmului 5.17. este dată de relația:

$$f(A) / f(B) \leq \left(\frac{5 - 4\delta}{2(1 - \delta)}\right)\sqrt{m} + \frac{1}{1 - \delta} + \frac{1}{2\sqrt{m}(1 - \delta)}.$$

https://biblioteca-digitala.ro / https://unibuc.ro

O altă euristică este dată de algoritmul următor:

Algoritmul 5.18. (***Asignare eficientă***)

1. [Inițializări] Se face $sum_j = 0$, pentru $j = 1, 2, ..., m$ și $S = \{1, 2, ..., n\}$.
2. [Terminare] Dacă $S = \varnothing$, atunci STOP.
3. [Prelucrare] Se determină un indice $i \in S$ astfel încât $\min_j\{sum_j + \mu(i, j)\} \leq$ $\leq \min_j\{sum_j + \mu(i', j)\}$, pentru toți $i' \in S$. Fie j astfel încât $sum_j + \mu(i, j)$ este minimă. Se definește $A(i) = j$. Se face $sum_j = sum_j + \mu(i, j)$ și $S = S - \{i\}$.
4. [Ciclare] Se trece la pasul 2. ■

Ideea de bază a algoritmului 5.18. este următoarea. După ce k activități au fost procesate, se încearcă procesarea încă a unei activități și minimizarea timpului de încheiere, pe calea alegerii activității și a procesorului căruia îi este asignată. Algoritmul se încheie după reluarea de n ori a pasului 3.

Problema eficienței algoritmului 5.18. este o problemă deschisă.

Menționăm un algoritm foarte simplu care, de fapt, generalizează algoritmii precedenți la situația în care există o relație de precedență. În acest caz un procesor poate fi neocupat cât timp condiția de precedență nu este satisfăcută. Timpul de încheiere al unei procesări depinde în mare măsură de ordinea procesării activităților.

Acest algoritm naiv constă în asignarea fiecărei activități unui procesor ce are eficiența 1 pentru activitatea considerată. Deoarece în acest caz general există o relație de precedență, unele procesoare pot fi temporar neocupate, dar nu există momente la care toate procesoarele să fie neocupate.

Se poate demonstra că eficiența algoritmului precedent este dată de relația:

$$f(B) / f(A) \leq m.$$

5.2.3.2. Metode de planificare pe bază de listă

Deoarece există situații în care timpii de execuție ai activităților nu sunt cunoscuți dinainte, este motivată considerarea unui mod de planificare care folosește lista activităților și constrângerile la care sunt supuse acestea. O planificare pe bază de liste, notată prescurtat *PL*, se poate descrie în felul următor: De fiecare dată când un procesor devine liber, se planifică la execuție pe acest procesor prima activitate din listă care nu a fost planificată și pentru care toți predecesorii au fost executați, dacă există o astfel de activitate. Lista de activități se alcătuiește, de cele mai multe ori, în ordinea dată de o lege de prioritate impusă de problema dată.

Exemplul 5.22. Să considerăm 9 activități cu timpii de execuție $t_1=3$, $t_2=t_3=t_4=2$, $t_5=t_6=t_7=t_8=4$, $t_9=9$, activitatea 1 precede pe 9 și activitatea 4 precede pe 5,6,7 și 8. Considerând lista $L=(1,2,3,4,5,6,7,8,9)$ se obține, pentru un sistem cu 3 procesoare, planificarea din figura 5.22. de durată totală 12. ■

https://biblioteca-digitala.ro / https://unibuc.ro

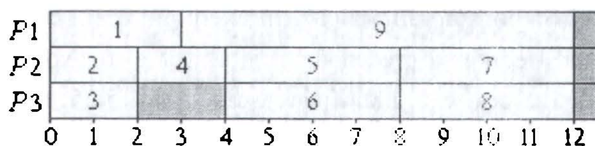


Fig. 5.22.

Planificarea *LPT* (vezi paragraful 1.1.2.3.1) este un caz particular de programare pe bază de listă, în care lista de priorități este dată de ordinea necrescătoare a timpilor de execuție.

Euristicile bazate pe programarea cu liste pot să producă unele anomalii pe care le vom exemplifica în continuare. Durata totală a unei planificări depinde în cea mai mare parte de numărul de procesori disponibili, de timpul de execuție al fiecărei activități, de constrângerile de precedență și de prioritățile atribuite activităților.

Dacă numărul procesoarelor crește, este de așteptat obținerea unei soluții mai bune. Exemplul următor arată că această regulă nu se poate aplica în toate cazurile.

Exemplul 5.23. Să considerăm datele din exemplul 5.22. cu singura deosebire că în loc de 3 procesori se pot folosi 4 procesori. Aplicarea euristicii pe bază de listă produce, în acest caz, planificarea din figura 5.23. de durată totală 15, deci mai mare decât cea din figura 5.22. pentru numai 3 procesoare. ■

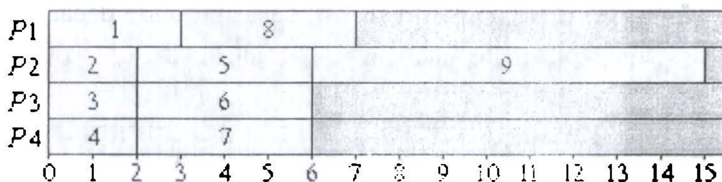


Fig. 5.23.

Dacă timpii de execuție se micșorează, se așteaptă obținerea unei soluții mai bune. Exemplul următor arată că această regulă nu se poate aplica în toate cazurile.

Exemplul 5.24. Să considerăm datele din exemplul 5.22. cu singura deosebire că fiecare din timpii de execuție se micșorează cu o unitate. Aplicarea euristicii pe bază de listă produce, în acest caz, planificarea din figura 5.24. de durată totală 13, deci mai mare decât cea din figura 5.22. cu timpi mai mari de execuție. ■

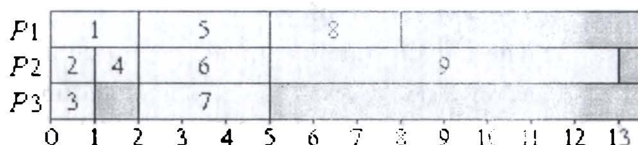


Fig. 5.24.

Dacă sunt eliminate unele dintre constrângerile de precedență, este de așteptat obținerea unei soluții mai bune, mulțimea soluțiilor posibile fiind mai numeroasă. Exemplul următor arată că această regulă nu se poate aplica în toate cazurile.

https://biblioteca-digitala.ro / https://unibuc.ro

Exemplul 5.25. Să considerăm datele din exemplul 5.22. cu singura deosebire că activitatea 4 nu mai trebuie să preceadă activitățile 5 și 6. Aplicarea euristicii pe bază de listă produce, în acest caz, planificarea din figura 5.25. de durată totală 16, deci mai mare decât cea din figura 5.22. cu mai multe constrângeri. ■

Fig. 5.25.

Dacă se fac unele modificări în listă care se par că aduc avantaje planificărilor, este de așteptat obținerea unei soluții mai bune. Exemplul următor arată că această regulă nu se poate aplica în toate cazurile.

Exemplul 5.26. Să considerăm datele din exemplul 5.22. cu singura deosebire că în loc de lista L se folosește lista L'=(1,2,4,5,6,3,9,7,8), făcând ca activitatea 3 care nu mai are succesori să fie de o prioritate mai mică decât activitatea 4 care are succesori și atribuind activității 9 care are o durată mare de execuție o prioritate mai mare, sugerată de *LPT*. Aplicarea euristicii pe bază de listă produce, în acest caz, planificarea din figura 5.26. de durată totală 14, deci mai mare decât cea din figura 5.22. care pare să se facă folosind o listă mai puțin eficientă. ■

Fig. 5.26.

5.2.3.2.1. *Margini pentru anomaliile din multiprocesare*

Fie date m procesoare P_j, j = 1, ..., m. și o mulțime de activități A = {1, ..., n} ce urmează a fi atribuite procesoarelor P_j, timpul de procesare al sarcinii i fiind t_i.

Algoritmul 5.19. (***Programare euristică***)

1. [Problema redusă] Se determină o soluție optimală pentru un număr $k \geq 0$ dintre sarcinile cu cei mai mari timpi de procesare, soluția obținută fiind o listă L.
2. [Completare] Se adaugă la lista L, într-o ordine arbitrară, cele n - k sarcini rămase, noua listă obținută fiind notată $L(k)$. ■

Sistemul multiprocesor are următorul mod de operare. Dacă un procesor P_i a terminat de procesat o sarcină, el caută o nouă sarcină în lista L. Dacă o astfel de sarcină nu există, atunci P_i rămâne neocupat până ce un alt procesor $P_{i'}$ devine liber.

https://biblioteca-digitala.ro / https://unibuc.ro

Teorema 5.17. Fie $\omega(k)$ timpul de încheiere al procesării listei $L(k)$ dată de algoritmul 5.20. și ω_0 timpul optimal necesar procesării mulțimii A; atunci are loc următoarea inegalitate:

$$\frac{\omega(k)}{\omega_0} \le 1 + \frac{1 - 1/m}{1 + [k/m]},$$

unde cu $[\cdot]$ este notată partea întreagă superioară a unui număr real. Această margine este cea mai bună pentru $k \equiv 0 \pmod n$.

Demonstrație. Fie ω_k soluția optimală furnizată de primul pas al algoritmului. Dacă $\omega(k)=\omega_k$, atunci $\omega(k)=\omega_0$ și teorema este verificată. Să presupunem că $\omega(k)>\omega_k$ și $r>k$. Fie $t^*=\max_{k+1\le i\le n}\{t_i\}$. În virtutea definiției lui t^* și a modului de operare a sistemului multiprocesor, rezultă că nici un procesor nu poate fi neocupat înaintea timpului $\omega(k)$ - t^*. Rezultă că:

$$\sum_{i=1}^{n} t_i \ge m\left(\omega(k) - t^*\right) + t^*$$

și prin urmare:

$$\omega_0 \ge \frac{1}{m}\sum_{i=1}^{n} t_i \ge \omega(k) - \left(\frac{m-1}{m}\right) t^*. \tag{5.14}$$

Există cel puțin k + 1 sarcini A_i ce au o durată de cel puțin t^*. Prin urmare, un anumit procesor execută măcar 1 + $[k / m]$ dintre aceste sarcini *lungi*. Rezultă că:

$$\omega_0 \ge \left(1 + \left[\frac{k}{m}\right]\right) t^*. \tag{5.15}$$

Din (5.14) și (5.15) se obține:

$$\omega(k) \le \omega_0 + \left(\frac{m-1}{m}\right) t^* \le \omega_0\left(1 + \frac{m-1}{m}\cdot\frac{1}{1+[k/m]}\right),$$

adică marginea enunțată în teoremă.

Pentru a demonstra că această margine este cea mai bună posibilă, pentru $k \equiv 0 \pmod n$, fie următorul exemplu. Se definește t_i, pentru $1 \le i \le k + 1 + m\,(m - 1)$, în felul următor:

$$t_i = \begin{cases} m, & \text{pentru } 1 \le i \le k+1, \\ 1, & \text{pentru } k+2 \le i \le k+1+m(m-1). \end{cases}$$

Pentru aceste sarcini și lista $L(k) = (A_1, \ldots, A_k, A_{k+2}, \ldots, A_{k+1+m(m-1)}, A_{k+1})$, se obține $\omega(k) = k + 2m - 1$. Deoarece $\omega_0 = k + m$, se obține:

$$\frac{\omega(k)}{\omega_0} = \frac{k+2m-1}{k+m} = 1 + \frac{m-1}{k+m} = 1 + \frac{1-1/m}{1+k/m} = 1 + \frac{1-1/m}{1+[k/m]}.$$

Cu aceasta demonstrația teoremei este încheiată. ■

Cazuri particulare. Pentru $k = 0$:

$$\frac{\omega(0)}{\omega_0} \le 2 - \frac{1}{m}.$$

Pentru $k = 2m$:

https://biblioteca-digitala.ro / https://unibuc.ro

$$\frac{\omega(2m)}{\omega_0} \leq 1 + \frac{1 - 1/m}{1 + [2m/m]} = \frac{4}{3} - \frac{1}{3m}.$$

Un alt caz particular este acela în care cele mai lungi $k = m$ sarcini sunt alocate câte una fiecărui procesor, ceea ce furnizează o soluție optimală pentru ω_k. Dacă cele $n - m$ sarcini rămase sunt alese arbitrar, rezultă că:

$$\frac{\omega(m)}{\omega_0} \leq \frac{3}{2} - \frac{1}{2m}.$$

Exemplul 5.27. Pentru $n = 13$, $k = 6$, $m = 3$ şi timpii de procesare:

$$t_i = \begin{cases} 3, \text{ pentru } 1 \leq i \leq 7, \\ 1, \text{ pentru } 8 \leq i \leq 13, \end{cases} \quad \omega(k) / \omega_0 = 1 + \frac{1 - 1/m}{1 + [k/m]} = 1 + \frac{1 - 1/3}{1 + 2} = \frac{11}{9}.$$

Soluția furnizată de algoritm şi cea optimală sunt date în figura 5.27. ■

P1	1	4	8	11	7
P2	2	5	9	12	
P3	3	6	10	13	

0 1 2 3 4 5 6 7 8 9 10 11

P1	1	4	7		
P2	2	5	8	10	12
P3	3	6	9	11	13

0 1 2 3 4 5 6 7 8 9

Fig. 5.27.

O margine a efectelor date de anomaliile euristicilor bazate pe liste este dată în următoarea teoremă.

Teorema 5.18. Fie un sistem S de n activități care trebuie planificate pe m procesori, cu timpii de execuție $t_1, t_2, \ldots, t_n$, cu lista L şi relația de precedență ρ. Fie un alt sistem S' cu n activități care trebuie planificate pe m' procesori, cu timpii de execuție $p_1, p_2, \ldots, p_n$, cu lista L' şi relația de precedență ρ'. Să presupunem că $p_i \leq t_i$, pentru $i=1,2,\ldots,n$, $\rho' \subseteq \rho$, timpul de planificare pe bază de listă a lui S este t şi cel al lui S' este t'. Atunci are loc relația:

$$\frac{t'}{t} \leq 1 + \frac{m-1}{m'}.$$

O demonstrație a acestei teoreme se poate găsi în [BÂS89]. ■

Faptul că nu se poate determina o margine mai bună se poate vedea din exemplul următor.

Exemplul 5.28. Să considerăm un sistem cu $2m$-1 activități independente având timpii $t_i=1$, pentru $i=1,2,\ldots,m-1$, $t_i=m-1$, pentru $i=m,m+1,\ldots,2m-2$ şi $t_{2m-1}=m$ care trebuiesc planificate pe m procesoare. Aplicând algoritmul de planificare pe bază de listă pentru lista $L=(1,2,\ldots,m-1,2m-1,m,m+1,\ldots,2m-2)$ se obține planificarea din figura 5.28.a) de durată totală m, fiind planificarea optimală. Dacă se foloseşte drept listă $L'=(1,m,m+1,\ldots,2m-2,2,3,\ldots,m-1,2m-1)$, algoritmul de programare pe bază de listă dă planificarea din figura 5.28.b) de durată $2m$-1. Cum raportul celor două durate este $\frac{2m-1}{m} = 1 + \frac{m-1}{m}$, rezultă că marginea din teorema 5.18. este cea mai bună posibilă. ■

https://biblioteca-digitala.ro / https://unibuc.ro

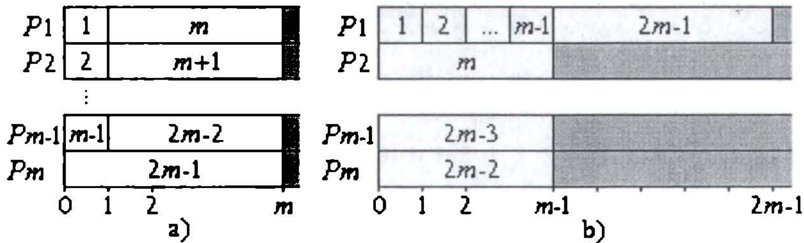


Fig. 5.28.

Se poate vedea că rezultatul din teorema 5.17 este un caz particular al rezultatului din teorema 5.18.

5.2.3.2.2. *Activităţi cu timpi egali şi precedenţe tip antiarborescenţe*

Fie un sistem A de n activităţi cu timpi de execuţie egali, pentru care relaţia de precedenţă corespunde unui graf de tip antiarborescenţă. Vom considera, fără a micşora din generalitate, că unitatea de timp este timpul de execuţie a unei activităţi

Vom spune că o activitate este *terminală* dacă nu are succesori. Numim *nivelul unui nod* lungimea maximă a unui drum de la nodul respectiv la un nod terminal.

Se poate construi o listă cu nodurile în ordinea necrescătoare a nivelurilor nodurilor aplicând următorul algoritm.

Algoritmul 5.20. (***Ordonare pe nivele***)

1. [Iniţializări] Se face $R = \{1, 2, ..., n\}$ şi $k = n + 1$.
2. [Terminare] Dacă R nu conţine noduri terminale, atunci STOP. (Dacă $R \neq \emptyset$, graful conţine cel puţin un ciclu şi deci nu se poate face planificarea).
3. [Alegere] Se alege o activitate terminală i din R.
4. [Adăugare la listă] Se face $k = k - 1$, se pune activitatea i pe locul k în lista L, se elimină i din R şi arcele care intră în i.
5. [Ciclare] Se merge la pasul 2. ■

Complexitatea algoritmului este $O(n)$.

Cu lista generată de algoritmul 5.20. se obţine o planificare a activităţilor aplicând algoritmul următor.

Algoritmul 5.21. (Planificare antiarborescenţă)

Când un procesor devine liber, se programează pe el prima activitate din listă care nu a fost programată şi pentru care toţi predecesorii direcţi au fost executaţi. ■

Demonstraţia faptului că algoritmul 5.21. construieşte o soluţie optimă poate fi găsită în [BÂS89].

https://biblioteca-digitala.ro / https://unibuc.ro

5.2.3.2.3. *Euristici bazate pe nivelurile nodurilor*

Considerăm un sistem cu n activităţi supuse la o relaţie de precedenţă oarecare, activitatea i având timpul de execuţie t_i.

Definim *nivelul* unui nod i, notat $niv(i)$, în modul următor: dacă i este nod terminal, adică fără succesori, atunci $niv(i)=t_i$, altfel $niv(i) = t_i + \max_{j \in S(i)} niv(j)$, unde $S(i)$ este mulţimea succesorilor direcţi ai lui i.

Algoritmul 5.22. (***Programare pe nivele***)

Când un procesor devine liber, se programează pe el o activitate de nivel maxim dintre activităţile care nu au fost programate şi pentru care toţi predecesorii direcţi au fost executaţi. ■

În cazul în care există mai multe activităţi cu acelaşi nivel se pot aplica diferite euristici de alegere cum ar fi:

- Se preferă o activitate de timp de execuţie cât mai mare.
- Se preferă o activitate de timp de execuţie cât mai mic.
- Se preferă o activitate cu cât mai mulţi succesori direcţi.
- Se alege la întâmplare oricare dintre ele.

Se pot da exemple prin care se arată că aceste euristici nu sunt comparabile.

Algoritmul 5.22. este de tip programare pe bază de listă dacă lista se alcătuieşte în ordinea descrescătoare a nivelului asociat fiecărui nod, eventual ţinând seama de euristica menţionată în cazul când există noduri cu acelaşi nivel.

5.2.3.2.4. *Programare optimală pentru doi procesori şi timpi egali*

Pentru cazul utilizării a doi procesori pentru programarea a n activităţi cu timpi de execuţie egali se poate construi o numerotare a activităţilor care permite obţinerea unei programări optimale. Numerotarea nodurilor se face aplicând algoritmul următor.

Algoritmul 5.23. (***numerotare***)

1. [Iniţializări] Se etichetează toate activităţile terminale cu λ (cuvântul vid) şi se face $k = 0$.
2. [Terminare] Dacă nu sunt noduri etichetate şi nenumerotate, atunci STOP. (Dacă au rămas noduri nenumerotate, atunci graful asociat conţine cel puţin un ciclu).
3. [Alegere] Se determină o activitate i etichetată şi nenumerotată care are eticheta cea mai mică în raport cu ordinea lexicografică dintre toate nodurile de acest tip. Dacă sunt mai multe activităţi cu aceeaşi etichetă minimă se ia oricare dintre ele.
4. [Numerotare] Se face $k = k + 1$ şi se numerotează activitatea i cu valoarea k.

https://biblioteca-digitala.ro / https://unibuc.ro

5. [Etichetare] Fiecare dintre predecesorii direcţi ai lui i care are toţi succesorii numerotaţi se etichetează cu şirul numerelor asociate succesorilor lui direcţi în ordine descrescătoare.
6. [Ciclare] Se merge la pasul 2. ■

Exemplul 5.29. Aplicând algoritmul 5.23. pentru sistemul cu 13 activităţi având graful de precedenţă din figura 5.29., mai întâi se etichetează activităţile 1 şi 3 cu (). Se numerotează activitatea 1 cu 1 şi se etichetează activităţile 2 şi 5 cu (1). Se numerotează activitatea 3 cu 2 şi se etichetează activitatea 4 cu (2). Se numerotează activitatea 5 cu 3 şi nu se fac etichetări. Se numerotează activitatea 2 cu 4 şi nu se fac etichetări. Se numerotează activitatea 4 cu 5 şi se etichetează activităţile 7 cu (5,4,3) şi 6 cu (5,4). Se numerotează activitatea 6 cu 6 şi nu se fac etichetări. Se numerotează activitatea 7 cu 7 şi se etichetează activităţile 8 cu (7) şi 9 cu (7,6). Analog se numerotează şi activităţile celelalte după ce au fost etichetate activităţile 10 cu (9,8), 11 cu (10), 12 cu (10) şi 13 cu (10,6). Numerotarea obţinută este dată în tabelul 5.6. ■

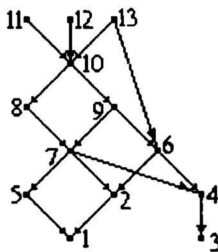


Fig. 5.29.

i	1	2	3	4	5	6	7	8	9	10	11	12	13
număr	1	4	2	5	3	6	7	8	9	10	11	12	13

Tabelul 5.6.

Lista se constituie luând activităţile în ordinea descrescătoare a numerelor asociate. Aplicând algoritmul de programare utilizând această listă se obţine soluţia optimă. Pentru demonstrarea acestei proprietăţi a se vedea [BÂS89].

Exemplul 5.30. Cu datele din exemplul 5.29., aplicând algoritmul de programare prin liste se obţine soluţia din figura 5.30. de timp total 8. ■

	0–1	1–2	2–3	3–4	4–5	5–6	6–7	7–8
$P1$	13	11	10	9	7	4	5	1
$P2$	12			8	6	2	3	

0 1 2 3 4 5 6 7 8

Fig. 5.30.

Prin relaxarea unor condiţii se obţin din algoritmul precedent diferite euristici. Astfel planificarea pe un număr de $m > 2$ procesoare poate să nu genereze soluţia optimală, după cum se vede din exemplul următor.

https://biblioteca-digitala.ro / https://unibuc.ro

Exemplul 5.31. Pentru sistemul cu 12 activităţi cu timpi de execuţie egali, cu graful de precedenţă din figura 5.31.a) (numerele de pe figură corespund celor obţinute prin algoritmul de numerotare), planificate pe 3 procesoare, aplicând algoritmul de planificare pe bază de liste se obţine planificarea din figura 5.31.b) de timp total 5. O planificare optimală este dată în figura 5.31.c) fiind de timp t   al 4. ■

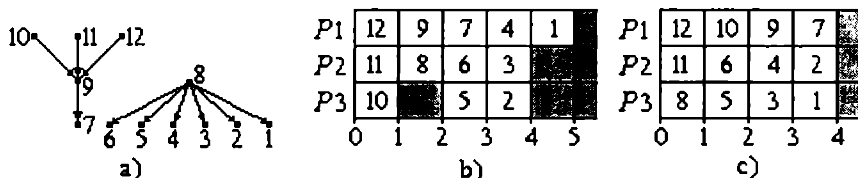


Fig. 5.31.

Un alt mod de relaxare prin care se obţine o euristică este cazul unor timpi diferiţi. Din exemplul următor se vede că nu se determină mereu soluţia optimală.

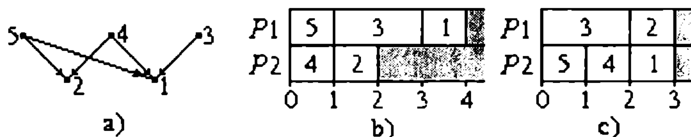


Fig. 5.32.

Exemplul 5.32. Pentru sistemul cu 5 activităţi cu timpi de execuţie $t_1=t_2=t_4=t_5=1$ şi $t_3=2$, cu graful de precedenţă din figura 5.32.a) (numerele de pe figură corespund celor obţinute prin algoritmul de numerotare), planificate pe 2 procesoare, aplicând algoritmul de planificare pe bază de liste se obţine planificarea din figura 5.32.b) de timp total 4. O planificare optimală este dată în figura 5.32.c) fiind de timp total 3. ■

5.3. Metode nepolinomiale de programarea activităţilor

În acest paragraf vom trata unele din metodele nepolinomiale de programarea activităţilor. Am considerat metodele acestea împărţite pe două tipuri: metode bazate pe programarea dinamică şi metode arborescente.

5.3.1. Algoritmi bazaţi pe programarea dinamică

O primă clasă de metode de programare nepolinomială se bazează pe metoda programării dinamice. Această metodă permite determinarea soluţiei optimale. Vom trata două cazuri şi anume tratarea prin programare dinamică a sistemelor cu activităţi independente şi cea a activităţilor dependente.

https://biblioteca-digitala.ro / https://unibuc.ro

5.3.1.1. Programarea activităţilor independente

Se consideră un sistem A cu n activităţi independente de durate $t_1,t_2,...,t_n$, disponibile la momentul 0, care trebuie să fie planificate pe un procesor. Se consideră funcţiile γ_i: $R^+ \to R^+$, care sunt funcţii cost crescătoare, $\gamma_i(t)$ este costul ca activitatea i să se termine la momentul t. Dintre planificările posibile, se caută o planificare care minimizează expresia $\sum_{i \in A} \gamma_i(c_i)$, unde c_i este momentul în care se termină activitatea i în planificarea considerată.

În termeni de programare dinamică, problema precedentă se enunţă astfel: starea la momentul k este $x(k) \subset A$, cu $x(0)=\varnothing$ şi $x(n)=A$; $\Delta(x,k)=A-x$, $r_k(x,u)=\gamma_u(t_x+t_u)$, unde $u \in \Delta(x,k)$ şi $t_x=\sum_{i \in x} t_i$ dacă $x \neq \varnothing$, altfel $t_x=0$. Drept criteriu se poate lua $J=\sum r_k(x,u)$. Un exemplu de funcţii γ_i este: $\gamma_i(t)=w_i$ dacă $t \leq d_i$, altfel $\gamma_i(t)=0$, unde w_i este profitul obţinut dacă activitatea i nu întârzie.

Soluţionarea se poate face prin *recurenţă înapoi* sau prin *recurenţă înainte*. În recurenţa înapoi se calculează funcţia f^* definită astfel:

$$f^*(A, n) = 0, f^*(x, k) = \max_{u \in A-x}\left\{\gamma_u\left(t_x + t_u\right) + f^*\left(x \cup \{u\}, k + 1\right)\right\}.$$

Analog, în recurenţa înainte se calculează funcţia g^* definită astfel:

$$g^*(\varnothing, 0) = 0, g^*(x, k) = \max_{u \in x}\left\{\gamma_u\left(t_x\right) + g^*\left(x - \{u\}, k - 1\right)\right\}.$$

Complexitatea acestor metode de soluţionare este $O(n\ 2^{n-1})$.

i	1	2	3	4
t_i	6	8	10	7
d_i	9	14	16	16
w_i	5	3	4	2

Tabelul 5.7.

Exemplul 5.33. Să considerăm un sistem cu 4 activităţi independente cu timpii de execuţie t_i, termenele de predare d_i şi profiturile w_i date în tabelul 5.7. În tabelul 5.8. sunt sintetizate calculele pentru recurenţa înapoi. Liniile, în afară de prima, corespund valorilor posibile pentru $x(k)$ menţionate în prima coloană şi coloanele reprezintă valorile calculate pentru aflarea lui f^*, cu menţionarea celor doi termeni din formula dată, la diferitele alegeri pentru u. Se marchează cu X combinaţiile care nu sunt valide. Penultima coloană este valoarea calculată pentru f^* şi ultima coloană indică valorile lui u pentru care se obţine valoarea lui f^*. Soluţiile se obţin astfel: din tabelul 5.8.d) rezultă o singură posibilitate pentru planificare şi anume activitatea 1; din tabelul 5.8.c), pentru mulţimea {1}, se află o singură alegere şi anume activitatea 3 este a doua planificată; din tabelul 5.8.b), pentru mulţimea {1,3}, sunt posibile două

https://biblioteca-digitala.ro / https://unibuc.ro

alegeri şi anume activităţile 2 şi 4 pentru a le planifica pe al treilea loc; dacă se alege 2, din tabelul 5.8.a) urmează 4, obţinând soluţia (1,3,2,4), şi dacă se alege 4, din tabelul 5.8.a) urmează 2, obţinând soluţia (1,3,4,2), ambele soluţii obţinute fiind optime, de valoare 9.

În mod analog, se pot face calculele pentru recurenţa înainte. Aceste calcule sunt sintetizate în tabelul 5.9. Soluţiile se obţin la fel ca mai sus, doar că parcurgerea programărilor se face de la sfârşit către început. ■

$x(3) \setminus u(3)$	1	2	3	4	$f^*(x,3)$	$v(x,3)$
{1, 2, 3}	X	X	X	0 + 0	0	4
{1, 2, 4}	X	X	0 + 0	X	0	3
{1, 3, 4}	X	0 + 0	X	X	0	2
{2, 3, 4}	0 + 0	X	X	X	0	1

a) $k = 3$.

$x(2) \setminus u(2)$	1	2	3	4	$f^*(x,2)$	$v(x,2)$
{1, 2}	X	X	0 + 0	0 + 0	0	3, 4
{1, 3}	X	0 + 0	X	0 + 0	0	2, 4
{1, 4}	X	0 + 0	0 + 0	X	0	2, 3
{2, 3}	0 + 0	X	X	0 + 0	0	2, 3
{2, 4}	0 + 0	X	0 + 0	X	0	1, 3
{3, 4}	0 + 0	0 + 0	X	X	0	1, 2

b) $k = 2$.

$x(1) \setminus u(1)$	1	2	3	4	$f^*(x,1)$	$v(x,1)$
{1}	X	3 + 0	4 + 0	2 + 0	4	3
{2}	0 + 0	X	0 + 0	2 + 0	2	4
{3}	0 + 0	0 + 0	X	0 + 0	0	1, 2, 4
{4}	0 + 0	0 + 0	0 + 0	X	0	1, 2, 3

c) $k = 1$.

$x(0) \setminus u(0)$	1	2	3	4	$f^*(x,0)$	$v(x,0)$
∅	5 + 4	3 + 2	4 + 0	2 + 0	9	1

d) $k = 0$.

Tabelul 5.8.

Se poate obţine un algoritm mai eficace bazat pe următoarea proprietate.

Teorema 5.19. Există o planificare optimală astfel încât activităţile terminate la termenele fixate să se execute înainte de cele întârziate şi în ordinea termenelor de predare. ■

https://biblioteca-digitala.ro / https://unibuc.ro

$x(1) \setminus u(1)$	1	2	3	4	$g^*(x, 1)$	$w(x, 1)$
{1}	5	X	X	X	5	1
{2}	X	3	X	X	3	2
{3}	X	X	4	X	4	3
{4}	X	X	X	2	2	4

a) $k = 1$.

$x(2) \setminus u(2)$	1	2	3	4	$g^*(x, 2)$	$w(x, 2)$
{1, 2}	0 + 3	3 + 5	X	X	8	2
{1, 3}	0 + 4	X	4 + 5	X	9	3
{1, 4}	0 + 2	X	X	2 + 5	7	4
{2, 3}	X	0 + 4	0 + 3	X	4	2
{2, 4}	X	0 + 2	X	2 + 3	5	4
{3, 4}	X	X	0 + 2	0 + 4	4	4

b) $k = 2$.

$x(3) \setminus u(3)$	1	2	3	4	$g^*(x, 3)$	$w(x, 3)$
{1, 2, 3}	0 + 4	0 + 9	0 + 8	X	9	2
{1, 2, 4}	0 + 5	0 + 7	X	0 + 8	8	4
{1, 3, 4}	0 + 4	X	0 + 7	0 + 9	9	4
{2, 3, 4}	X	0 + 4	0 + 5	0 + 4	5	3

c) $k = 3$.

$x(4) \setminus u(4)$	1	2	3	4	$f^*(x, 4)$	$w(x, 4)$
{1, 2, 3, 4}	0 + 5	0 + 9	0 + 8	0 + 9	9	2, 4

d) $k = 4$.

Tabelul 5.9.

O mulţime de activităţi cu terminare în termenele fixate ale unei soluţii optimale se numeşte *mulţime dominantă*. Pentru o mulţime x vom nota $\bar{t}(x) = \sum_{i \in x} t_i$ şi $\bar{w}(x) = \sum_{i \in x} w_i$. Are loc următoarea proprietate.

Teorema 5.20. Dacă $y_1, y_2 \subset \{1, 2, ..., i\}$, $\bar{t}(y_1) \geq \bar{t}(y_2)$ şi $\bar{w}(y_1) \leq \bar{w}(y_2)$, atunci orice soluţie x_1 care restricţionată la $\{1, 2, ..., i\}$ furnizează drept rezultat pe y_1 este dominată de o soluţie x_2 pentru care $y_2 \subset x_2$. ■

Algoritmul 5.24. (***Programare dinamică***)

O stare $x(i)$ corespunde la o submulţime din $\{1, 2, ..., i\}$ nedominată de o stare a etapei i deja calculată cu algoritmul de activităţi la termen. Se consideră $x(0) = \varnothing$ şi starea finală necunoscută. Mulţimea deciziilor $\Delta(x, i)$ se defineşte astfel: dacă are loc inegalitatea $\bar{t}(x) + t_{i+1} \leq d_i$, atunci se face $\Delta(x, i) = \{u_0, u_1\}$, altfel se face $\Delta(x, i) = u_1$ şi $g(x, u_0, i) = x \cup \{i + 1\}$, $g(x, u_1, i) = x$. Criteriul de evaluare este $\bar{w}(x(n))$. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Complexitatea algoritmului este O(n M), unde $M = \min(\bar{t}(A), \overline{w}(A))$.

Exemplul 5.34. Cu datele din exemplul 5.33., se obţin succesiv mulţimile $X_0=\{\varnothing\}$, $X_1=\{\varnothing,\{1\}\}$, $X_2=\{\varnothing,\{1\},\{1,2\}\}$, $X_3=\{\varnothing,\{1\},\{1,2\},\{1,3\}\}$, $X_4=\{\varnothing,\{1\},\{1,2\},\{1,3\},\{1,4\}\}$. Dintre variantele găsite, cea de profit maxim este $\{1,3\}$, care conduce la cele două soluţii (1,3,2,4) şi (1,3,4,2) de profit 9. ■

5.3.1.2. Programarea activităţilor dependente

Pentru rezolvarea problemei în cazul activităţilor dependente prin programare dinamică se poate adapta algoritmul recursiv înainte sub forma următoare. În acest algoritm se consideră *mulţime iniţială* mulţimea nodurilor fără predecesori şi *mulţime terminală*, notată $s(x)$, mulţimea activităţilor din x fără succesori.

Algoritmul 5.25. (***Programare dinamică***)

1. [Iniţializări] Se face $f^*(\varnothing, 0) = 0$ şi $k = 0$.
2. [Terminare] Dacă toate valorile lui f^* au fost calculate, atunci STOP.
3. [Calcul] Pentru un x şi un k pentru care se pot defini elementele de calcul se face:
$$f^*(x, k) = \min_{u\in s(x)} \{\gamma_u(t_x) + f^*(x - \{u\}, k - 1)\}.$$
4. [Ciclare] Se merge la pasul 2. ■

5.3.2. Metode arborescente

5.3.2.1. Minimizarea sumei întârzierilor

Punerea problemei. Fiind dată mulţimea $A=\{1,2,...,n\}$ de activităţi independente cu duratele de execuţie $t_1,t_2,...,t_n$ şi termenele limită de predare $d_1,d_2,...,d_n$, să se determine ordinea de planificare a acestor activităţi pe un procesor astfel încât să se minimizeze $\sum_{i=1}^{n} T_i$ cu $T_i=\max\{0,c_i-d_i\}$, unde c_i este momentul terminării activităţii i în planificarea dată. ■

Folosirea metodei arborescente pentru rezolvarea acestei probleme se bazează pe următoarele proprietăţi:

Teorema 5.21. Pentru orice submulţime $B=\{i_1,i_2,...,i_r\}$ de activităţi ale nodului S_σ care sunt programate în ordinea $\sigma=(i_r,i_{r-1},...,i_1)$, aceste activităţi sunt terminate cel mai târziu la momentele: $c_{i_1} = \sum_{i=1}^{n} t_i$, $c_{i_2} = c_{i_1} - t_{i_1}$, ..., $c_{i_r} = c_{i_{r-1}} - t_{i_{r-1}}$. ■

Teorema 5.22. În aceleaşi condiţii ca la teorema 5.21. valoarea:
$$F(\sigma) = \sum_{j=1}^{r} \max\left(c_{i_j} - d_{i_j}, 0\right)$$
este o evaluare prin lipsă a sumei întârzierilor. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Teorema 5.23. (de dominanţă) Fie S_σ un nod al arborescenţei şi B mulţimea activităţilor acestui nod. Dacă există $k \notin B$ astfel încât $d_k > \sum_{i \in A-B} t_i$, atunci se poate înlocui S_σ cu $S_{k\sigma}$. ■

Teorema 5.24. (de dominanţă) Fie S_{σ_1} cu activităţile B_1 şi S_{σ_2} cu activităţile B_2 două noduri ale arborescenţei astfel încât $B_1 \subset B_2$ şi $F(\sigma_2) \le F(\sigma_1)$; atunci se poate elimina nodul S_{σ_1}. ■

Algoritmul de determinare a unei soluţii optimale este următorul:

Algoritmul 5.26. (***Optimal***)

1. [Iniţializări] Se consideră rădăcina arborescenţei corespunzătoare succesiunii vide $\sigma = ()$, se face $\hat{f} = +\infty$ şi $F(\sigma) = 0$.
2. [Terminare] Dacă toate nodurile neeliminate au fost expandate, atunci STOP. Dacă $\hat{f} \neq +\infty$, atunci aceasta este valoarea optimală; altfel problema nu are soluţie.
3. [Alegere] Se alege un nod S_σ de valoare $F(\sigma)$ cât mai mică dintre nodurile neeliminate şi neexpandate; dacă sunt mai multe cu aceeaşi valoare, se preferă un nod cu cât mai multe activităţi.
4. [Expandare] Se expandează nodul S_σ introducând nodurile $S_{k\sigma}$, pentru orice $k \notin \sigma$.
5. [Extindere] Pentru fiecare nod nou introdus $S_{\sigma'}$ şi orice $k \notin \sigma'$ astfel încât $d_k > \sum_{i \in A-\sigma'} t_i$ se înlocuieşte $S_{\sigma'}$ cu $S_{k\sigma'}$.
6. [Evaluare] Se evaluează valorile corespunzătoare noilor noduri introduse folosind formulele din teoremele 5.21. şi 5.22.
7. [Ajustare] Dacă pentru un nod terminal nou, care conţine toate activităţile, valoarea calculată $F(\sigma)$ este mai mică decât $\hat{f}$, atunci se face $\hat{f} = F(\sigma)$.
8. [Eliminare] Se elimină nodurile care au valoare mai mare sau egală cu $\hat{f}$ sau cele pentru care există o supramulţime de valoare cel mult valoarea nodului respectiv (conform teoremei 5.24).
9. [Ciclare] Se trece la pasul 2. ■

Exemplul 5.35. Fie $A = \{1,2,3,4,5\}$ cu timpii de execuţie $t_1=4$, $t_2=3$, $t_3=7$, $t_4=2$, $t_5=2$ şi termenele de predare $d_1=5$, $d_2=6$, $d_3=8$, $d_4=8$ şi $d_5=17$. Se pleacă de la rădăcina S_0 de valoare 0 şi cu $\hat{f}=+\infty$. Se alege pentru expandare nodul S_0 obţinându-se noile noduri S_1,S_2,S_3,S_4 şi S_5. Pasul 5 permite înlocuirea primelor patru noduri respectiv prin S_{51},S_{52},S_{53} şi S_{54}. Prin evaluare se obţin următoarele: $F(51)=13$, $F(52)=12$, $F(53)=10$, $F(54)=10$ şi $F(5)=1$. Nu se poate aplica o ajustare şi nici eliminări de noduri. Se alege pentru expandare nodul S_5 şi se adaugă noile noduri S_{15},S_{25},S_{35} şi S_{45} care prin evaluări dau valorile: $F(15)=12$, $F(25)=11$, $F(35)=9$ şi $F(45)=9$. Aceste valori duc la eliminarea nodurilor S_{51},S_{52},S_{53} şi S_{54}. În continuare se pot expanda oricare din

https://biblioteca-digitala.ro / https://unibuc.ro

nodurile S_{35} şi S_{45}. Se alege S_{45} şi se introduc noile noduri S_{145},S_{245} şi S_{345}, cu valorile corespunzătoare $F(145)=18$, $F(245)=17$ şi $F(345)=15$. Se alege apoi S_{35} pentru expandare şi se introduc noile noduri S_{135},S_{235} şi S_{435}, cu valorile corespunzătoare $F(135)=13$, $F(235)=12$ şi $F(435)=10$. Se elimină nodul S_{345}. Se alege S_{435} pentru expandare şi se introduc noile noduri S_{1435} şi S_{2435} şi se pot înlocui cu S_{21435} şi S_{12435} cu valorile corespunzătoare $F(21435)=12$ şi $F(12435)=11$. Se elimină nodurile S_{145},S_{245}, S_{135},S_{235} şi S_{21435}. Se face $\hat{f}=11$ şi algoritmul se termină deoarece nu mai sunt noduri neeliminate care pot să fie expandate. Valoarea optimală este 11 şi corespunde programării (12435). Din cele 206 noduri posibil de expandat, au fost expandate numai 5.■

5.3.2.2. Minimizare durată totală pentru activităţi cu timp de latenţă

Punerea problemei. Se consideră mulţimea $A = \{1, 2, ..., n\}$ de activităţi independente cu duratele de execuţie t_1, t_2, ..., t_n, cu momentele de disponibilitate r_1, r_2, ..., r_n şi duratele de latenţă, adică timpul de execuţie a diferitelor operaţii ce se fac după terminarea execuţiei unei activităţi pe procesor, q_1, q_2, ..., q_n. Se cere să se determine o planificare a activităţilor pe un procesor astfel încât să se minimizeze $\max_{i\in A} (s_i + t_i + q_i)$, unde s_i este momentul de începere a execuţiei activităţii i. ■

O evaluare prin lipsă a duratei optimale este dată de expresia:

$$h(B) = \min_{i\in B} r_i + \sum_{i\in B} t_i + \min_{i\in B} q_i,$$

pentru orice $B \subset A$.

În algoritmul următor notăm cu U mulţimea activităţilor planificate şi cu $\overline{U}$ mulţimea activităţilor neplanificate la un moment dat. Se consideră două activităţi fictive, 0 activitatea de început, care precede toate activităţile din A şi $n + 1$ activitatea finală, care succede la toate activităţile din A. La terminarea algoritmului, s_{n+1} reprezintă timpul total de programare.

Algoritmul 5.27. (***Jackson***)

1. [Iniţializări] Se face $s_0 = 0$, $t = \min_{i\in A} r_i$, $U = \varnothing$ şi $\overline{U} = A$.
2. [Terminare] Dacă $\overline{U} = \varnothing$, atunci se face $s_{n+1} = \max_{i\in A}(s_i + t_i + q_i)$ şi STOP.
3. [Ajustare] Se face $t = \max\{t, \min_{i\in \overline{U}} r_i\}$.
4. [Alegere] Se alege din $\overline{U}$ o activitate i disponibilă la momentul t, adică pentru care $r_i \le t$, şi care are pentru q_i o valoare cât mai mare posibil.
5. [Modificări] Se face $U = U \cup \{i\}$, $\overline{U} = \overline{U} - \{i\}$, $s_i = t$ şi $t = t + t_i$.
6. [Ciclare] Se trece la pasul 2. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Exemplul 5.36. Fie un sistem cu n=7 activități ale căror date sunt mențonate în tabelul 5.10. Aplicând algoritmul se obține succesiv s_0=s_6=0, s_1=10, s_2=15, s_3=21, s_4=28, s_5=32, s_7=35, s_8=53. Un drum critic în graful asociat acestui sistem de activități este (0,1,2,3,4,8) de valoare 53. ■

i	1	2	3	4	5	6	7
r_i	10	13	11	20	30	0	30
t_i	5	6	7	4	3	6	2
q_i	7	26	24	21	8	17	0

Tabelul 5.10.

Fie mulțimea $A = \{1,2,...,n\}$, unde activitățile sunt numerotate în ordinea dată de algoritmul precedent. Vom spune că l este *activitate pivot inițială* dacă este de număr maxim dintre activitățile ce aparțin unui drum critic; analog spunem că h este *activitate pivot terminală* dacă este de număr minim dintre activitățile ce aparțin drumului critic $(0,...,l,n+1)$. Vom spune că c este *activitate critică* dacă este de număr maxim pe drumul $(h,...,l)$ pentru care $q_c<q_l$. Dacă nu există o astfel de activitate, se ia prin definiție $c=h$-1. Se definește *mulțimea critică* mulțimea $J=\{c+1,c+2,...,l\}$ de pe drumul critic.

Teorema 5.25. (fundamentală) Diferența între durata optimală și durata obținută prin aplicarea algoritmului lui Jackson este majorată de durata maximă de execuție a unei activități. Mai precis, dacă soluția dată de algoritmul lui Jackson nu este optimală, atunci $h(J) > f^* - t_c$; altfel, $f^* = h(J)$. ■

Se poate obține eventual o soluție mai bună dacă se pleacă de la soluția obținută aplicând algoritmul lui Jackson și se mută c după l. Apoi se alege dintre cele două soluții cea mai bună.

Dacă se consideră o estimare prin adaus $\hat{f}$ a valorii optimale, se pot determina soluții mai bune dacă se aplică următoarele proprietăți, în care am notat cu:

$$K = \{k \in A - J \mid t_k > \hat{f} - t_c\}.$$

Teorema 5.26. Dacă există $k \in K$ astfel încât $r_k + t_k + \sum_{i \in J} t_i + q_l \geq \hat{f}$, atunci, pentru o soluție de durată mai mică decât $\hat{f}$, activitatea k se va executa după toate activitățile din J și deci se poate pune $r_k = \max\,(r_k, \min_{i \in J} r_i + \sum_{i \in J} t_i)$. ■

Teorema 5.27. Dacă există $k \in K$ astfel încât $\min_{i \in J} r_i + t_k + \sum_{i \in J} t_i + q_k \geq \hat{f}$, atunci, pentru o soluție de durată mai mică decât $\hat{f}$, activitatea k se va executa înainte de toate activitățile din J și deci se poate pune $q_k = \max\,(q_k, \min_{i \in J} q_i + \sum_{i \in J} t_i)$. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Pentru aplicarea metodei arborescente la rezolvarea problemei, nodurile sunt descrise prin tripletele (r_i,t_i,q_i) şi se determină o evaluare prin lipsă $g(S)$ a problemei de rezolvat care poate fi determinată de exemplu prin rezolvarea problemei :. reemtive corespunzătoare. Se mai poate utiliza drept evaluare $h(J)$. În algoritm se alege un nod S terminal cu $g(S)$ minim şi se aplică pentru el algoritmul Jackson. Se calculează c şi J şi dacă valoarea programării este $f_0=\hat{f}$, atunci se elimină nodul S, altfel se construieşte soluţia cu c după J de valoare f_1 şi dacă $f_1<\hat{f}$, atunci se face $\hat{f}=f_1$. Se face apoi separarea lui S în două probleme: una cu c înainte de J în care se consideră $q_c=\max(q_c, \sum_{i\in J} t_i+q_l)$ şi una cu c după J în care $r_c=\max(r_c,\min_{i\in J} q_i + \sum_{i\in J} t_i)$. Algoritmul de tip metodă arborescentă construit conform procedeului descris anterior are următoarea structură.

Algoritmul 5.28. (***Metoda arborescentă***)

1. [Iniţializare] Se face $\hat{f} = +\infty$ şi se consideră arborescenţa cu un nod ce corespunde problemei de rezolvat.
2. [Terminare] Dacă toate nodurile au fost eliminate, atunci STOP.
3. [Alegere] Se determină un nod terminal S de cea mai mică evaluare prin lipsă $g(S)$. Dacă $g(S) \geq \hat{f}$, atunci se elimină toate nodurile arborescenţei şi STOP.
4. [Algoritm Jackson] Se aplică algoritmul lui Jackson pentru problema asociată lui S şi se calculează c, J şi valoarea f_0.
5. [Ajustare] Dacă $f_0<\hat{f}$, se scrie soluţia găsită, se face $\hat{f}=f_0$ şi se trece la pasul 2.
6. [Eliminare] Dacă $c = h - 1$, atunci se elimină nodul S şi se trece la pasul 2.
7. [Soluţie cu c după J] Se construieşte soluţia cu c după J de durată f_1. Dacă $f_1 < \hat{f}$, atunci se scrie soluţia şi se face $\hat{f} = f_1$.
8. [Succesor înainte] Se construieşte succesorul S_1 al lui S asociat problemei înainte recopiind datele lui S, apoi se modifică q_c. Se calculează $g(S_1)$ aplicând un algoritm cu preemţiune. Dacă $g(S_1) \geq \hat{f}$, atunci se elimină S_1.
9. [Succesor după] Se construieşte succesorul S_2 al lui S asociat problemei înapoi recopiind datele lui S, apoi se modifică r_c. Se calculează $g(S_2)$ aplicând un algoritm cu preemţiune. Dacă $g(S_2) \geq \hat{f}$, atunci se elimină S_2.
10. [Ciclare] Se trece la pasul 2. ■

Exemplul 5.37. Cu datele din exemplul 5.36., prin aplicarea algoritmului bazat pe metoda arborescentă se obţine mai întâi $h(J)$=53, c=1 şi J={2,3,4}. Deoarece pentru c înainte de J se obţine $h(J\cup\{1\})$=53, nu se mai introduce nodul S_1. Deci c se execută după J, se pune r_1=28 şi se introduce nodul S_2. Se aplică apoi algoritmul lui Jackson pentru problema obţinută şi se găseşte valoarea 50. Cu c după J se obţine valoarea 51, deci valoarea optimală este 50 şi corespunde la succesiunea (6,3,2,4,1,5,7). ■

https://biblioteca-digitala.ro / https://unibuc.ro

5.4. Alți algoritmi aproximativi pentru programarea activităților

5.4.1. Planificarea activităților pe procesoare distincte

Punerea problemei. Se studiază problema planificării a n activități pe m procesoare într-o ordine *comună*. ■

Problema este NP-completă chiar şi în cazul unui singur procesor, soluția optimă neputând fi identificată decât pe calea parcurgerii celor $n!$ permutări posibile ale activităților.

Fie t_i^k timpul necesar pentru activitatea i=1,2,...,n pe procesorul k =1,2,...,m şi s_{ij}^k intervalul de timp scurs între terminarea activității i pe procesorul k şi începutul activității j pe acelaşi procesor. În lucrarea [SZW87] se consideră cazul aditiv în care:

$$s_{ij}^k = x_{ik} + y_{jk},\ 1 \le i \ne j \le n,\ 1 \le k \le m,$$

unde x_{ik} şi y_{jk} sunt mărimi pozitive. În continuare se consideră cazul în care aceste mărimi pot lua şi valori negative, fapt ce arată că modelul aditiv este mai general decât se presupune în [SZW87].

Fie o secvență P arbitrară de n activități numerotate 1, 2, ..., n. Fie T^k timpul de încheiere a activității i pe procesorul k. Momentul încheierii activității i pe procesorul k fiind egal cu max(T_i^{k-1}, T_{i-1}^k + $s_{i-1,1}^k$) rezultă că:

$$T_i^k = \max(T_i^{k-1},\ T_{i-1}^k + s_{i-1,1}^k) + t_i^k,\ \text{unde } T_i^0 = T_0^k = 0, \qquad (5.16)$$

pentru i = 1, 2, ..., n şi k = 1, 2, ..., m. Pentru a construi modelul aditiv, se caută pentru fiecare k acei x_{ik} şi y_{jk}, $1 \le i \ne j \le n$, care minimizează valoarea expresiei:

$$z = \sum_i \sum_j (s_{ij}^k - x_{ik} - y_{jk})^2.$$

Anulând gradientul expresiei z şi rezolvând sistemul liniar de ecuații în x_{ik} şi y_{ik} rezultă soluțiile:

$$x_{ik} = x_{1k} + (n - 1) / n\ u_i^k - v_k,\ y_{ik} = u_i^k - x_{ik}, \qquad (5.17)$$

unde x_{1k} are o *valoare arbitrară pozitivă sau negativă* şi u_i^k şi v_k sunt expresiile:

$$u_i^k = 1 / (n - 2)\ (\sum_{j=1}^{n} s_{ij}^k + \sum_{j=1}^{n} s_{ji}^k - 1 / (n - 1) \sum_i \sum_j s_{ij}^k),\ i \ne j,$$

$$v_k = 1 / n\ (u_1^k + \sum_{j=1}^{n} s_{1j}^k + \sum_{j=1}^{n} s_{j1}^k - 1 / (n - 1) \sum_i \sum_j s_{ij}^k),\ i \ne j.$$

În locul problemei inițiale se caută mai întâi o soluție definită de (5.17) şi se consideră modelul aditiv ce se obține prin înlocuirea termenilor $s_{i-1,i}^k$ cu $x_{i-1,k} + y_{ik}$ în (5.16). Acest model este definit de:

https://biblioteca-digitala.ro / https://unibuc.ro

$$T_i^k = \max(T_i^{k-1}, T_{i-1}^k + x_{i-1,k} + y_{ik}) + t_i^k.$$

Se consideră apoi substituţiile:

$$a_{ik} = -x_{i,k-1} - y_{ik},\ a_{i1} = 0 \text{ şi } t_{ik} = t_i^k + x_{ik} + y_{ik} \tag{5.18}$$

şi

$$A_{iu} = \sum_{k=u}^{m-1} t_{ik} + \sum_{k=u+1}^{m} a_{ik},\ B_{iu} = \sum_{k=u+1}^{m} t_{ik} + \sum_{k=u+1}^{m} a_{ik}.$$

În continuare se prezintă un algoritm în care se utilizează algoritmul lui Johnson [JOH54], devenit acum clasic.

Algoritmul 5.29. (***tip Johnson***)

1. [Diferenţe] Se calculează diferenţele $A_{iu} - B_{iu}$, pentru toţi i.
2. [Separare] Fie $\alpha = \{i \mid A_{iu} - B_{iu} \le 0\}$ şi $\beta = \{i \mid A_{iu} - B_{iu} > 0\}$. Se aşează activităţile din α înaintea celor din β.
3. [Ordonare] Se ordonează activităţile din α în ordinea nedescrescătoare a valorilor A_{iu} şi activităţile din β în ordinea necrescătoare a valorilor B_{iu}.

Un algoritm echivalent ce utilizează metoda lui Johnson este următorul:

Algoritmul 5.30. (***Variantă***)

1. [Iniţializare] Se face $u = 1$.
2. [Doi procesori] Se utilizează metoda lui Johnson pentru rezolvarea problemei planificării pe două procesoare cu timpi de lucru A_{iu} şi B_{iu}, pe primul, respectiv al doilea procesor. Fie $P = p_1, p_2, ..., p_n$ secvenţa rezultată.
3. [Timpi] Se calculează timpul consumat pentru P cât şi pentru cele $n - 1$ secvenţe de forma $p_1, p_2, ..., p_{i-1}, p_{i+1}, ..., p_n, p_i$, $i = 1, 2, ..., n - 1$, utilizând formula (5.16).
4. [Ciclare] Se face $u = u + 1$. Dacă $u < n$, se trece la pasul 2; altfel se identifică cea mai bună secvenţă dintre cele $n (n - 1)$ secvenţe examinate şi STOP. ■

Problema alegerii valorilor x_{1k}, $1 \le k \le m$ este neesenţială, putându-se arăta că soluţia nu depinde de aceste valori, ele putând fi deci toate nule.

Exemplul 5.38. Se consideră o problemă de dimensiune 2×6, intervalele de timp t_i^1 şi t_i^2 fiind respectiv 25,88,12,12,48,1 şi 24,71,33,8,99,95. Valorile s_{ij}^1 şi s_{ij}^2 sunt date în tabelele 5.11 şi 5.12.

$(s_{ij}^1) =$

-	3	6	11	18	2	40
18	-	18	9	13	1	59
10	17	-	1	9	11	48
4	7	5	-	2	6	24
10	6	17	4	-	5	42
11	3	7	8	19	-	48
53	36	53	33	61	25	(261)

Tabelul 5.11.

https://biblioteca-digitala.ro / https://unibuc.ro

	-	31	42	43	41	45	202
	30	-	47	42	32	30	181
	32	46	-	36	38	45	197
$(s_{ij}^2) =$	30	49	32	-	42	44	197
	42	37	35	44	-	32	190
	48	32	48	42	41	-	211
	182	195	204	207	194	196	(1178)

Tabelul 5.12.

În aceste tebele sunt menționate și sumele valorilor s_{ij}^m pe linii și pe coloane și în paranteze sumele tuturor valorilor s_{ij}^m, $m = 1, 2$. Valorile x_{ik} și y_{ik}, calculate cu (5.17) și rotunjite la valori întregi, sunt date în tabelul 5.13.

	x_{i1}	y_{i1}	x_{i2}	y_{i2}
1	0	10	0	37
2	3	8	-4	39
3	2	10	0	41
4	-4	5	0	42
5	1	12	-2	39
6	1	4	2	41

Tabelul 5.13.

Cum $m = 2$, singura valoare a variabilei u este 1. Utilizând formulele (5.18) se calculează valorile a_{i2}, t_{i1} și t_{i2}, valorile a_{i1} sunt toate nule. Valorile A_{i1} și B_{i2} sunt:

-2, 57, -19, -25, 21, -36 și respectiv 24, 64, 31, 12, 96, 96.

Soluția furnizată de pasul 1 al algoritmului pentru exemplul considerat este dată de secvența P=6,4,3,1,5,2 și necesită un timp egal cu 515 (unități de timp). După testarea în pasul 2 a încă cinci secvențe pe calea mutării fiecărei activități din P la sfârșit a rezultat permutarea 6,4,3,1,2,5 care necesită un timp egal cu 500 și este acceptată ca soluție aproximativă. Soluția optimală este 6,2,5,3,4,1 și necesită un timp egal cu 496.

Generarea valorii 515 menționată mai sus se determină astfel:

$T_6^1 = \max(T_6^0, --) + t_6^1 = \max(0, --) + 1 = 1;$

$T_4^1 = \max(T_4^0, T_6^1 + s_{64}^1) + t_4^1 = \max(0, 1 + 8) + 12 = 21;$

$T_3^1 = \max(T_3^0, T_4^1 + s_{43}^1) + t_3^1 = \max(0, 21 + 5) + 12 = 38;$

$T_1^1 = \max(T_1^0, T_3^1 + s_{31}^1) + t_1^1 = \max(0, 38 + 10) + 25 = 73;$

$T_5^1 = \max(T_5^0, T_1^1 + s_{15}^1) + t_5^1 = \max(0, 73 + 18) + 48 = 139;$

https://biblioteca-digitala.ro / https://unibuc.ro

$T_2^1 = \max(T_2^0, T_5^1 + s_{52}^1) + t_2^1 = \max(0, 139 + 61) + 88 = 233;$
$T_6^2 = \max(T_6^1, --) + t_6^2 = \max(0, --) + 96 = 96;$
$T_4^2 = \max(T_4^1, T_6^2 + s_{64}^2) + t_4^2 = \max(21, 96 + 42) + 8 = 146;$
$T_3^2 = \max(T_3^1, T_4^2 + s_{43}^2) + t_3^2 = \max(38, 146 + 32) + 33 = 211;$
$T_1^2 = \max(T_1^1, T_3^2 + s_{31}^2) + t_1^2 = \max(73, 211 + 32) + 24 = 267;$
$T_5^2 = \max(T_5^1, T_1^2 + s_{15}^2) + t_5^2 = \max(139, 267 + 41) + 99 = 397;$
$T_2^2 = \max(T_2^1, T_5^2 + s_{52}^2) + t_2^2 = \max(233, 397 + 37) + 71 = 515.$

La efectuarea calculelor s-a ţinut seama de faptul că la aplicarea relaţiei (5.16) indicii i=1,2,3,4,5,6 din această relaţie corespund respectiv activităţilor 6,4,3,1,5,2. ■

5.4.2. Altă euristică pentru planificarea pe procesoare diferite

Punerea problemei. Fiind date n activităţi ce urmează a fi procesate pe m maşini, timpul de ocupare (procesare) al procesorului k de către activitatea i fiind t_i^k, $i = 1, 2, ..., n$; $k = 1, 2, ..., m$, se cere identificarea ordinii în care trebuiesc parcurse activităţile în vederea minimizării timpului total de ocupare a celor m maşini. ■

Fie o secvenţă σ după care urmează o activitate a. Atunci (vezi [GUP71]) relaţia recursivă ce defineşte timpul de completare al secvenţei σa pe maşina j este:

$$T(\sigma a, j) \max \{T(\sigma, j); T(\sigma a, j - 1)\} + t_a^j, \quad \text{unde } T(\varnothing, j) = T(\sigma, 0) = 0, \text{ pentru toţi } \sigma \text{ şi } j. \tag{5.19}$$

Problema de rezolvat constă în minimizarea lui $T(\sigma a, m)$, unde a aparţine mulţimii activităţilor şi σ aparţine mulţimii tuturor secvenţelor de (n - 1) activităţi şi care nu conţin activitatea a.

Ideea de bază a algoritmului propus constă în construirea, cu ajutorul datelor problemei de rezolvat, a mai multor probleme auxiliare de procesare pe numai două maşini, după care se alege ordinea de parcurgere a activităţilor ce necesită un timp minimal. Pentru cazul a două maşini, problema studiată este reductibilă (vezi [GUP71]) la cea a sortării, în ordine crescătoare, a n numere asociate fiecărei activităţi, conform definiţiei:

$$f(i) = A_i / \min(t_i^1, t_i^2), A_i = \begin{cases} 1, & \text{dacă } t_i^1 \geq t_i^2 \\ -1, & \text{altfel,} \end{cases} \tag{5.20}$$

unde t_i^1, respectiv t_i^2, este timpul de ocupare de activitatea i a maşinii 1, respectiv 2.

În cazul a $m > 2$ maşini, se construiesc $p = m - 1$ probleme auxiliare după cum urmează: suma timpilor de procesare pentru primele k, $k = 1, 2, ..., p$, maşini este interpretată drept timp de procesare pe maşina 1 a problemei auxiliare şi suma

https://biblioteca-digitala.ro / https://unibuc.ro

timpilor de procesare a ultimilor k maşini este interpretată ca fiind timpul de procesare pe maşina 2 a problemei auxiliare.

Definițiile timpilor de procesare $p_k(i,1)$ şi $p_k(i,2)$ ai problemei auxiliare k sunt:

$$\left.\begin{aligned} p_k(i,1) &= \sum_{j=1}^{k} t_i^j \qquad i=1,2,\ldots,n \\ p_k(i,2) &= \sum_{j=m-k+1}^{m} t_i^j \quad k=1,2,\ldots,m \end{aligned}\right\}. \tag{5.21}$$

Funcţia corespunzătoare $R(k, i)$, analoagă cu $f(i)$ din (5.20), este:

$$R(k, i) = A_{ki} / \min\left(p_k(i, 1), p_k(i, 2)\right),\ A_{ki} = \begin{cases} 1, & \text{dacă } p_k(i,1) \geq p_k(i,2) \\ -1, & \text{altfel,} \end{cases} \tag{5.22}$$

Cu aceste notații, algoritmul euristic propus este:

Algoritmul 5.31. (***Procesoare diferite***)

1. [Calcule R] Pentru fiecare k şi i, se calculează $R(k, i)$ prin relațiile (5.21) şi (5.22).
2. [Ordonare] Pentru fiecare k, se ordonează activitățile în ordinea crescătoare a valorilor $R(k,i)$. Dacă $R(k,i)=R(k,i+1)$ se acordă prioritate activității cu valoare mai mică a lui $R(\alpha,i)$, $\alpha=k+1,k+2,\ldots,m-1$ sau $\alpha=k-1,k-2,\ldots,1$. Dacă această procedură nu se încheie cu un rezultat, atunci se alege în mod arbitrar una dintre activități, adică i sau $i+1$.
3. [Timp] Pentru fiecare dintre cele k ordonări astfel obținute, se calculează timpul total de lucru, utilizând relația (5.19).
4. [Alegere] Se alege ordonarea cu timp total de lucru minimal şi care reprezintă soluția aproximativă a problemei. ■

Exemplul 5.39. Fie problema cu 4 activități şi 5 maşini cu timpii de execuție dați în tabelul 5.14.

i \ j	1	2	3	4	5
1	4	3	7	2	8
2	3	7	2	8	5
3	1	2	4	3	7
4	3	4	3	7	2

Tabelul 5.14.

Pasul 1. Cu relațiile (5.21) şi (5.22), se obțin rezultatele din tabelul 5.15.

Pasul 2. Urmărind tabelul 5.15., se constată că, pentru $k=2$, valorile $R(2,1)=R(2,4)$ sunt egale. Deoarece $k=2$, se consideră $\alpha=k+1=3$ şi se observă că $R(3,4)<R(3,1)$. În consecință, se acordă prioritate activității 4. Procedând în acest mod, se obțin patru programe de procesare.

https://biblioteca-digitala.ro / https://unibuc.ro

i	$R(k, i)$			
	$k = 1$	$k = 2$	$k = 3$	$k = 4$
1	-1/4	-1/7	-1/14	-1/16
2	-1/3	-1/10	-1/12	-1/20
3	-1	-1/3	-1/7	-1/10
4	1/2	-1/7	-1/10	1/16

Tabelul 5.15.

Pasul 3. Cu relaţia (5.19), se obţin timpii de lucru din tabelul 5.16.

Programare	Timp de lucru
3214	36
3412	33
3421	39
3124	34

Tabelul 5.16.

Pasul 4. Programarea 3412 are timpul de lucru minimal şi este acceptată drept soluţie a problemei. ■

Euristica propusă are la bază ideea de a reduce problema dată la câteva probleme care reflectă, într-o măsură suficient de bună, intercondiţionările dintre activităţile date.

5.4.3. Minimizarea duratei totale cu timpi de adaptare

Punerea problemei. Se studiază problema minimizării timpului de încheiere a execuţiei mai multor activităţi, pe calea ordonării lor convenabile, în cazul unei singure unităţi productive (procesor). ■

Fie $A=\{a_i\}$, $i=1,2,...,n$, o mulţime de n activităţi ce urmează a fi procesate pe o singură unitate productivă; t_{ij} = timpul necesar pentru a adapta unitatea productivă în vederea procesării activităţii j după activitatea i. Se presupune că întotdeauna $t_{ij}>0$ şi că, în general, $t_{ij} \neq t_{ji}$. De asemenea, se presupune că activitatea a_1 este procesată iniţial. Rezultă că, în acest caz, numărul de secvenţe distincte ce încep cu activitatea a_1 este egal cu $(n - 1)!$. Deoarece duratele de execuţie a activităţilor nu depind de ordinea lor, rezultă că problema studiată constă în minimizarea sumei intervalelor de timp necesare pentru trecerea de la o activitate la alta. Fie k permutarea $k_1=1,k_2,...,k_n$. Atunci problema constă în minimizarea sumei:

$$T_1 = \sum_{i=1}^{n-1} t_{k_i k_{i+1}},$$

valoarea ei minimală fiind notată în continuare cu T_{min}.

https://biblioteca-digitala.ro / https://unibuc.ro

Exemplul 5.40. Fie cazul a şase activități pentru care intervalele t_{ij} sunt date în tabelul 5.17. În acest tabel, locurile notate cu X corespund la succesiuni de perechi de activități ce nu se realizează şi valorile din tabel reprezintă pe t_{ij}. În acest exemplu, timpul T_{min} este egal cu 8.6 şi corespunde secvenței 1, 4, 3, 6, 2, 5. ■

i	j					
	1	2	3	4	5	6
1	X	3.0	2.8	2.2	2.5	1.7
2	X	X	1.7	2.3	1.8	0.9
3	X	3.4	X	3.5	2.7	1.9
4	X	1.8	1.2	X	1.2	2.2
5	X	2.5	2.4	3.6	X	2.5
6	X	1.5	2.6	3.4	2.4	X

Tabelul 5.17.

Problema studiată este o variantă a problemei comisvoiajorului şi, în consecință, identificarea unei soluții optimale este dificilă. Din acest motiv, utilizarea unor metode euristice este de preferat. O primă metodă euristică este *metoada celei mai apropiate activități*. Ideea acestei metode constă în selectarea în calitate de activitate următoare a acelei activități pentru care timpul t_{ij} este minimal.

Exemplul 5.41. În cazul datelor din exemplul 5.40., această metodă furnizează secvența 1, 6, 2, 3, 5, 4, pentru care $T_k = 11.2$. ■

Altă euristică se numeşte *metoda celei mai apropiate activități cu schimbarea originii*. În acest caz, metoda precedentă este aplicată succesiv, de cel puțin n - 1 ori, alegând pe rând altă activitate ce urmează după activitatea 1. Dacă la o etapă oarecare sunt mai multe activități candidat, atunci procesul se reia pentru toate aceste activități.

Exemplul 5.42. Pentru exemplul studiat, aplicând acestă metodă se obțin soluțiile din tabelul 5.18. ■

Secvența	T_k
1, 2, 6, 5, 3, 4	12.2
1, 3, 6, 2, 5, 4	11.6
1, 4, 3, 6, 2, 5	8.6
1, 4, 5, 3, 6, 2	9.2
1, 5, 3, 6, 2, 4	10.6
1, 6, 2, 3, 5, 4	11.2

Tabelul 5.18.

A treia euristică se numeşte *metoda celei mai apropiate activități cu micşorarea valorilor pe coloane*. Metoda celei mai apropiate activități se aplică după scăderea din valorile fiecărei coloane a valorii minime din această coloană.

https://biblioteca-digitala.ro / https://unibuc.ro

Exemplul 5.43. Pentru exemplul studiat, se obţine astfel tabelul 5.19.

	j					
i	1	2	3	4	5	6
1	X	1.5	1.6	0	1.3	0.8
2	X	X	0.5	0.1	0.6	0
3	X	1.9	X	1.3	1.5	1.0
4	X	0.3	0	X	0	1.3
5	X	1.0	1.2	1.4	X	1.6
6	X	0	0.4	1.2	1.2	X

Tabelul 5.19.

Aplicând metoda celei mai apropiate activităţi se obţine secvenţele 1,4,3,6,2,5 şi 1,4,5,2,6,3 la care corespund valorile T_k: 8.6 şi 9.4, prima fiind optimală. ■

Din testările experimentale efectuate rezultă că metoda propusă furnizează valori cu cel mult circa 1.27 ori mai mari decât cele optimale.

5.4.4. Planificări cu profit maxim

Punerea problemei. Fiind date n activităţi independente de durate $t_1, t_2, ..., t_n$, cu termene de predare la momentele $d_1, d_2, ..., d_n$ şi cu profiturile $w_1, w_2, ..., w_n$ dacă activităţile sunt terminate la timp, se cere o programare pe un procesor în aşa fel încât profitul total să fie maxim. ■

Să presupunem că activităţile sunt numerotate în aşa fel încât $d_1 \le d_2 \le ... \le d_n$ şi că profiturile sunt numere naturale. O euristică polinomială pentru determinarea unui profit cât mai mare este următorul algoritm.

Algoritmul 5.32. (***ProfitMax***)

1. [Iniţializări] Se face $P_0(0) = 0$ şi $P_0(i) = +\infty$, pentru $i = 1, 2, ..., \sum_{j=1}^{n} w_j$.
2. [Ciclare activităţi] Pentru $i = 1, 2, ..., n$ se execută paşii 3 şi 4.
3. [Valori iniţiale] Se face $P_i(k) = P_{i-1}(k)$, pentru $k = 1, 2, ..., \sum_{j=1}^{i-1} w_j$.
4. [Ciclare valoare profit] Pentru $k = w_i, w_i + 1, ..., \sum_{j=1}^{i} w_j$ se execută pasul 5.
5. [Ajustare] Dacă $P_{i-1}(k-w_i) + t_i < \min(d_i, P_i(k))$, atunci se face $P_i(k) = P_{i-1}(k-w_i) + t_i$.
6. [Calcul profit] Se determină $w^* = \max\{w \mid 0 \le w \le \sum_{i=1}^{n} w_i$ şi $P_n(w) \ne +\infty\}$. Se calculează mulţimea J^* optimală corespunzătoare, apoi STOP. ■

Complexitatea acestui algoritm este $O(n \sum_{i=1}^{n} w_i)$.

https://biblioteca-digitala.ro / https://unibuc.ro

Se poate folosi algoritmul precedent pentru a obţine o soluţie ε-aproximativă pentru cazul general, după cum se vede din algoritmul următor.

Algoritmul 5.33. (***Caz general***)

1. [Iniţializări] Se face $W = \max_{1 \le i \le n}\{w_i\}$ şi $K = \varepsilon W / n$.
2. [Trunchiere] Se face $w_i' = [w_i / K]$, $t_i' = t_i$, $d_i' = d_i$, pentru $i = 1, 2, ..., n$.
3. [Algoritm] Se aplică algoritmul ***ProfitMax*** pentru datele transformate, rezultatul fiind o aproximaţie a soluţiei problemei iniţiale, şi STOP. ■

Complexitatea acestui algoritm este $O(n^3 / \varepsilon)$.

O metodă mai eficientă se numeşte metoda de partiţie dinamică a intervalelor şi este dată în algoritmul următor.

Algoritmul 5.34. (partiţie dinamică a intervalelor)

1. [Iniţializări] Se face $m=[n/\varepsilon]$, $T_{ik}=+\infty$ şi $W_{ik}=0$, pentru $i=0,1,...,n$ şi $k=1,2,...,m$, $T_{01}=0$, $h_0=0$.
2. [Ciclu activităţi] Pentru $i=1,2,...,n$ se execută paşii 3,4 şi 5, apoi se trece la pasul 6.
3. [Calcul h_i] Se face $h_i = h_{i-1}$ şi pentru $k = 1, 2, ..., m$, dacă $T_{i-1,k} + \cdot \le d_i$ şi $W_{i-1,k} + w_i \ge h_i$, atunci se face $h_i = W_{i-1,k} + w_i + 1$.
4. [Calcule $T_{i,j}$ şi $W_{i,j}$] Pentru $k = 1, 2, ..., n$ se face $j = [(m\ W_{i-1,k} + 1) / h_i]$ şi dacă $T_{i-1,k} < T_{i,j}$, atunci se face $T_{i,j} = T_{i-1,k}$ şi $W_{i,j} = W_{i-1,k}$.
5. [Calcule $T_{i,l}$ şi $W_{i,l}$] Pentru $k = 1, 2, ..., m$ se face $l = [(m\ (W_{i-1,k} + w_i) + 1) / h_i]$ şi dacă $T_{i-1,k} + t_i < \min(T_{i,l}, d_i)$, atunci se face $P_{i,l} = P_{i-1,k} + t_i$ şi $W_{i,l} = W_{i-1,k} + w_i$.
6. [Soluţie aproximativă] Se face $k' = 1$ şi $\hat{w} = 0$. Pentru $k = 1, 2, ..., m$, dacă $T_{n,k} \ne +\infty$ şi $W_{n,k} > \hat{w}$, atunci se face $k' = k$ şi $\hat{w} = W_{n,k}$. Apoi STOP. ■

Complexitatea acestui algoritm este $O(n^2 / \varepsilon)$.

5.4.5. Minimizarea sumei întârzierilor

Punerea problemei. Iniţial sunt accesibile n activităţi de către un procesor ce poate parcurge o singură activitate la un moment dat. Nu se admit priorităţi în parcurgerea activităţilor. Activitatea i, unde $i=1,...,n$, necesită un *timp de procesare* t_i, are o *scadenţă* d_i şi o *pondere* w_i. Problema constă în identificarea unei permutări s_i din mulţimea permutărilor mulţimii {1,...,n} care minimizează întârzierea totală ponderată:

$$\sum_{i=1}^{n} w_{s(i)} \max\left(\sum_{j=1}^{i} t_{s(j)} - d_{s(i)}, 0\right). \blacksquare$$

Problema enunţată este NP-completă. În continuare se prezintă euristici de rezolvare bazate pe înlocuirea generării tuturor permutărilor prin generarea unor submulţimi de permutări, vecine într-un anumit sens, cu permutarea $(1,2,...,n)$.

https://biblioteca-digitala.ro / https://unibuc.ro

5.4.5.1. Căutarea locală şi vecinătăţi

Se consideră patru vecinătăţi ale unei permutări s definite după cum urmează:

- $N_A(s)$ = vecinătatea bazată pe interschimbarea adiacentă şi definită prin:
  $\{g \mid g \in S$, unde g se obţine din s prin permutarea unei perechi de activităţi adiacente$\}$.
- $N_K(s)$ = vecinătatea bazată pe K interschimbări în s şi definită astfel:
  $\{g \mid g \in S$, unde g se obţine din s prin interschimbarea oricărei perechi de activităţi de cel mult K ori succesiv, pentru $K = 1, 2, ..., n\text{-}1\}$.
- $N_{BS}(s)$ = vecinătatea bazată pe mutarea înapoi şi definită astfel:
  $\{g \mid g \in S$, unde $g = (s(1), ..., s(k\text{-}1), s(k\text{+}1), ..., s(l), s(k), s(l\text{+}1), ..., s(n))$, pentru $k = 1, 2, ..., n\text{-}1$ şi $l = k\text{+}1, ..., n\}$.
- $N_{GS}(s)$ = vecinătatea bazată pe schimbarea generală şi definită astfel:
  $N_{BS}(s) \cup \{g \mid g \in S$, unde $g = (s(1), ..., s(l\text{-}1), s(k), s(l), ...,s(k\text{-}1), s(k\text{+}1),..., s(n))$, pentru $k = 2, 3,..., n$ şi $l = 1, 2, ..., k\text{-}1\}$.

Exemplul 5.44. Cele patru vecinătăţi definite mai înainte sunt ilustrate în tabelul 5.20., pentru cazul $s = (1, 2, 3, 4)$ şi $n = 4$. ■

Permutări	Vecinătate			
	$N_A(s)$	$N_l(s)$	$N_{BS}(s)$	$N_{GS}(s)$
1243	*	*	*	*
1324	*	*	*	*
1342			*	*
1423				*
1432		*		
2134	*	*	*	*
2143				
2314			*	*
2341			*	*
3124				*
3214		*		
4123				*
4231		*		

Tabelul 5.20.

Din definiţii rezultă că $N_A(s) \subset N_1(s) \subset ... \subset N_{n\text{-}1}(s)$ şi $N_A(s) \subset N_{BS}(s) \subset N_{GS}(s)$. Numerele de elemente din mulţimile de vecinătăţi sunt respectiv $|N_A(s)| = n - 1$, $|N_1(s)| = |N_{BS}(s)| = n\,(n - 1) / 2$ şi $|N_{GS}(s)| = (n - 1)^2$.

https://biblioteca-digitala.ro / https://unibuc.ro

Procesul căutării locale constă în identificarea unei permutări g din vecinătatea unei permutări inițiale s care micșorează întârzierea totală. Dacă există un asemenea g, se pune $s = g$ și se repetă procedura precedentă. Altfel, s este un optim local. Atunci când se elaborează o euristică de căutare locală, trebuie determinate următoarele trei elemente: 1) metoda de identificare a permutării sursă (*seed permutation*), 2) vecinătatea și 3) metoda de identificare a următoarei permutări sursă.

Metodele de căutare bazate pe vecinătățile N_A, N_K, N_{BS} și N_{GS} sunt notate respectiv cu *ADJ*, *K-INT*, *BS* și *GS*. Pentru evaluarea performanțelor euristicilor de căutare locală sunt folosite următoarele notații:

t - vector cu $2n$ elemente $(t_1, t_2, ..., t_n, d_1, d_2, ..., d_n)$ care specifică problema dată, numită în continuare problema t.

F - mulțimea problemelor t posibile = $\{t \mid t = (t_1, t_2, ..., t_n, d_1, d_2, ..., d_n)$ și $d_i \geq 0$, pentru $i = 1, 2, ..., n\}$.

$LS(t; N)$ - mulțimea optimelor locale pentru problema t în vecinătatea lui N, unde N reprezintă una dintre cele patru vecinătăți.

$T(s; t)$ - întârzierea totală a permutării s la problema t.

$L(t; N)$ - valoarea maximă a întârzierii în problema t în cadrul optimelor locale din vecinătatea $N = \max T(s; t)$, cu $s \in LS(t; N)$.

$v(t)$ - întârzierea totală optimă pentru problema t.

$$e(t; N) = \begin{cases} L(t;N)/v(t) & \text{pentru } v(t)>0 \\ 1 & \text{pentru } v(t)=L(t;N)=0 \\ M & \text{pentru } v(t)=0 \text{ și } L(t;N)>0, \end{cases}$$

unde $M \geq 1$.

Eroarea relativă a unei euristici de căutare locală folosind vecinătatea N este definită astfel:

$$re(N) = \sup_{t \in F} e(t; N).$$

În continuare se consideră numărul de pași per iterație parcurși în cel mai defavorabil caz pentru fiecare euristică. Pentru *ADJ*, $|N_A(s)| = n-1$. Pentru compararea a două permutări adiacente se parcurge un număr finit de pași. Rezultă că numărul de pași per iterație este $O(n)$. Următorul exemplu arată că *BS* are o complexitate apropiată de *ADJ*.

Exemplul 5.45. Fie problema cu patru activități: s_2=(2,1,3,4), s_3=(2,3,1,4) și s_4=(2,3,4,1). Toate aceste permutări fac parte din $N_{BS}(s_1)$, pentru s_1=(1,2,3,4). Fie Δ_2, Δ_3 și Δ_4 creșterea întârzierii totale la trecerea din s_1 în s_2, din s_2 în s_3 și din s_3 în s_4. Dacă una dintre valorile Δ_2, $\Delta_2+\Delta_3$ și $\Delta_2+\Delta_3+\Delta_4$ este negativă, căutarea locală se efectuează pentru o altă vecinătate. Dacă euristica *BS* este realizată astfel încât

https://biblioteca-digitala.ro / https://unibuc.ro

creşterile să fie parcurse în ordinea Δ_2, $\Delta_2+\Delta_3$ şi $\Delta_2+\Delta_3+\Delta_4$ atunci numărul de paşi necesari pentru calculul fiecărei creşteri este constant. ■

Deoarece $|N_{BS}(s)| = n(n-1)/2$ rezultă că o iteraţie din *BS* necesită un număr de paşi $O(n^2)$. În mod similar, *GS* necesită un număr de $O(n^2)$ paşi per iteraţie. Deoarece aparent nu există relaţii recursive între valorile obiectiv (*objective values*) între membrii lui N_1, rezultă că 1-*INT* necesită un număr de $O(n^3)$ paşi per iteraţie.

5.4.5.2. Valorile maxime ale erorilor pentru *K-INT* şi *ADJ*

Teorema 5.28. Eroarea relativă pentru o interschimbare euristică n-2-*INT* poate fi oricât de mare, pentru $n \geq 4$, adică $re(N_{n-2}) = \infty$, pentru $n \geq 4$.

Demonstraţie. Se construieşte un exemplu cu $n \geq 4$ care are o eroare relativă arbitrar de mare. Fie ε, $0<\varepsilon<1/2$ şi următoarea problemă t: $d_1=2n-2$, $d_2=2-\varepsilon$, $d_i=1+(2i-3)(1-\varepsilon)$, pentru $i=3,4,\dots,n-1$, $d_n=1+(2n-4)(1-\varepsilon)$, $t_1=1-\varepsilon$, $t_2=1$, $t_i=2(1-\varepsilon)$, pentru $i=3,4,\dots,n-1$, şi $t_n=2-\varepsilon$. Au loc următoarele relaţii:

$$d_2 < d_3 < \dots < d_{n-1} < d_n < d_1, \tag{5.23}$$

$$\sum_{i=1}^{n} t_i - d_n = 1, \tag{5.24}$$

$$\sum_{i=2}^{n} t_i - d_n = \varepsilon, \tag{5.25}$$

$$\sum_{i=2}^{n} t_i - d_{n-1} = 1, \tag{5.26}$$

$$\sum_{i=2}^{k} t_i - d_{k-1} = 1 - \varepsilon, \text{ pentru } k = 3, 4, \dots, n-1. \tag{5.27}$$

Din (5.24) şi (5.25) rezultă respectiv:

$$T(1, 2, \dots, n) = \sum_{i=1}^{n} t_i - d_n = 1,\ T(2, 3, \dots, n, 1) = \sum_{i=2}^{n} t_i - d_n = \varepsilon.$$

În continuare se arată că $T(a_1,a_2,\dots,a_n) \geq 1$ pentru toate secvenţele $(a_1,a_2,\dots,a_n)$ cu excepţia secvenţelor $(1,2,\dots,n)$ şi $(2,3,\dots,n,1)$. Rezultă că $(1,2,\dots,n)$ este optim local şi $(2,3,\dots,n,1)$ este optim global. Deoarece $e(t) \geq 1/\varepsilon$, dacă ε tinde către zero, $e(t)$ tinde către infinit.

Pentru a arăta că $T(a_1, a_2, \dots, a_n) \geq 1$, pentru orice $(a_1, a_2, \dots, a_n)$ cu excepţia cazurilor $(1, 2, \dots, n)$ şi $(2, 3, \dots, n, 1)$, se consideră următoarele trei cazuri:

Cazul 1: $a_n \neq 1$.

$$T(a_1, a_2, \dots, a_n) \geq \max\left(\sum_{i=1}^{n} t_{a_i} - d_{a_n}, 0\right) \geq \sum_{i=1}^{n} t_i - d_n \text{ (din (5.23))} = 1 \text{ (din (5.24))}.$$

Cazul 2: $a_n = 1$ şi $a_{n-1} \neq n$.

$$T(a_1, a_2, \dots, a_{n-1}, 1) \geq \max\left(\sum_{i=1}^{n-1} t_{a_i} - d_{a_{n-1}}, 0\right) \geq \sum_{i=2}^{n} t_i - d_{n-1} \text{ (din (5.23))} = 1 \text{ (din (5.26))}.$$

https://biblioteca-digitala.ro / https://unibuc.ro

Cazul 3: $a_n = 1$, $a_{n-1} = n$, $a_{n-2} = n - 1$, ..., $a_k = k + 1$, $a_{k-1} \neq k$, pentru $k = n - 1, n - 2, ..., 3$.

$$T(a_1, a_2, ..., a_{k-1}, k+1, ..., n-1, n, 1) \geq \max(\sum_{i=1}^{k-1} t_{a_i} - d_{a_{k-1}}, 0) + \max(\sum_{i=1}^{n-1} t_{a_i} - d_{a_{n-1}}, 0) \geq$$

$$\geq \sum_{i=2}^{k} t_i - d_{k-1} + \sum_{i=2}^{n} t_i - d_{n-1} \text{ (din (5.24))} = 1 - \varepsilon + \varepsilon = 1 \text{ (din (5.27) şi (5.25))}. \blacksquare$$

Corolarul 1. Eroarea relativă pentru euristica 1-*INT* poate fi oricât de mare, pentru $n \geq 4$, adică $re(N_1) = \infty$, pentru $n \geq 4$. ■

Corolarul 2. Eroarea relativă pentru euristica *ADJ* poate fi oricât de mare, pentru $n \geq 4$, adică $re(N_A) = \infty$, pentru $n \geq 4$. ■

Se pot demonstra proprietăţi asemănătoare pentru erorile relative ale euristicilor *BS* şi *GS*.

Observaţie: Identificarea vecinătăţilor din tabelul 5.20. se efectuează astfel. Pentru $N_A(s)$ şi $N_1(s)$ modul de identificare este evident. Pentru $N_{BS}(s)$ şi $N_{GS}(s)$ se obţin succesiunile de mai jos din care se elimină, parcurgându-le de la dreapta la stânga $s(i)$-urile ce se repetă.

$N_{BS}(s)$ $k = 1, l = 2$ $g = (s(1), s(2), s(1), s(3), s(4)) = (2, 1, 3, 4)$,
$k = 1, l = 3$ $g = (s(1), s(2), s(3), s(1), s(4)) = (2, 3, 1, 4)$,
$k = 1, l = 4$ $g = (s(1), s(2), s(3), s(4), s(1)) = (2, 3, 4, 1)$,
$k = 2, l = 3$ $g = (s(1), s(1), s(3), s(2), s(4)) = (1, 3, 2, 4)$,
$k = 2, l = 4$ $g = (s(1), s(1), s(3), s(4), s(2)) = (1, 3, 4, 2)$,
$k = 3, l = 4$ $g = (s(1), s(2), s(4), s(3)) = (1, 2, 4, 3)$.

$N_{GS}(s) = N_{BS}(s) \cup$ mulţimea ce urmează în care secvenţele marcate cu * se elimină, deoarece figurează şi în $N_{BS}(s)$.

$k = 2, l = 1$ $g = (s(1), s(2), s(1), s(3), s(4)) = (2, 1, 3, 4)$, *
$k = 3, l = 1$ $g = (s(1), s(3), s(1), s(2), s(4)) = (3, 1, 2, 4)$,
$k = 3, l = 2$ $g = (s(1), s(3), s(2), s(4)) = (2, 3, 4, 1)$, *
$k = 4, l = 1$ $g = (s(1), s(4), s(1), s(2), s(3)) = (4, 1, 2, 3)$,
$k = 4, l = 2$ $g = (s(1), s(1), s(4), s(2), s(3)) = (1, 4, 2, 3)$,
$k = 4, l = 3$ $g = (s(1), s(2), s(4), s(3)) = (1, 2, 4, 3)$. *

5.4.6. Planificarea activităţilor folosind grafuri liniare

Punerea problemei. Fie $G(A, R)$ un graf orientat în care A este mulţimea nodurilor reprezentând operaţiile şi R mulţimea arcelor reprezentând ordonarea operaţiilor pe procesoare, astfel încât operaţia corespunzătoare nodului k o precede pe cea corespunzătoare nodului l dacă arcul $(k, l) \in R$ şi se notează $k \ll l$. Fiecărui arc (k, l) i se asociază o pondere t_k reprezentând timpul necesar efectuării operaţiei ce

https://biblioteca-digitala.ro / https://unibuc.ro

corespunde nodului k. Fie n numărul de activități și A_j mulțimea nodurilor asociate operațiilor ce urmează să fie efectuate pe procesorul j, m numărul de procesoare și $B_i \subset A$ mulțimea nodurilor asociate activității i. Momentele de start ale operațiilor corespunzătoare nodului k se notează cu s_k și momentele (timpii) de încheiere ale acestor operații cu c_k, unde $c_k = s_k + t_k$. Se cere să se identifice momentele de start ale operațiilor astfel încât timpul de încheiere a procesării $T(S) = \max_k [c_k]$ să fie minim. ■

Se prezintă un algoritm pentru găsirea unei soluții optimale sau aproape optimale a planificării activităților. În cadrul algoritmului se folosesc grafuri liniare pentru formularea problemei, un criteriu eficient pentru rezolvarea conflictelor și un procedeu quasi-boolean pentru evaluarea secvențelor realizabile.

Exemplul 5.46. Fie cazul a două activități ce urmează să fie efectuate pe trei procesoare, în modul descris în tabelul 5.21. și figura 5.33.

Activitatea i		1				2	
Procesorul j	1	3	2		3	1	2
Operația i_j	11	13	12		23	21	22
Nodul k	1	5	3		6	2	4
Timpul de lucru t_k	2	4	1		3	4	5

Tabelul 5.21.

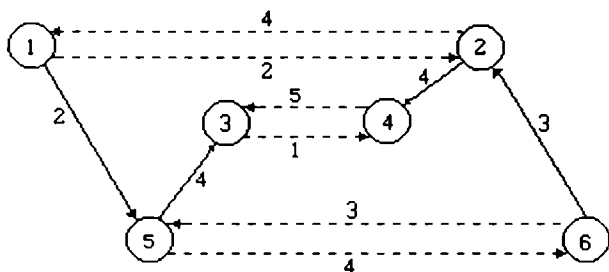


Fig. 5.33.

Ordinea de ocupare a celor trei procesoare de către cele două activități este 11 <<13 <<12, pentru activitatea 1, și 23 << 21 << 22, pentru activitatea 2.

Nodurile grafului de precedență din figura 5.33. sunt numerotate astfel încât $k=j+(i-1)n$, unde $n=2$ este numărul de activități. Rezultă că $B_1=\{1,2\}$, $B_2=\{3,4\}$ și $B_3=\{5,6\}$. Mulțimile A_1 și A_2 sunt respectiv $\{1,5,3\}$ și $\{6,2,4\}$. Arcele pline din figura 5.33. definesc ordinea ocupării procesoarelor de către activitățile 1 și 2, nodurile grafului fiind parcurse în ordinea (1,5,3) pentru activitatea 1 respectiv în ordinea (6,2,4) pentru activitatea 2. Arcele punctate definesc succesiunea posibilă a operațiilor celor două activități pe cele trei procesoare și anume: 1<<2 sau 2<<1 pe procesorul 1, 3<<4 sau 4<<3 pe procesorul 2 și 5<<6 sau 6<<5 pe procesorul 3.

https://biblioteca-digitala.ro / https://unibuc.ro

Urmărind graful din figura 5.33. și ținând seama de ordinile 11<<13<<12, respectiv, 23<<21<<22, ale operațiilor celor două activități pe cele trei procesoare, rezultă că operațiile 11, 13, respectiv 23, 21, pot fi efectuate în paralel pe procesoarele 1, 3 și 3, 1. Urmează efectuarea operațiilor 12 și 22 pe procesorul 2, fapt ce constitue un conflict, în sensul că aceste operații nu pot fi efectuate simultan. Rezolvarea acestui conflict trebuie să aibe în vedere minimizarea timpului de lucru, dar având în vedere că operațiile menționate sunt ultimele, rezultă că, independent de ordinea în care vor fi efectuate, ele vor necesita împreună 1+5=5+1=6 unități de timp.

Ținând seamă de ordinea în care au fost parcurse arcele succesive divergente din nodul 6 al grafului din figura 5.33., rezultă că timpul consumat pentru efectuarea activităților este egal cu 13 unități. Acest rezultat este ilustrat pe figura 5.34. în care s-a presupus că momentele de start ale celor două activități sunt $r \in [0, 1]$ (pe desen s-a luat $r = 0$), respectiv 0, și intervalele de timp ocupate de aceste activități sunt desenate cu linii pline, respectiv întrerupte. ■

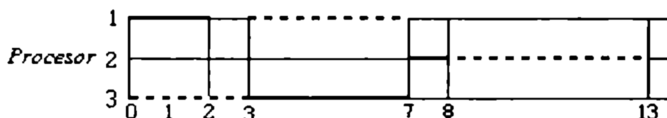


Fig. 5.34.

Plecând de la un graf de forma celui din figura 5.33., se construiește o matrice inițială de precedență Q^0 în care toate elementele asociate aceluiași procesor sunt grupate într-un același bloc. Astfel această matrice constă din m blocuri, fiecare conținând n elemente, dacă toate activitățile sunt executate pe fiecare procesor. Fiecare element $q^{\cup}_{k,l}$ al matricii este egal cu t_k dacă relația $k << l$ este precizată (arc plin în graf), xt_k dacă $k << l$ este posibilă (arce punctate în graf) și 0 dacă relația menționată este imposibilă. Această matrice este succesiv rescrisă astfel încât la fiecare iterație măcar una dar nu mai multe decât m valori xt_k trec în t_k și celelalte devin nule, ceea ce corespunde la reprezentarea prin linii pline a arcelor din graful G asociate valorilor t_k și eliminarea celor asociate valorilor devenite nule.

În procesul iterativ de atribuire de noi valori elementelor matricii pot apare situații de conflict, de forma celei exemplificate. Pentru rezolvarea acestor situații se pot folosi diferite strategii, cum ar fi: alegerea aleatoare a variantei de continuare sau utilizarea unei margini convenabil alese. Pe baza investigațiilor experimentale se propune o margine care asigură valori foarte apropiate ale timpului consumat față de timpul minimal. Această margine este definită prin atribuirea fiecărui nod k a mulțimii în conflict a valorilor expresiei:

$$L(k) = c_k + \max\Big(\sum_{\substack{l \in M_{j,\Phi} \\ l \neq k}} t_l,\ \sum_{\substack{l \in M_{j,\theta} \\ l \neq k}} t_l,\ \sum_{\substack{l \in J_{m,\Phi} \\ l \neq k}} t_l,\ \min_{\substack{m \\ l \in J_{m,\Phi}}} (s_l - t_l) - c_k + \sum_l t_l\Big),$$

https://biblioteca-digitala.ro / https://unibuc.ro

unde c_k este timpul de terminare în nodul k, θ mulţimea nodurilor în conflict şi Φ mulţimea nodurilor neasignate. Prima sumă reprezintă timpul total de procesare al nodurilor neasignate asociate aceleiaşi activităţi ca şi nodul k. A doua sumă reprezintă timpul total de procesare a nodurilor din mulţimea în conflict asociate altor procesoare decât cele asociate nodului k. A treia sumă reprezintă timpul total de procesare a nodurilor neasignate asociate aceluiaşi procesor ca şi nodul k. În sfârşit, ultimul termen reprezintă timpul minimal pentru procesarea celorlalte noduri neasignate. Din mulţimea nodurilor în conflict se alege acela pentru care $L(k)$ are o valoare minimă. Algoritmul propus este prezentat în continuare.

Algoritmul 5.35. (*Grafuri liniare*)

1. [Iniţializare] Se construieşte vectorul start iniţial: $S^0 = [s_k^0]$, $k = 1, 2, ..., n$, unde $s_k^0 = i$ dacă k nu are predecesor şi $s_k^0 = 0$ altfel. Se construieşte matricea de precedenţă cuantificată astfel încât: $Q^0 = [q_{k,l}^0]$, k, $l = 1, 2, ..., n$, unde $q_{k,l}^0 = t_k$ dacă $k << l$ este dată, $q_{k,l}^0 = xt_k$ dacă $k << l$ este posibilă şi $q_{k,l}^0 = 0$ dacă $k << l$ este imposibilă. Se face $i = 0$.
2. [Alegere] Se identifică coloana (coloanele) candidat în care elementele nenule figurează într-un rând marcat sau au coeficientul x. Dacă nu există noduri (coloane ale matricii) în conflict în nici un bloc asociat procesoarelor, se marchează fiecare coloană candidat cu "V". Dacă există un conflict într-un bloc de procesare, se calculează marginea $L(k)$ pentru fiecare nod şi se alege nodul cu $L(k)$ minim.
3. [Asignare] Se asignează nodurile având coloanele marcate. Se schimbă elementele coloanei l din matricea de precedenţă punând $xt_k = t_k$ dacă rândul k este marcat şi $xt_k = 0$ altfel. Se schimbă fiecare rând marcat k din matricea de precedenţă punând $xt_k = 0$ în toate coloanele. Se marchează modificările pe coloane cu "W" şi pe linii cu "V". Se obţine astfel matricea Q^{i+1}. Se face $i = i + 1$.
4. [Ciclare] Se determină o secvenţă realizabilă repetând paşii 2 şi 3 până când toate nodurile sunt marcate.
5. [Evaluare] Se evaluează secvenţa obţinută. Se identifică primul vector S calculând succesiv: $S^k = S^{k-1} * [S^{k-1} \# Q^k]$, $k = 1, 2, ...$ până când doi vectori succesivi devin identici. Se calculează timpul de lucru: $T(S) = \max_k (s_k + t_k)$ şi secvenţa corespunzătoare. ■

Operatorul # reprezintă un caz special de înmulţire a două matrice, la înmulţire participând numai liniile şi coloanele marcate "V" şi elementele de forma xj nu participă la înmulţire. Operatorul * semnifică concatenarea intervalelor de timp şi

https://biblioteca-digitala.ro / https://unibuc.ro

alegerea celui mai mare dintre intervale atunci când două sau mai multe rânduri ale lui Q sunt marcate cu "V".

Exemplul 5.47. Pentru datele din exemplul 5.46., calculele evoluează după cum urmează.

Pasul 1. Iniţializare graful de precedenţă. Vectorul iniţial S^0 şi matricea de precedenţă Q^0 se deduc din figura 5.33. şi au forma:

$$S^0 = [i\ 0\ 0\ 0\ 0\ i] \text{ şi } Q^0 = \begin{array}{c} \begin{array}{cccccc} V & & & & & V \end{array} \\ \begin{pmatrix} 0 & x2 & 0 & 0 & 2 & 0 \\ x4 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & x1 & 0 & 0 \\ 0 & 0 & x5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & x4 \\ 0 & 3 & 0 & 0 & x3 & 0 \end{pmatrix} \end{array}.$$

Pasul 2. Se identifică coloanele candidate în matrice, iniţial numai coloanele 1 şi 6, şi sunt marcate cu "V".

Pasul 3. Se asignează nodurile 1 şi 6, adică se anulează elementele $q_{2,1}$ şi $q_{5,6}$ din coloanele marcate. Drept rezultat se obţine matricea Q^1.

$$Q^1 = \begin{array}{c|cccccc} & W & V & & & V & W \\ \hline V & 0 & x2 & 0 & 0 & 2 & 0 \\ & 0 & 0 & 0 & 4 & 0 & 0 \\ & 0 & 0 & 0 & x1 & 0 & 0 \\ & 0 & 0 & x5 & 0 & 0 & 0 \\ & 0 & 0 & 4 & 0 & 0 & 0 \\ V & 0 & 3 & 0 & 0 & x3 & 0 \end{array},$$

Pasul 4. Se repetă paşii 2 şi 3 până când se obţine o secvenţă realizabilă. Se asignează nodurile 2 şi 5, adică $q_{1,2}$ ia valoarea 2 şi $q_{6,5}$ ia valoarea 3. Se obţine matricea Q^2.

$$Q^2 = \begin{array}{c|cccccc} & W & W & V & & W & W \\ \hline V & 0 & 2 & 0 & 0 & 2 & 0 \\ V & 0 & 0 & 0 & 4 & 0 & 0 \\ & 0 & 0 & 0 & x1 & 0 & 0 \\ & 0 & 0 & x5 & 0 & 0 & 0 \\ V & 0 & 0 & 4 & 0 & 0 & 0 \\ V & 0 & 3 & 0 & 0 & 3 & 0 \end{array}.$$

Nodurile 3 şi 4 fiind în conflict, urmează să li se calculeze marginile lor. Efectuând calculele, se obţine $L(3)=8+\max(0,5,5,5)=13$ şi $L(4)=12+\max(0,1,1,1)=13$. Cele două margini fiind egale, se alege nodul 3. Efectuând asignările rezultă matricea Q^3. În încheiere se asignează nodul 4 şi se obţine matricea Q^4.

Pasul 5. Se evaluează secvenţa. Multiplicând matricea finală Q^4 cu vectorul iniţial S^0 rezultă patru vectori, al treilea şi al patrulea fiind identici şi egali cu:

https://biblioteca-digitala.ro / https://unibuc.ro

$$S=(i\;3\;7\;8\;3\;i)$$

$$Q^3=\begin{array}{c}\\ V\\ V\\ V\\ \\ V\\ V\end{array}\begin{array}{c}\begin{matrix}W & W & W & V & W & W\end{matrix}\\ \begin{pmatrix}0 & 2 & 0 & 0 & 2 & 0\\ 0 & 0 & 0 & 4 & 0 & 0\\ 0 & 0 & 0 & x1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 4 & 0 & 0 & 0\\ 0 & 3 & 0 & 0 & 3 & 0\end{pmatrix}\end{array},$$

$$Q^4=\begin{array}{c}\\ V\\ V\\ V\\ V\\ V\\ V\end{array}\begin{array}{c}\begin{matrix}W & W & W & W & W & W\end{matrix}\\ \begin{pmatrix}0 & 2 & 0 & 0 & 2 & 0\\ 0 & 0 & 0 & 4 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 4 & 0 & 0 & 0\\ 0 & 3 & 0 & 0 & 3 & 0\end{pmatrix}\end{array}.$$

Prin urmare timpul de lucru este egal cu 13 unităţi. ■

5.4.7. Analiza euristicilor pentru programări stohastice

În activitatea de planificare managerială apar frecvent probleme cu mai multe etape de decizie. Drept exemplu se poate considera problema procesării activităţilor, în cadrul căreia există măcar două etape de decizie. În prima etapă se decide achiziţionarea resurselor. De obicei nu se ştie cu precizie cât de largă trebuie să fie alocarea resurselor, din cauza necunoaşterii exacte a desvoltărilor viitoare, sau această problemă este în mod intenţionat ignorată, în vederea facilitării luării deciziei. În continuare, la nivelul *detalierii* se decide alocarea resurselor în timp, atunci când se cunosc informaţiile relevante.

Să considerăm mai întâi cazul procesării pe maşini identice. În prima etapă se decide numărul m de *maşini paralele identice* ce trebuiesc achiziţionate, cunoscând *costul* c al unei maşini, numărul n de *activităţi* $I_1, ..., I_n$ ce urmează a fi procesate şi *distribuţia probabilităţii* vectorului $\bar{t} = (\bar{t}_1, \ldots, \bar{t}_n)$ al *timpilor de procesare* a activităţilor. În a doua etapă, după ce a fost determinat m şi este cunoscută o realizare t a lui $\bar{t}$, se decide ordinea activităţilor astfel încât fiecare maşină să fie ocupată cu cel mult o activitate, orice activitate I_j fiind procesată fără întreruperi într-un timp de mărime t_j şi nu înaintea momentului 0.

Obiectivul celei de a doua etape constă în minimizarea timpului de procesare, cunoscând valorile m şi t. Fie $C^*(m, t)$ valoarea minimă a timpului ce urmează a fi calculată. Obiectivul global constă în minimizarea, pe mulţimea realizărilor lui $\bar{t}$, a valorii medii a costului total $\bar{z}^*(m) = cm + C^*(m, \bar{t})$, adică determinarea unui m^* astfel încât să aibă loc relaţia:

https://biblioteca-digitala.ro / https://unibuc.ro

$$E\bar{z}^*(m^*) = \min_m\{cm + EC^*(m,\bar{t})\}.$$

Calculul lui $C^*(m, t)$, pentru valori date m şi t, constituie o problemă NP-dificilă, astfel încât determinarea mediei $EC^*(m,\bar{t})$ ca funcţie de m şi o distribuţie arbitrară a probabilităţii vectorului $\bar{t}$ pare imposibilă. În cadrul elaborării euristicii pentru prima etapă a problemei, se aplică o idee care este fundamentală pentru toate sistemele de planificare ierarhică şi care constă în ignorarea structurii combinatoriale fine a celei de-a doua etape şi înlocuirea lui $C^*(m, t)$ cu *marginea inferioară* T / m, unde $T = \sum_{j=1}^{n} t_j$. Problema rezultată constă în minimizarea valorii medii a marginii inferioare a lui $z^*(m)$ definită prin $\bar{z}^{LB} = cm + \bar{T}/m$, adică euristica acceptă pentru m valoarea $m = m^{LB}$, unde:

$$E\bar{z}^{LB}(m^{LB}) = \min_m\left\{cm + \frac{E\bar{T}}{m}\right\}.$$

Derivata funcţiei obiectiv se anulează pentru $m = \sqrt{E\bar{T}/c}$. Deoarece m trebuie să fie un întreg pozitiv, valoarea lui m^{LB} se determină pe calea minimizării lui $cm + E\bar{T}/m$, cu condiţia că:

$$m \in \left\{\left\lfloor\sqrt{E\bar{T}/c}\right\rfloor \left\lceil\sqrt{E\bar{P}/c}\right\rceil\right\} \cap \mathrm{N}.$$

Euristica celei de a doua etape constă în procesarea activităţilor I_1, I_2, ..., I_n cu timpi de lucru t_1, t_2, ..., t_n pe m^{LB} maşini în conformitate cu regula: activităţile sunt ordonate arbitrar şi la fiecare pas următoarea activitate din listă este asignată primei maşini devenită liberă (ca în figura 5.35.). Fie $C^{LS}(m, t)$ timpul minim în care toate activităţile sunt procesate în conformitate cu regula descrisă, pentru valori date ale lui m şi t şi fie $\bar{z}^{LS}(m) = cm + C^{LS}(m,\bar{t})$. Euristica în două etape generează o soluţie de cost total $z^{LS}(m^{LR})$.

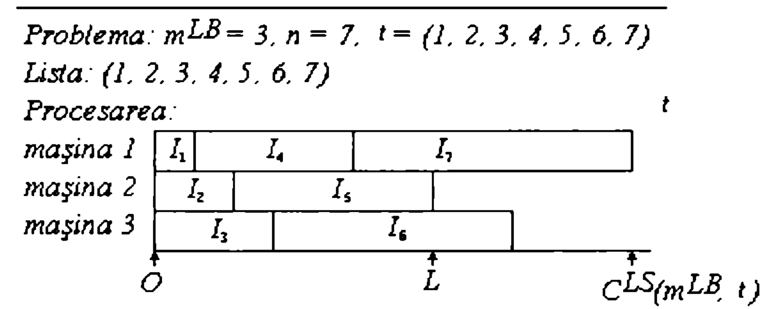


Fig. 5.35.

Valoarea medie în cel mai rău caz a euristicii propuse este caracterizată de rezultatul teoremei 5.29. Fie $t_{\max} = \max_j\{t_j\}$.

https://biblioteca-digitala.ro / https://unibuc.ro

Teorema 5.29.

$$\frac{E\bar{z}^{LS}\left(m^{LB}\right)}{E\bar{z}^{*}\left(m^{*}\right)} \leq 1 + \frac{E\bar{t}_{max}}{2\sqrt{cE\bar{T}}}.$$

Demonstrație. Fie o procesare obținută în conformitate cu regula descrisă mai sus, utilizând m^{LB} mașini pentru realizarea t a lui $\bar{t}$. Fie L cel mai lung timp cât sunt ocupate toate mașinile și I_k lucrarea cel mai târziu terminată. Din felul în care este realizată procesarea (vezi figura 5.35.) rezultă că:

$$C^{LS}\left(m^{LB}, t\right) \leq L + t_k \leq T / m^{LB} + t_{max}$$

și prin urmare:

$$z^{LS}\left(m^{LB}\right) \leq z^{LB}\left(m^{LB}\right) + t_{max}.$$

Într-adevăr, din relațiile $\bar{z}^{LS}(m) = cm + C^{LS}\left(m, \bar{t}\right)$ și $\bar{z}^{LB} = c\,m + \bar{T} / m$, pentru un vector T dat și $m = m^{LB}$, rezultă:

$$z^{LS}\left(m^{LB}\right) = cm^{LB} + C^{LS}\left(m^{LB}, t\right),\ z^{LB}\left(m^{LB}\right) = cm^{LB} + T / m^{LB} \text{ și}$$

$$z^{LS}\left(m^{LB}\right) - z^{LB}\left(m^{LB}\right) = C^{LS}\left(m^{LB}, t\right) - T / m^{LB} \leq t_{max}.$$

Luând mediile rezultă că:

$$E\bar{z}^{LS}\left(m^{LB}\right) \leq E\bar{z}^{LB}\left(m^{LB}\right) + E\bar{t}_{max} \leq E\bar{z}^{LB}\left(m^{*}\right) + E\bar{t}_{max} \leq E\bar{z}^{*}\left(m^{*}\right) + E\bar{t}_{max}.$$

A doua inegalitate rezultă din definiția lui m^{LB} și ultima deoarece $T / m^{*} \leq C^{*}\left(m^{*}, t\right)$ pentru că $\bar{z}^{*}\left(m^{*}\right) = cm^{*} + C^{*}\left(m^{*}, \bar{t}\right)$, $\bar{z}^{LB}\left(m^{*}\right) = cm^{*} + \bar{T} / m^{*}$ și T / m^{*} este margine inferioară pentru $C^{*}\left(m^{*}, t\right)$.

Pe de altă parte:

$$E\bar{z}^{*}\left(m^{*}\right) \geq E\bar{z}^{LB}\left(\sqrt{E\bar{T} / c}\right) = 2\sqrt{cE\bar{T}}.$$

Din inegalitatea de mai înainte rezultă:

$$E\bar{z}^{LS}\left(m^{LB}\right) \leq E\bar{z}^{*}\left(m^{*}\right) + E\bar{t}_{max}$$

și din cea precedentă se deduce:

$$\frac{E\bar{z}^{LS}\left(m^{LB}\right)}{Ez^{*}\left(m^{*}\right)} \leq \frac{E\bar{z}^{*}\left(m^{*}\right) + E\bar{t}_{max}}{E\bar{z}^{*}\left(m^{*}\right)} = 1 + \frac{E\bar{t}_{max}}{\mathrm{E}z^{*}\left(m^{*}\right)} \leq 1 + \frac{E\bar{t}_{max}}{2\sqrt{\mathrm{cE}\bar{t}}}. \blacksquare$$

O altă proprietate interesantă este următoarea:

$$\lim_{n \to \infty} \frac{E\bar{z}^{LS}\left(m^{LB}\right)}{E\bar{z}^{*}\left(m^{*}\right)} = 1.$$

Se poate considera o extensie a modelului studiat mai înainte, pentru cazul unei mulțimi date de *mașini neuniforme*, pentru fiecare dintre ele cunoscându-se *costul* c_j și *viteza* v_j. Atunci când în etapa a doua o activitate I_i este asignată mașinii j, ea trebuie procesată într-un interval de timp de mărime t_i / v_j.

https://biblioteca-digitala.ro / https://unibuc.ro

5.4.8. Planificări preemtive şi cu divizarea procesorilor

Unele sisteme de procesoare permit întreruperea execuţiei unei activităţi şi reluarea acesteia ulterior, eventual pe un alt procesor. Aceste planificări se numesc *planificări preemtive.* Dacă timpul necesar pentru efectuarea unei întreruperi (conservarea stării la întrerupere şi restabilirea condiţiilor la reluare) este neglijabil, programările preemtive pot să fie mai eficiente decât cele nepreemtive, după cum se vede din exemplul următor.

Exemplul 5.48. Să considerăm 3 activităţi independente de durate 2 fiecare programate pe două procesore. O planificare optimală nepreemtivă este arătată în figura 5.36.a) fiind de durată 4 şi o planificare optimală preemtivă este dată în figura 5.36.b) fiind de durată 3. ■

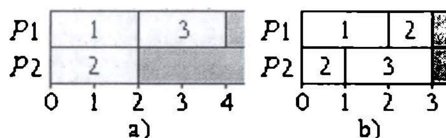


Fig. 5.36.

Planificarea preemtivă a n activităţi independente cu timpii de execuţie $t_1, t_2, \dots, t_n$ pe m procesoare identice se poate face aplicând algoritmul următor.

Algoritmul 5.36. (Planificare preemtivă)

1. [Timp probabil] Se calculează $T = \frac{1}{m}\sum_{i=1}^{n} t_i$.
2. [Depăşiri] Dacă există o activitate i astfel încât $t_i > T$, atunci se planifică activitatea i pe procesorul m, se face $m = m - 1$, se elimină activitatea i din mulţimea activităţilor, se face $n = n - 1$ şi se trece la pasul 1.
3. [Iniţializări] Se face $j = 1$, $\tau = T$ şi $i = 1$.
4. [Terminare] Dacă $i > n$, atunci STOP.
5. [Testare procesor] Dacă $t_i \le \tau$, atunci se planifică activitatea i pe procesorul j, se face $\tau = \tau - t_i$, $i = i + 1$ şi se trece la pasul 4.
6. [Preemţiune] Se face $t_i' = \tau$, $t_i'' = t_i - \tau$. Dacă $\tau \ne 0$, se planifică t_i' din activitatea i pe procesorul j.
7. [Procesorul următor] Se face $j = j + 1$, $\tau = T - t_i''$, se planifică t_i'' din activitatea i pe procesorul j. Se face $i = i + 1$ şi se trece la pasul 4. ■

Teorema 5.30. Algoritmul 5.36. determină o planificare optimă preemtivă a unui sistem de activităţi independente pe m procesoare, timpul total al planificării fiind dat de expresia:

$$t_{\min} = \max\left\{\max_i\{t_i\}, \frac{1}{m}\sum_{i=1}^{n} t_i\right\}.$$

https://biblioteca-digitala.ro / https://unibuc.ro

O demonstrație a acestei teoreme se poate găsi în [BÂS89]. ■

Exemplul 5.49. Pentru un sistem cu 7 activități care au timpii de execuție respectiv 3, 7, 2, 4, 2, 5, 2, care trebuiesc planificate pe 4 procesoare, aplicând algoritmul 5.36. se obține planificarea din figura 5.37. de valoare 7 în care activitățile 4 și 6 sunt întrerupte după ce execută 3, respectiv 4, unități de timp, urmând ca o unitate de timp să o execute fiecare pe un alt procesor. Pe figură apare numerotarea inițială. Se poate determina aici o programare nepreemtivă de aceeași durată. ■

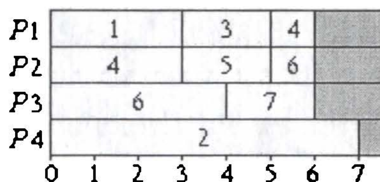


Fig. 5.37.

În planificarea cu divizarea procesorilor, se consideră că în același timp pe un procesor sunt asignate mai multe activități, fiecare activitate i folosind o anumită fracție β_i, cu $0 < \beta_i \leq 1$. Pentru a acoperi timpul t_i de execuție corespunzător activității i, în acest caz, trebuie să treacă un timp fizic t_i / β_i. Diagrama Gantt se extinde pentru aceste planificări prin introducerea unor subdiviziuni orizontale pentru a nota asignările de factor β. Factorul β nu se schimbă pe tot parcursul execuției unei activității.

Exemplul 5.50. Pentru un sistem cu 5 activități ce necesită o unitate de timp pentru fiecare și care au graful de precedență din figura 5.38.a), se poate determina programarea cu divizarea procesorilor pe 2 procesori din figura 5.38.b) de valoare 3.5. O programare optimală fără divizarea procesorilor a acestui sistem este dată în figura 5.38.c), fiind de valoare 4. ■

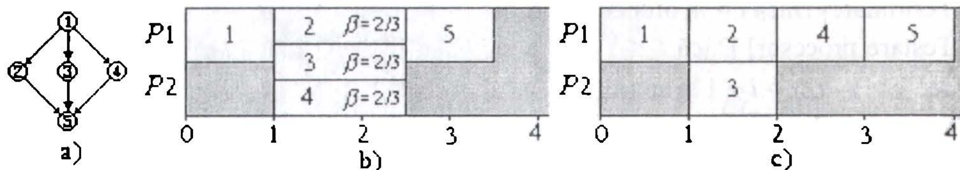


Fig. 5.38.

Dacă este permisă modificarea în timp a factorului de alocare a unui procesor la o activitate se obține o *planificare generală*. Planificările cu preemțiune și cu divizarea procesorilor pot fi considerate cazuri particulare ale planificării generale.

Exemplul 5.51. Să considerăm un sistem cu 8 activități care au timpii de execuție 1,1,2,2,3,1,1,1 și sunt planificate pe doi procesori. Graful de precedență este dat în figura 5.39.a) și o planificare generală optimală în figura 5.39.b). ■

https://biblioteca-digitala.ro / https://unibuc.ro

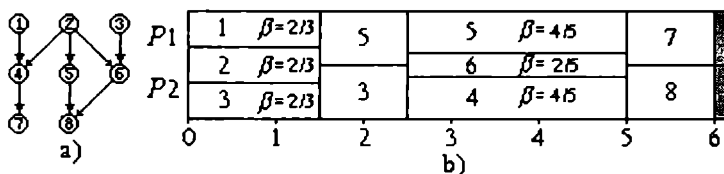


Fig. 5.39.

Pentru compararea unor strategii de planificare vom considera că strategia S_1 este *mai eficace* decât strategia S_2 dacă şi numai dacă pentru fiecare planificare P_2 obţinută aplicând strategia S_2 unui sistem de activităţi cu o relaţie de precedenţă dată se poate obţine o planificare P_1 prin strategia S_1 de durată cel mult egală cu durata lui P_2 şi există sisteme de activităţi pentru care strategia S_1 conduce la programări de durată mai mică decât cele obţinute aplicând strategia S_2. Două strategii S_1 şi S_2 sunt de *aceeaşi eficacitate* dacă planificările generate de ele au aceeaşi durată.

Exemplul 5.50. arată că planificarea cu divizarea procesorilor, deci şi planificarea generală, este mai eficace decât planificarea nepreemtivă. Exemplul 5.48. arată că planificarea preemtivă, deci şi planificarea generală, este mai eficace decât cea nepreemtivă.

Teorema 5.31. Planificarea generală şi planificarea preemtivă sunt de aceeaşi eficacitate.

Demonstrație. Este suficient de a demonstra că pentru orice planificare generală se poate determina o planificare preemtivă de aceeaşi durată. Se vede imediat că aplicând algoritmul următor pentru orice planificare generală pe m procesori pentru un sistem de activităţi cu o relaţie de precedenţă dată se obţine o programare preemtivă, validă în raport cu relația de precedenţă de aceeaşi durată. Validitatea programării rezultă din observația că activitățile planificate pe o perioadă de timp pe care nu se schimbă asignările sunt independente între ele, altfel una dintre ele ar trebui să fie terminată înaite de a începe cealaltă. ■

Algoritmul 5.37. (Transformare general-preemtiv)

1. [Inițializări] Se face $t_0 = 0$ şi $i = 0$.
2. [Terminare] Dacă t_i este egal cu timpul programării generale, atunci STOP.
3. [Interval următor] Se face $i = i + 1$ şi se determină momentul următor t_i din planificarea generală când au loc modificări în asignări (terminări, întreruperi sau modificarea coeficientului de atribuire a cel puțin unei activități).
4. [Echivalare timpi] Pentru toate activitățile $i_1, i_2, ..., i_k$ ce se execută în intervalul de timp $[t_{i-1}, t_i]$ cu factorii de ocupare $\beta_1, \beta_2, ..., \beta_k$ se consideră timpii efectivi de execuție dați de expresiile $t'_j = \beta_j (t_i - t_{i-1}), j = 1, 2, ..., k$.

https://biblioteca-digitala.ro / https://unibuc.ro

5. [Planificare preemtivă] Se aplică algoritmul 5.36. de planificare preemtivă a activităţilor i_1, i_2, ..., i_k ce se execută în intervalul de timp $[t_{i-1}, t_i]$ cu timpii t'_j, $j = 1, 2, ..., k$.
6. [Ciclare] Se trece la pasul 2. ■

Exemplul 5.52. Aplicând algoritmul 5.37. la programarea obţinută în exemplul 5.51. se obţine planificarea din figura 5.40, unde t_0=0, t_1=1.5, t_2=2.5, t_3=5 şi t_4=6. ■

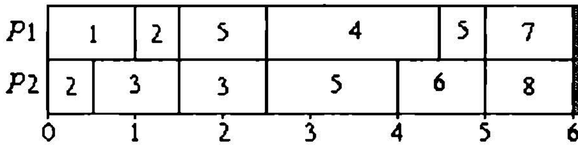
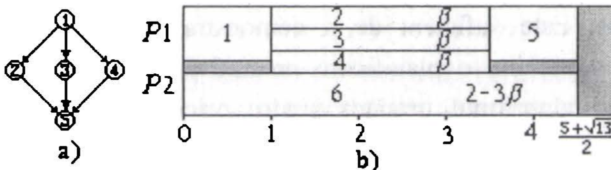


Fig. 5.40.

Teorema 5.32. Planificarea preemtivă, deci şi planificarea generală este mai eficientă decât planificarea cu divizarea procesorilor.

Demonstraţie. Deoarece planificarea cu divizarea procesorilor este un caz particular de planificare generală, aplicând proprietatea din teorema 5.31., rezultă că este suficient de dat un exemplu în care planificarea preemtivă dă un timp mai bun decât planificarea optimală cu divizarea procesoarelor. Aceasta este dovedită de exemplul 5.53. ■



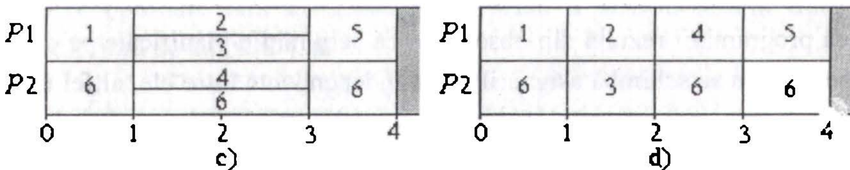


Fig. 5.41.

Exemplul 5.53. Pentru un sistem de 6 activităţi având timpii de execuţie respectiv 1,1,1,1,1,3, cu graful de precedenţă din figura 5.41.a) şi care trebuie planificate pe 2 procesori, o planificare optimă cu divizarea procesorilor are forma din figura 5.41.b). Valoarea lui β se determină din condiţia ca activităţile 5 şi 6 să se termine în acelaşi timp. Pentru toate celelalte variante valide se pot da contraexemple de programări de durate mai scurte. Se obţine astfel ecuaţia $2+1/\beta=3/(2-3\beta)$. Din aceasta rezultă ecuaţia de gradul al II-lea $3\beta^2+\beta-1=0$, care are soluţia pozitivă

https://biblioteca-digitala.ro / https://unibuc.ro

$\beta=\frac{\sqrt{13}-1}{6}$. De aici rezultă timpul programării $\frac{5+\sqrt{13}}{2} > 4$. În figura 5.41.c) este dată soluția generală optimală de valoare 4 și în figura 5.41.d) este dată soluția preemtivă corespunzătoare de aceeași durată. ■

Teorema 5.33. Fie t_N și t_P duratele planificărilor obținute la planificarea nepreemtivă, respectiv preemtivă, a unui sistem de activități pe $m > 1$ procesoare. Atunci are loc relația:

$$t_N / t_P \leq 2 - 1 / m.$$

În plus, $2 - 1 / m$ este cea mai mică limită superioară a raportului duratelor celor două programări pentru orice sistem de activități.

O demonstrație a acestei proprietății se poate găsi în [BÂS89]. ■

Determinarea unei planificări generale pentru un sistem de activități se poate face cu euristica descrisă în algoritmul 5.38. Se folosește aici noțiunea de nivel al unei activități i, notat $niv(i)$ și definit în felul următor:

$niv(i) = t_i$, pentru orice activitate i terminală;

$niv(i) = t_i + \max\{niv(j) \mid j$ succesor direct al lui $i\}$, pentru orice activitate i pentru care au fost determinate nivelurile succesorilor ei direcți.

Algoritmul 5.38. (***Programare pe nivele***)

1. [Inițializări] Se face $s_0 = 0$, $i = 0$ și se consideră toate activitățile *neexecutate*.
2. [Nivel] Se calculează $niv(j)$, pentru activitățile sistemului. Dacă există activități pentru care nu se poate face calculul, atunci STOP. (Graful are cel puțin un ciclu).
3. [Terminare] Dacă toate activitățile au fost *executate*, atunci STOP; altfel, se face $p = m$ și se consideră că nu sunt activități *în execuție*.
4. [Alegere nivel] Se determină mulțimea B a activităților pentru care au fost executate predecesoarele directe, neprogramate la execuție și de nivel maxim cu aceste proprietăți. Dacă $B = \emptyset$, atunci se merge la pasul 7.
5. [Caz fără divizare] Dacă $|B| \leq p$, atunci se programează din momentul s_i activitățile din B cu factorul 1, acestea fiind considerate *în execuție*, se face $p = p - |B|$ și dacă $p > 0$, atunci se trece la pasul 4; altfel se trece la pasul .
6. [Caz cu divizare] Se programează din momentul s_i activitățile din B cu factorul $p / |B|$, acestea fiind considerate *în execuție*.
7. [Moment realocare] Se face $i = i + 1$ și se determină $s_i = s_{i-1} + \tau$, unde τ este dat de formula:

   $\tau = \min\{\tau_1, \tau_2, \tau_3\}$, cu

   $\tau_1 = \min\{t_j / \beta_j \mid j$ activitate *în execuție*$\}$ și

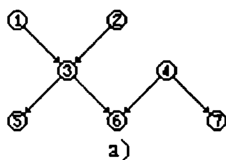
   $\tau_2 = \min\{(niv(j) - niv(k)) / (\beta_k - \beta_j) \mid j \neq k$ activități *în execuție* pentru care $niv(j) > niv(k)$ și $\beta_j < \beta_k\}$, (dacă mulțimea e vidă se ia $\tau_2 = +\infty$).

https://biblioteca-digitala.ro / https://unibuc.ro

$\tau_3 = \min\{(niv(j) - niv(k)) / \beta_j \mid j$ activitate *în execuţie* şi k activitate neprogramată, cu toţi predecesorii *executaţi* şi $niv(j) > niv(k)\}$, (dacă mulţimea e vidă se ia $\tau_3 = +\infty$).

8. [Ajustări] Pentru activităţile j *în execuţie* se face $niv(j)=niv(j)-\tau \cdot \beta_j$ şi $t_j=t_j-\tau \beta_j$; activităţile j pentru care se obţine $t_j = 0$ se consideră *executate* şi celelalte se consideră că nu sunt *executate*.
9. [Ciclare] Se merge la pasul 3. ■

Exemplul 5.54. Pentru un sistem de 7 activităţi având timpii de execuţie respectiv 2,2,2,4,1,1,2, cu graful de precedenţă din figura 5.42.a) şi care trebuie planificate pe 2 procesori, o planificare optimală generală a procesorilor care se obţine aplicând algoritmul precedent este dată în figura 5.42.b). Valorile factorilor β sunt indicate pe figură numai pentru valorile subunitare. Momentele de schimbare a asignărilor sunt s_0=0, moment iniţial, s_1=2, moment în care activitatea 4 ajunge la acelaşi nivel cu activităţile 1 şi 2, s_3=3.5, moment în care se termină activităţile 1 şi 2, s_4=4.5, moment în care se termină activitatea 4, s_5=5.5, moment în care se termină activitatea 3, şi s_6 = 7, moment în care se termină activităţile 5, 6 şi 7. Planificarea este de durată totală 7. ■



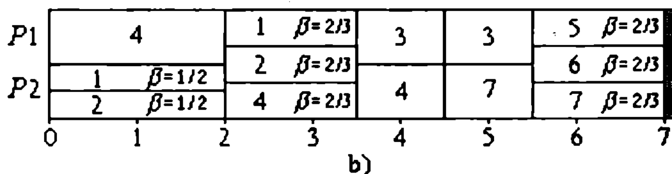


Fig. 5.42.

Teorema 5.34. Dacă se face planificare pe 2 procesori sau dacă graful asociat sistemului de activităţi este o antiarborescenţă, atunci soluţia rezultată prin aplicarea algoritmului 5.38. este optimală.

O demonstraţie a acestor proprietăţi se poate găsi în [MUN69] şi [MUN70]. ■

Faptul că algoritmul 5.38. nu furnizează soluţia optimală pentru orice precedenţă şi $m > 2$ se deduce din următorul exemplu.

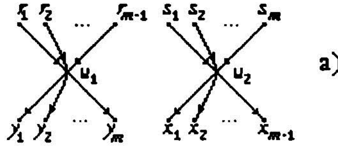
Exemplul 5.55. Fie sistemul cu $4m$ activităţi, fiecare dintre ele executându-se într-o unitate de timp şi având graful de precedenţă din figura 5.43.a), planificate pe m procesori. Pentru o mai uşoară descriere am notat activităţile cu litere cu indici. Aplicarea algoritmului 5.38. conduce la soluţia din figura 5.43.b) de durată totală $5-2/m$. O programare optimală este arătată în figura 5.43.c), fiind de durată totală 4. ■

5.4.9. Planificări cu minimizarea timpului mediu

În majoritatea planificărilor prezentate până acum, criteriul de comparare este timpul total de procesare a sistemului de activităţi. Pot exista unele aplicaţii în care să

https://biblioteca-digitala.ro / https://unibuc.ro

fie considerate şi alte criterii de evaluare a planificărilor. Vom studia în acest paragraf un astfel de caz studiind problema de planificare în care criteriul este timpul mediu în sistem al activităților în care intervin atât timpii de execuție cât şi timpii de aşteptare.



P_1	r_1 β	u_1	x_1 β	
	r_2 β		x_2 β	
P_2	⋮	u_2	⋮	
	r_{m-1} β		x_{m-1} β	
	s_1 β		y_1 β	
	s_2 β		y_2 β	
	⋮		⋮	
P_m	s_m β		y_m β	

0 1 $2-1/m$ $3-1/m$ $5-2/m$

b) $\beta=m/(2m-1)$

P_1	s_1	u_2	u_1	y_1
P_2	s_2	r_1	x_1	y_2
	s_2	r_2	x_2	y_2
	⋮	⋮	⋮	⋮
	s_{m-1}	r_{m-2}	x_{m-2}	y_{m-1}
P_m	s_m	r_{m-1}	x_{m-1}	y_m

0 1 2 3 4

c)

Fig. 5.43.

Dacă c_i este momentul terminării execuției activității i şi presupunând că toate activitățile sunt disponibile în momentul inițial de programare, atunci timpul mediu în sistem, notat $\bar{t}$, este dat de următoarea expresie:

$$\bar{t} = \sum_{i=1}^{n} c_i .$$

Se poate stabili o legătură între timpul mediu în sistem şi numărul mediu de activități neterminate din intervalul $(0,t_{\max})$, unde $t_{\max}=\max\{c_i\}$, notat cu $\bar{n}$. Numărul de activități neterminate este o variabilă discretă care ia valoarea n la momentul $t=0$ şi valoarea 0 la momentul $t=t_{\max}$. Pentru o planificare dată, considerăm activitățile ordonate crescător în raport de momentul de terminare a execuției, adică $c_1 \le c_2 \le \ldots \le c_n$. Dacă se consideră $c_0=0$, atunci:

$$\bar{n} = \frac{1}{t_{\max}} \sum_{i=0}^{n-1} (n-i)\left(c_{i+1} - c_i\right).$$

Această relație se mai poate scrie sub forma:

$$\bar{n} = \frac{1}{t_{\max}} \left\{ \sum_{i=1}^{n} (n-i+1)c_i - \sum_{i=1}^{n} (n-i)c_i \right\} = \frac{1}{t_{\max}} \sum_{i=1}^{n} c_i$$

şi, deoarece $\sum_{i=1}^{n} c_i = n\bar{t}$, rezultă relația $\frac{\bar{n}}{n} = \frac{\bar{t}}{t_{\max}}$. Aceasta arată că timpul mediu în sistem este direct proporțional cu numărul mediu de activități neterminate.

https://biblioteca-digitala.ro / https://unibuc.ro

Pentru o problemă de planificare pe un singur procesor, în orice planificare optimală, nu există timpi de neutilizare a procesorului. Altfel, prin eliminarea acestor timpi, s-ar obţine o utilizare mai bună în majoritatea criteriilor folosite. Astfel t_{max} este o constantă ce reprezintă suma timpilor de execuţie şi orice procedură care minimizează timpul mediu în sistem va minimiza numărul mediu de activităţi neterminate şi reciproc.

Să considerăm un sistem de activităţi constituit din r lanţuri independente $C_1,C_2,...C_r$, fiecare lanţ $C_i=\{A_{i1},A_{i2},...,A_{in_i}\}$, cu A_{ij} precede A_{ij+1} pentru $j=1,2,...,n_i-1$. Să notăm cu t_{ij} timpul de execuţie a activităţii A_{ij}. Vom nota $n = \sum_{i=1}^{r} n_i$.

Algoritmul 5.39. (***Timp mediu minim***)

1. [Timpi medii parţiali] Se face $p_i = 0$, pentru $i = 1, 2, ..., r$. Se calculează valorile $\bar{t}_{ij} = \frac{1}{j-p_i}\sum_{h=p_i}^{j} t_{ij}$, pentru $j = p_i + 1, p_i + 2, ..., n_i$ şi $i = 1, 2, ..., r$. Pentru fiecare lanţ i se calculează $Z_i(h_i) = \min\{\bar{t}_{ij} \mid j = p_i + 1, p_i + 2, ..., n_i\}$, unde h_i este indicele pentru care este atins minimul; dacă sunt mai mulţi indici se ia cel mai mare. Se consideră toate activităţile din toate lanţurile ca neplanificate.
2. [Terminare] Dacă toate activităţile din lanţuri au fost planificate, atunci STOP.
3. [Alegere] Pentru un procesor care devine liber se alege un lanţ C_k pentru care $p_k < n_k$, care nu conţine activităţi ce urmează să se execute pe un alt procesor în acelaşi timp şi pentru care $Z_k(h_k)$ este minim între activităţile cu aceste proprietăţi. Dacă nu există astfel de lanţuri, atunci se trece la pasul 2.
4. [Planificare] Se planifică pe procesorul liber activităţile $A_{kp_k+1},...,A_{kh_k}$ din lanţul C_k şi se face $p_k = h_k$.
5. [Ajustări] Dacă $p_k < n_k$, atunci se recalculează $Z_k(h_k)$ pentru lanţul C_k cu formulele date la pasul 1.
6. [Ciclare] Se merge la pasul 2. ■

Exemplul 5.56. Să considerăm lanţurile $C_1=\{A_{11},A_{12},A_{13},A_{14}\}$, cu timpii de execuţie 4,2,3,5, $C_2=\{A_{21},A_{22},A_{23}\}$, cu timpii de execuţie 1,4,2, $C_3=\{A_{31},A_{32},A_{33},A_{34}, A_{35}\}$, cu timpii de execuţie 3,2,3,4,6 şi $C_4=\{A_{41},A_{42}\}$, cu timpii de execuţie 2,5, executate pe 2 procesori cu minimizarea timpului mediu în sistem. Aplicând algoritmul 5.39. se obţine în pasul 1, $p_1=p_2=p_3=p_4=0$ şi timpii medii următori: 4,3,3,3.5 cu $Z_1(3)=3$ pentru lanţul C_1, 1,2.5,2.33... cu $Z_2(1)=1$ pentru lanţul C_2, 3,2.5, 2.66...,3,3.6 cu $Z_3(2)= 2.5$ pentru lanţul C_3 şi 2,3.5 cu $Z_4(1)=2$ pentru lanţul C_4.

La momentul $t = 0$, se consideră că procesorul P_1 devine liber şi se selectează dintre toate lanţurile pentru programare lanţul C_2 din care se programează

https://biblioteca-digitala.ro / https://unibuc.ro

activitateta A_{21} până la momentul 1. Se recalculează timpii medii: 4, 3 cu $Z_2(3) = 3$. Tot la momentul $t = 0$, se consideră că şi procesorul P_2 devine liber şi se selectează pentru programare dintre C_1, C_3 şi C_4 (C_2 are activităţi planificate în acelaşi timp pe P_1) lanţul C_4 din care se programează activitateta A_{41} până la momentul 2. Se recalculează timpii medii: 5 cu $Z_4(2) = 5$.

La momentul t=1, procesorul P_1 devine liber şi se selectează dintre C_1, C_2 şi C_3 (C_4 are activităţi planificate în acelaşi timp pe P_2) pentru programare lanţul C_3 din care se programează activităţile A_{31} şi A_{32} până la momentul 6. Se recalculează timpii medii: 3, 3.5, 4.33... cu $Z_3(3)$=3.

La momentul t=2, procesorul P_2 devine liber şi se selectează dintre C_1, C_2 şi C_4 (C_3 are activităţi planificate în acelaşi timp pe P_2) pentru programare lanţul C_1 din care se programează activităţile A_{11}, A_{12} şi A_{13} până la momentul 11. Se recalculează timpii medii: 5 cu $Z_1(4) = 5$.

Procedând în acelaşi fel în continuare se obţine planificarea din figura 5.44. care dă un timp mediu în sistem de 155 / 14 = 11.07.... ■

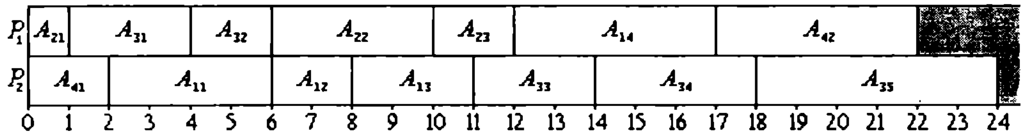


Fig. 5.44.

Teorema 5.35. Aplicând algoritmul 5.39. pentru planificarea pe un procesor se obţine un timp mediu în sistem minim.

O demonstraţie a acestei proprietăţi se găseşte în [BÂS89]. ■

Un caz particular este cel în care fiecare lanţ conţine o singură activitate. Un astfel de sistem este cu activităţi independente. Algoritmul 5.39. se poate simplifica după cum se arată în continuare şi se numeşte *cel mai scurt timp de excecuţie*, prescurtat *SPT*.

Algoritmul 5.40. (*SPT*)

1. [Iniţializări] Se consideră toate activităţile neplanificate.
2. [Terminare] Dacă toate activităţile au fost planificate, atunci STOP.
3. [Planificare] Când un procesor este liber, se planifică pe el o activitate care are cel mai scurt timp de execuţie dintre activităţile neplanificate. Dacă sunt mai multe activităţi de acest fel, atunci se alege oricare dintre ele.
4. [Ciclare] Se trece la pasul 2. ■

https://biblioteca-digitala.ro / https://unibuc.ro

SPT este, într-un anumit fel, opusă planificării *LPT* (vezi paragraful 1.1.2.3.1). Dacă se notează cu t_{SPT} timpul total obţinut aplicând planificarea *SPT* şi cu t_{LPT} timpul total obţinut aplicând planificarea *LPT*, atunci din teorema 5.33., rezultă inegalităţile:

$$1 \leq t_{SPT} / t_{LPT} \leq 2 - 1 / m.$$

Un alt caz particular este cel în care unele subşiruri din sublanţuri trebuie să fie executate fără a se putea întrerupe execuţia între două activităţi consecutive din subşirul respectiv. Se poate obţine un algoritm pentru determinarea soluţiei din algoritmul 5.39. făcându-se calculele pentru timpii medii din paşii 1 şi 3 numai pentru indicii care corespund la activităţi, după care este posibilă o întrerupere a executării activităţilor din lanţul respectiv.

5.5. Aplicaţii

5.5.1. Algoritm efectiv de asamblare a lucrărilor

Punerea problemei. O linie de asamblare constă dintr-o succesiune de staţii de lucru. Fiecare staţie efectuează o mulţime de operaţii: fiecare operaţie este efectuată fără preempţiune. Timpul de execuţie t_i al fiecărei operaţii i, unde $i = 1, 2, ..., n$, este cunoscut şi nu depinde de staţia care îl efectuează, nici de operaţiile care o preced sau o succed. Datorită restricţiilor tehnologice, unele operaţii trebuiesc efectuate înainte ca altele să fi început. Aceste condiţii de precedenţă se presupun cunoscute. Fiecărei staţii i se alocă o valoare predeterminată a timpului de lucru c, numită *durata ciclului*, în vederea operaţiilor atribuite. Lucrările sunt terminate după ce toate operaţiile au fost efectuate, $1 / c$ fiind productivitatea sistemului. ■

În continuare se consideră două variante ale problemei enunţate, numite *SALB-I* şi *SALB-II*. Prima problemă constă în determinarea numărului minim al staţiilor, necesar pentru asigurarea ratei de producţie ţinând seama de restricţiile de precedenţă. A doua problemă constă în asignarea operaţiilor unui număr dat de staţii în vederea maximizării ratei de producţie ţinând sema de condiţiile de precedenţă.

5.5.1.1. Euristici pentru problema *SALB-I*

O primă euristică pentru prima problemă se numeşte *euristica încărcării primei staţii accesibile* şi este exprimată prin următorul algoritm:

Algoritmul 5.41. (Prima staţie accesibilă)

1. [Scoruri] Se atribuie un scor $s(i)$ fiecărei operaţii i.
2. [Actualizare] Se actualizează mulţimea operaţiilor accesibile, adică a operaţiilor a căror predecesori au fost atribuiţi.

https://biblioteca-digitala.ro / https://unibuc.ro

3. [Asignare] Se asignează operaţia cu cel mai înalt scor în prima staţie în care capacitatea şi restricţiile de precedenţă sunt satisfăcute.
4. [Ciclare] Dacă sunt operaţii neasignate, atunci se trece la pasul 2, altfel STOP. ■

Rezultatele sunt influenţate de modul de alegere a funcţiei scor. Mai multe definiţii ale funcţiei scor $s(i)$ sunt date în tabelul 5.22.

Nr.	Nume	Descrierea
1.	Pondere poziţională	Suma duratelor operaţiei *i* şi a celor care o urmează
2.	Pondere poziţională inversă	Suma duratelor operaţiei *i* şi a celor care o preced
3.	Numărul urmaşilor	Numărul operaţiilor ce urmează după *i*
4.	Numărul urmaşilor imediaţi	Numărul operaţiilor ce urmează imediat după *i*
5.	Numărul predecesorilor	Numărul operaţiilor ce preced pe *i*
6.	Timpul de lucru	Durata operaţiei *i*
7.	Pondere poziţională amonte	Suma duratei operaţiei *i* şi a tuturor operaţiilor de pe drumuri având pe *i* ca rădăcină
8.	Arce amonte	Numărul de arce având drept rădăcină pe *i*

Tabelul 5.22.

Rezultatele experimentale privind eficienţa euristicii *SALB-I* pentru diferite funcţii $s(i)$ arată că pentru funcţiile numerice 1, 3, 4, 6, 7, 8 mai mult de 50% din soluţiile furnizate de euristică sunt optimale.

O altă euristică pentru prima problemă se numeşte *euristica ordonare şi asignare* şi este exprimată prin următorul algoritm:

Algoritmul 5.42. (*Ordonare şi asignare*)

1. [Scoruri] Se atribuie un scor $s(i)$ fiecărei operaţii i.
2. [Ordonare] Se ordonează operaţiile în ordinea necrescătoare a scorului.
3. [Asignare] Se asignează operaţia cu cel mai înalt scor în prima staţie în care capacitatea şi restricţiile de precedenţă sunt satisfăcute.
4. [Ciclare] Dacă sunt operaţii neasignate, atunci se trece la pasul 3, altfel STOP. ■

Această euristică a fost testată pentru funcţiile numerice 1, 3, 6, 7, 8, în toate cazurile, mai mult de 50% din soluţiile furnizate fiind optimale.

Euristicile prezentate au următoarele două proprietăţi:

1. Fiecare pereche de staţii consecutive însumează un timp de lucru care este mai mare decât c, unde timpul de lucru al unei staţii este definit ca fiind suma duratelor operaţiilor ce i-au fost asignate.
2. Fiecare staţie, eventual cu excepţia ultimei staţii, funcţionează un timp mai mare decât $c - t^*$, unde $t^* = \max_i t_i$.

https://biblioteca-digitala.ro / https://unibuc.ro

Fie S^* numărul optimal de stații. În teorema ce urmează sunt deduse marginile superioare ale performanței euristicilor propuse.

Teorema 5.36. Fie S numărul de stații identificat de un algoritm, euristic sau optimal, ce au proprietățile 1. și 2. Atunci:

a) $S \le 2S^* - 1$, b) $S < c/(c - t^*)S^* + 1$.

Demonstrație. Fie $S(j)$ mulțimea operațiilor asignate de euristică stației j. Deoarece timpul de lucru al unei stații nu depășește pe c, rezultă că:

$$S^* c \ge \sum_i t_i = \sum_{j=1}^{S} \sum_{i \in S(j)} t_i. \tag{5.28}$$

Din proprietățile 1. și 2. rezultă că:

$$\sum_{j=1}^{S} \sum_{i \in S(j)} t_i > \begin{cases} \frac{S}{2} \cdot c, & \text{dacă } S \text{ este par} \\ \frac{S-1}{2} \cdot c, & \text{dacă } S \text{ este impar} \end{cases} \tag{5.29}$$

$$\sum_{j=1}^{S} \sum_{i \in S(j)} t_i > (S-1)\left(c - t^*\right). \tag{5.30}$$

Ținând seama că dacă $a>b$, $a(=2S^*)$ și $b(=S$ sau S-1) sunt întregi pari, atunci $a \ge b+2$, (5.28) și (5.29) implică $S \le 2S^*-1$ și din (5.28) și (5.30) rezultă $S<c/(c-t^*)S^*+1$. ■

5.5.1.2. Euristici pentru problema *SALB-II*

În continuare arătăm felul în care metodele folosite pentru soluționarea problemei *SALB-I* pot fi adaptate la problema *SALB-II*. Fie $g(k)$, $k = 1, 2, \ldots$ valoarea optimală a soluției problemei *SALB-II* dacă numărul de stații este k și $f(c)$, $c \ge t^*$ valoarea optimală furnizată de *SALB-I* pentru un c dat. Pentru determinarea lui $g(k)$ sunt propuse două metode.

În *metoda marginii inferioare* se începe cu o valoare cunoscută c_L a marginii inferioare și se determină $f(c_L)$. Dacă $f(c_L)$ nu întrece pe k, atunci valoarea optimală este c_L; altfel, valoarea c_L nu este realizabilă și procesul de identificare a soluției se reia pentru $c_L + 1$.

În *metoda marginii superioare* se începe cu valoarea cunoscută c_U a marginii superioare, se face $c_m = \infty$ și se determină $f(c_U)$. Dacă $f(c_U)$ este mai mare decât k, atunci soluția optimală este c_m. Altfel se atribuie lui c_m valoarea timpului maxim de lucru al stațiilor ce corespunde marginii superioare curente c_U, se înlocuiește valoarea marginii superioare cu c_m - 1 și se continuă procesul.

În continuare se indică metoda de calcul a marginilor c_L și c_U, cu $T = \sum_i t_i$ fiind notat timpul total de lucru.

https://biblioteca-digitala.ro / https://unibuc.ro

Teorema 5.37. $c_L(k) \le g(k) \le c_U(k)$, unde:

$$c_L(k) = \max\left\{\left[\frac{T}{k}\right]^+, t^*\right\}$$

şi

$$c_U(k) = \max\left(t^*, \begin{cases} \frac{2T}{k} & \text{dacă } k \text{ este par} \\ \frac{2T}{k+1} & \text{dacă } k \text{ este impar şi } k \ge 3 \\ T & \text{dacă } k = 1. \end{cases} \right) \quad (5.31)$$

Demonstraţie. Valoarea marginii inferioare este evidentă. Valoarea marginii superioare este evident corectă dacă $g(k) = t^*$ sau $k = 1$. Fie atunci $g(k) > t^*$ şi $k \ge 2$. Din (5.29) rezultă că:

$$c \le \begin{cases} \frac{2T}{f(c)} & \text{dacă } f(c) \text{ este par} \\ \frac{2T}{f(c)-1} & \text{dacă } f(c) \text{ este impar, } f(c) \ge 3. \end{cases} \quad (5.32)$$

Fie $\varepsilon = \min_i t_i$. Din definiţiile lui f şi g şi din faptul că $g(k) > t^*$, rezultă că $f(g(k)-\varepsilon) \ge k+1$. Dacă $f(g(k)-\varepsilon) = k+1$, atunci înlocuim pe c cu $g(k)-\varepsilon$ şi pe $f(g(k)-\varepsilon)$ cu $k+1$ în (5.32) şi făcând pe ε să tindă către 0 rezultă (5.31). Dacă $f(g(k)-\varepsilon) > k+1$ şi ţinând seama că $f(g(k)-\varepsilon)$ este întreg, rezultă că $f(g(k)-\varepsilon) \ge k+2$. Înlocuind pe c cu $g(k)-\varepsilon$ şi pe $f(g(k)-\varepsilon)$ cu $k+2$ în (5.32) şi făcând pe ε să tindă către 0, se observă că $g(k) \le 2T/(k+1)$ independent de paritatea lui $f(g(k)-\varepsilon)$. Prin urmare are loc (5.31). ■

Observaţie. Din (5.29) şi (5.28) rezultă că $S < 2S^*$ dacă S este par, respectiv, $S < 2S^*+1$ dacă S este impar. În primul caz $S < 2S^*+1$ şi în al doilea caz $S \le 2S^*-1$, deci $S \le 2S^*-1$ în toate cazurile.

5.5.2. Performanţa unor euristici pentru linii de asamblare

Într-o linie de asamblare întreaga producţie este distribuită pe o mulţime $A = \{1, 2, ..., i, ..., n\}$ de elemente ale produsului. Fiecare element al produsului necesită un timp de lucru t_i şi i se asigură un beneficiu pe unitatea de timp egal cu w_i. Între unele sau toate elementele produsului există relaţii de precedenţă ce trebuie respectate în procesul de fabricare a unei unităţi a produsului. Elementele produsului sunt executate de sisteme de producţie numite *staţii*. Aceste staţii sunt aranjate secvenţial şi alcătuiesc o linie de asamblare corespunzătoare ordinii de executare a elementelor produsului. Timpul atribuit fiecărei staţii este limitat la o valoare c, egală cu durata producţiei împărţită la măsura (valoarea) ei. Beneficiul pe unitatea de timp W_j al unei staţii j este egal cu valoarea maximă a beneficiului realizat de elementele aparţinând staţiei, adică $W_j = \max\{w_i \mid i \in A_j\}$, unde A_j este submulţimea lui A ce conţine elementele asignate staţiei j.

https://biblioteca-digitala.ro / https://unibuc.ro

Problema planificării unei lucrări pe o linie de asamblare (*PLA*) constă în asignarea elementelor produsului diferitelor stații astfel încât să fie respectate relațiile de precedență și să fie minimizat costul produsului. Cu presupunerile:

α - ordinea parțială definită pe A,

$i \alpha i'$ - i este predecesorul lui i',

$i \alpha i' \Rightarrow i < i'$, adică elementele lui A sunt sortate topologic,

problema *PLA* poate fi formulată astfel:

Să se găsească o partiție $P = \{A_1, A_2, ..., A_m\}$ a lui A astfel încât:

$$Z = \sum_{i=1}^{m} W_i \cdot c \text{ este minimizată} \tag{5.33}$$

cu condițiile:

$$\sum_{i \in A_j} t_i \leq c, \text{ pentru } j = 1, 2, ..., m$$

și

$$\text{dacă } i \in A_j,\ k \in A_{j'} \text{ și } i \alpha k, \text{ atunci } j \leq j'.$$

Deoarece c din (5.33) este constantă, funcția obiectiv poate fi înlocuită cu:

$$\text{minimizează } W = \sum_{i=1}^{m} W_i. \tag{5.34}$$

Deoarece toate elementele produsului au același beneficiu pe unitatea de timp, adică același α_i, atunci funcția obiectiv (5.34) poate fi înlocuită cu minimizarea numărului de stații.

Exemplul 5.57. Fie $A = \{1, 2, 3, 4\}$; $t_1 = 1$, $t_2 = 2$, $t_3 = 1$, $t_4 = 2$; $W_1 = 2.1$, $W_2 = 1$, $W_3 = 2.1$, $W_4 = 1$; $1 \alpha 2$, $1 \alpha 3$, $2 \alpha 4$, $3 \alpha 4$; $c = 3$. Două partiții împreună cu valorile corespunzătoare ale lui m și Z sunt următoarele:

$$P_1 = \{\{1, 2\}, \{3, 4\}\}: m = 2, Z = 2.1 \cdot 3 + 2.1 \cdot 3 = 12.6,$$

$$P_2 = \{\{1, 3\}, \{2\}, \{4\}\}: m = 3, Z = 2.1 \cdot 3 + 1 \cdot 3 + 1 \cdot 3 = 12.3.$$

Partiția P_1 minimizează numărul de stații și P_2 minimizează pe Z. ■

Principiile elaborării euristicilor pot fi descrise după cum urmează. Unul sau mai multe criterii de prioritate sunt luate în considerare pentru asignarea elementelor produsului la stații, în raport de valoarea lui c și relațiile de precedență. Diferențele dintre euristici se datorează în special regulilor de prioritate, care sunt de forma:

- reguli bazate pe duratele de execuție t_i ale elementelor produsului,
- reguli bazate pe beneficiul pe unitatea de timp w_i al elementelor produsului,
- reguli de prioritate ce combină cele două reguli anterioare.

Pentru studiul performanței în cazul cel mai defavorabil vom defini mai întâi noțiunea de *euristică rezonabilă*. O euristică H este *rezonabilă* dacă are loc condiția:

$$\sum_{i \in A_j} t_i + \sum_{i \in A_{j+1}} t_i > c, j = 1, ..., m - 1, \tag{5.35}$$

https://biblioteca-digitala.ro / https://unibuc.ro

pentru toate soluțiile generate de euristica H. Dacă nu are loc condiția (5.35), atunci elemente ale produsului din stația $j+1$ pot fi asignate stației j, drept rezultat obținându-se o soluție mai bună a problemei *PLA* studiate.

Fie W^* valoarea funcției obiectiv a unei *PLA*, W^H valoarea funcției obiectiv generată de euristica H, $w_{max} = \max\{w_i \mid i \in A\}$ și $w_{min} = \min\{w_i \mid i \in A\}$.

Teorema 5.38. Pentru orice euristică H pentru rezolvarea *PLA*, raportul performanțelor în cazul cel mai defavorabil este dat de:

$$W^H / W^* \leq 2 \cdot w_{max} / w_{min}.$$

Demonstrație. În soluția optimală a *PLA*, ce conține m^* stații, măcar pentru una dintre ele $W_j = w_{max}$ și pentru toate celelalte are loc $W_j \geq w_{min}$. În consecință:

$$w_{max} + (m^* - 1) \cdot w_{min} \leq W^*. \tag{5.36}$$

Pentru toate soluțiile generate de euristica H are loc relația:

$$m^H \cdot w_{max} \geq W^H, \tag{5.37}$$

unde m^H este numărul de stații furnizat de euristica H.

Fie $m^{min} = \lceil \sum_{i \in A} t_i / c \rceil$; atunci $m^* \geq m^{min}$. Într-adevăr, numărul de stații nu poate fi mai mic decât cel realizat în cazul când timpul de funcționare al tuturor stațiilor este același și egal cu valoarea de mai sus a lui m^{min}.

Deoarece $\sum_{i \in A_j} t_i + \sum_{i \in A_{j+1}} t_i > c$, $j = 1, 2, ..., m - 1$, (vezi (5.35)) pentru toate soluțiile generate de o euristică rezonabilă H are loc următoarea inegalitate:

$$(m^H - 1) \cdot c < \sum_{i \in A_1} t_i + \sum_{i \in A_1} t_i + \sum_{i \in A_2} t_i + ... + \sum_{i \in A_{m^H - 1}} t_i + \sum_{i \in A_{m^H}} t_i + \sum_{i \in A_{m^H}} t_i = 2 \cdot \sum_{i \in A} t_i \leq 2 \cdot m^{min} \cdot c.$$

Rezultă că $m^H < 2 \cdot m^{min} + 1$. Deoarece $m^H, m^{min} \in \mathrm{N}$, rezultă $m^H \leq 2 \cdot m^{min}$, deci:

$$m^H < 2 \cdot m^*. \tag{5.38}$$

Din (5.36), (5.37) și (5.38) se deduce:

$$\frac{W^H}{W^*} \leq \frac{W^H}{w_{max} + (m^* - 1) w_{min}} \leq \frac{m^H w_{max}}{w_{max} + (m^* - 1) w_{min}} \leq \frac{2m^* w_{max}}{m^* w_{min}} = 2 \frac{w_{max}}{w_{min}}. \blacksquare$$

5.5.3. Desvoltarea capacității cu mai multe tipuri de facilități

Punerea problemei. Modelele de desvoltare ale capacității sunt folosite pentru planificarea extinderii în timp a facilităților. ■

Aplicații posibile: industriile ce necesită investiții substanțiale de capital, rețele de comunicație, servicii de energie electrică, sisteme de alimentare cu apă, procese din industria grea, servicii.

https://biblioteca-digitala.ro / https://unibuc.ro

Se studiază un model determinist de desvoltare a capacității cu $n \geq 2$ tipuri de facilități. O facilitate i reprezintă un anumit nivel de calitate. Facilitățile sunt numerotate astfel încât facilitatea i este de calitate mai bună decât facilitatea j dacă $i < j$. Aceste facilități sunt multiplicate în timp pentru satisfacerea a n tipuri de comenzi (cereri), numerotate ca mai înainte. Fiecare dintre aceste comenzi necesită o facilitate de o anumită calitate. O facilitate i este prevăzută să satisfacă în mod preferențial comanda i, dar poate de asemenea deservi orice cerere j cu $j > i$.

Costurile considerate includ costurile de desvoltare pentru fiecare facilitate și costurile de stocare a capacităților nefolosite. Funcțiile cost sunt nedescrescătoare și concave și reflectă strategia economiei sortimentelor. Fiind date comenzile în timp dealungul a T perioade, obiectivul problemei studiate constă în elaborarea unei desvoltări ce satisface comenzile la un cost minimal.

Modelul studiat presupune un număr finit de perioade temporale. Cererile de creștere, desvoltările capacităților și conversiile capacităților apar instantaneu, imediat după începutul fiecărei perioade. Sunt folosite următoarele notații:

i, j, k, l, m - Indicii facilităților. Cele n facilități sunt ordonate de la 1 la n în ordinea descrescătoare a calității.

t, u, v - Indicii perioadelor temporale. Orizontul planificării constă din T perioade.

r_{it} - Creșterile comenzii i pentru capacitate adițională în perioada t; $r_{it} \geq 0$. Comanda i poate fi efectuată de facilitatea k, $k \leq i$. Prin convenție, r_{it} se presupun întregi.

$$R_{ij}(u, v) = \sum_{t=u}^{v} \sum_{k=i}^{j} r_{kt}.$$

x_{it} - Creșterea desvoltării facilității i în perioada t.

y_{ijt} - Mărimea capacității convertită din facilitatea i în facilitatea j, $i < j$, în perioada t. O dată convertită, capacitatea devine parte integrantă a facilității j.

I_{it} - Mărimea capacității în exces a facilității i la începutul perioadei t sau, echivalent, la sfârșitul perioadei t - 1. Se presupune că $I_{i1} = 0$ și că lipsurile de capacitate nu sunt permise, adică $I_{it} > 0$.

I_t - Vectorul capacităților în exces, $I_t = (I_{1t}, I_{2t}, ..., I_{nt})$, $I_t > 0$ ($I_t = 0$) reprezintă situațiile $I_{it} > 0$ ($I_{it} = 0$) pentru toți i.

$c_{it}(x_{it})$ - Funcția cost a creșterii capacității.

$h_{ij}(I_{i,k+1})$ - Funcția costului de înmagazinare în cadrul transferării capacității în exces a facilității i din perioada t în perioada $t + 1$.

Pentru simplificare se presupune că funcțiile de cost satisfac condițiile $c_{it}(0) = h_{it}(0) = 0$. În afară de aceasta, toate funcțiile de cost se presupun nedescrescătoare și concave.

Problema desvoltării capacității (*PDC*) este formulată astfel:

https://biblioteca-digitala.ro / https://unibuc.ro

$$\underset{x_{it}, y_{ijt}}{\text{minimizează}} \left(\sum_{t=1}^{T} \sum_{i=1}^{n} \left[c_{it}(x_{it}) + h_{it}(I_{i,t+1}) \right] \right)$$

astfel încât:

$$\left. \begin{aligned} I_{i,t+1} &= I_{it} + x_{it} + \sum_{k=1}^{i-1} y_{kit} - \sum_{k=i+1}^{n} y_{ikt} - r_{it}, \\ I_{it} &\geq 0, \\ I_{i1} &= I_{i,T+1} = 0, \\ x_{it} &\geq 0, y_{ijt} \geq 0, \end{aligned} \right\} \quad \begin{aligned} t &= 1, 2, \ldots, T \\ i &= 1, 2, \ldots, n \\ j &= i+1, \ldots, n, \end{aligned}$$

unde $\sum_{1}^{0}(\bullet)$ şi $\sum_{n+1}^{n}(\bullet)$ sunt prin definiție nule. Obiectivul *PDC* constă în minimizarea costului total pe durata a *T* perioade. Prima condiție descrie capacitatea în exces a facilității *i* la începutul perioadei $t + 1$ ca funcție de capacitatea în exces la începutul perioadei *t* şi schimbarea capacității în perioada *t*. Constrângerile $I_{it} \geq 0$ şi $I_{i1} = 0$ sunt luate ca ipoteze. În afară de aceasta, deoarece funcțiile cost sunt nedescrescătoare, se poate presupune că $I_{i,T+1} = 0$ fără a creşte costurile totale. Notația PDC_{ij}, $j \geq i$, se va referi în continuare la o *PDC* cu facilitățile i, $i + 1$, ..., j. Rezultă că formularea precedentă se referă la PDC_{1n}.

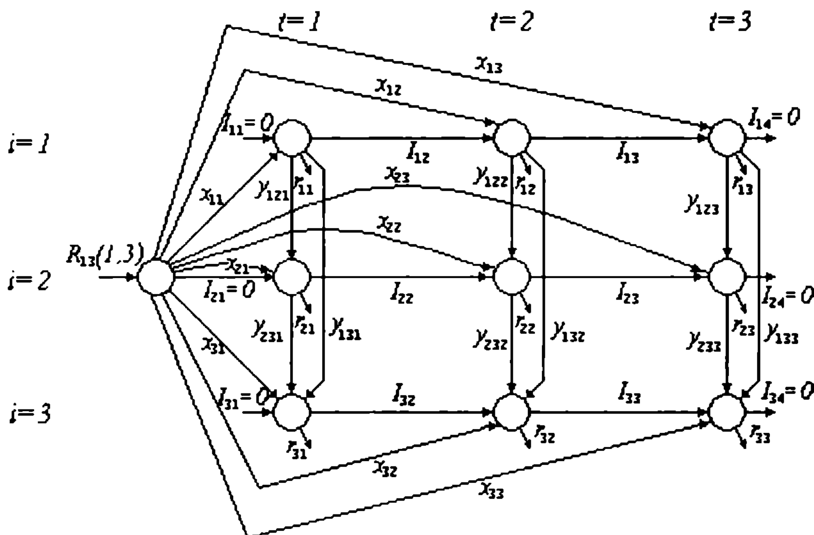


Fig. 5.45.

Schema reprezentativă a *PDC* pentru $n = 3$ şi $T = 3$ este arătată în figura 5.45. Deoarece funcțiile cost sunt concave, există o soluție optimală reprezentată printr-un punct extremal. În reprezentarea pe schemă, aceasta înseamnă că există o soluție optimală cu nu mai mult de un flux pozitiv incident în fiecare nod.

Ideea metodei constă în rezolvarea repetată a unei probleme cu o singură perioadă. Fie $SPDC_{ij}(u; D_i, D_{i+1}, \ldots, D_j)$ o problemă de desvoltare cu perioada u pentru

https://biblioteca-digitala.ro / https://unibuc.ro

facilitățile i, $i+1$, ..., j și creșterile corespunzătoare D_i, D_{i+1}, ..., D_j ale cererii. Problema se formulează astfel:

$$S_{ij}(u;\, D_i, D_{i+1}, \ldots, D_j) = \underset{x_{ku}, y_{kmu}}{\text{minimum}}\left[\sum_{k=i}^{j} c_{ku}(x_{ku})\right],$$

astfel încât:

$$x_{ku} + \sum_{l=i}^{k-1} y_{lku} - \sum_{l=k+1}^{j} y_{klu} = D_k,\; x_{ku} \geq 0, y_{kmu} \geq 0,\; k = i, \ldots, j;\; m = k+1, \ldots, j,$$

unde $\sum_{i}^{i-1}(\bullet)$ și $\sum_{j+1}^{j}(\bullet)$ sunt nule prin definiție.

Schema reprezentativă a *SPDC* este arătată în figura 5.46.

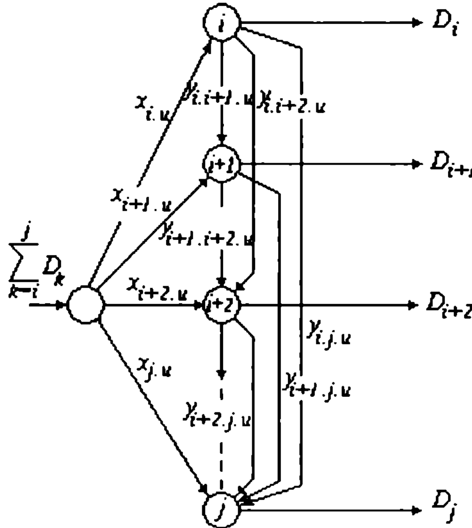


Fig. 5.46.

Algoritmul 5.43. (***Euristica de bază***)

Creșterile capacităților și conversiile sunt considerate numai în perioadele în care $I_t = 0$. Fie $PDC_{1n}(u, v)$ problema desvoltării pentru facilitățile 1, 2, ..., n pe perioadele u, ..., v știind că $I_u = I_{v+1} = 0$, $I_t > 0$, pentru $t = u + 1, \ldots, v$ și presupunând că toate creșterile și conversiile au loc în perioada u. Soluția optimală $PDC_{1n}(u, v)$, notată $d(u, v)$, se obține rezolvând problema $SPDC_{1n}(u;\, D_i, \ldots, D_j)$. Mai precis:

$$d(u, v) = S_{1n}(u;\, D_i, \ldots, D_j) + C(I_u, I_{v+1}),$$

unde:

$$D_i = R_{ii}(u, v),\; i = 1, 2, \ldots, n,\; I_u = I_{v+1} = 0 \text{ și } C(I_u, I_{v+1}) = \sum_{t=u}^{v} \sum_{i=1}^{n} h_{it}(I_{i,t+1}). \;\blacksquare$$

Euristica calculează toate cele $T\,(T + 1)\,/\,2$ valori posibile ale lui $d(u, v)$ și identifică secvența optimală a problemelor *PDC* utilizând programarea dinamică.

https://biblioteca-digitala.ro / https://unibuc.ro

6 PROGRAMARE LINIARĂ ÎNTREAGĂ

6.1. Algoritmi euristici pentru programarea liniară întreagă

Punerea problemei. Se cere maximizarea lui x_0 dat de expresia:

$$x_0 = \sum_{j=1}^{n} c_j x_j$$

cu restricțiile:

$$\sum_{j=1}^{n} a_{ij} x_i \leq b_i, \text{ pentru } i = 1, 2, ..., m, \tag{6.1}$$

$$x_j \geq 0, \text{ pentru } j = 1, 2, ..., n, \tag{6.2}$$

$$x_j \text{ este întreg, pentru } j = 1, 2, ..., n.$$

Se presupune că regiunea din spațiul n-dimensional a punctelor ce satisfac condițiile (6.1) şi (6.2) conține cel puțin un punct interior. ■

Fiecare dintre procedurile de rezolvare propuse conțin trei faze. Scopul primei faze constă în identificarea unei regiuni în interiorul căreia se caută soluții suficient de bune. În acest scop se selectează mai întâi un punct reprezentând o soluție optimală neîntreagă. În continuare se identifică un al doilea punct, suficient de apropiat de primul şi satisfăcând condiția că segmentul ce îl uneşte cu primul punct aparține regiunii ce conține soluții ale problemei. În cadrul fazei a doua se încearcă identificarea unei soluții cât mai apropiată de o soluție întreagă, prin deplasarea dealungul segmentului anterior menționat. Scopul fazei a treia constă în încercarea de îmbunătățire a soluției obținute în faza anterioară pe calea schimbării valorii uneia sau simultan a două variabile până când nu se mai poate obține o îmbunătățire a soluției determinate la un moment dat.

Pentru unele dintre metodele din fazele 1 şi 2, este necesară normalizarea restricțiilor. Pentru acasta se introduc notațiile:

https://biblioteca-digitala.ro / https://unibuc.ro

$$a'_{ij} = a_{ij} \Big/ \sqrt{\sum_{j=1}^{n} a_{ij}^2}\text{, pentru } i = 1, 2, ..., m; j = 1, 2, ..., n$$

şi

$$b'_i = b_i \Big/ \sqrt{\sum_{j=1}^{n} a_{ij}^2}\text{, pentru } i = 1, 2, ..., m,$$

cu ajutorul cărora restricţiile (6.1) iau forma:

$$\sum_{j=1}^{n} a'_{ij} x_j \le b'_i\text{, pentru } i = 1, 2, ..., m.$$

Rezultă că b'_i reprezintă distanţa euclidiană de la origine la planul $\sum_{j=1}^{n} a_{ij} x_j = b_i$. În continuare, se vor folosi notaţiile a_{ij} şi b_i numai în cazul în care faptul că restricţiile au fost normalizate nu este relevant.

6.1.1. Metodele fazei 1

Primul pas al *fazei 1* constă în utilizarea metodei simplex pentru a găsi o soluţie optimală neîntreagă. Fie $\bar{x}^{(1)}$ vectorul cu n componente reprezentând soluţia, $B = \{j \mid x_j^{(1)}$ este o variabilă de bază$\}$ şi N numărul de elemente din B. În continuare se identifică o nouă soluţie $\bar{x}^{(2)}$, suficient de apropiată de soluţia $\bar{x}^{(1)}$.

În cadrul *metodei 1* se efectuează mai întâi înlocuirea lui b_i cu:

$$b_i^{(2)} = b_i - \frac{1}{2} \sum_{j \in B} \left|a_{ij}\right|,$$

după care se identifică soluţia $\bar{x}^{(2)}$ satisfăcând condiţia:

$$\sum_{j=1}^{n} a_{ij} x_j^{(2)} = b_i^{(2)}.$$

În virtutea acestei condiţii rezultă că valoarea tuturor variabilelor din $\bar{x}^{(2)}$ poate fi modificată cu cel mult 1/2 fără a contrazice versiunea iniţială a restricţiilor:

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i.$$

Mai mult, $\bar{x}^{(1)}$ şi $\bar{x}^{(2)}$ au aceleaşi variabile de bază (principale) astfel încât condiţiile de nenegativitate pentru $\bar{x}^{(1)}$, ignorând variabilele de bază degenerate, sunt satisfăcute şi de $\bar{x}^{(2)}$. Rezultă că $\bar{x}^{(2)}$ poate fi modificat pe calea rotunjirii fiecăreia din componentele sale neîntregi până la cel mai apropiat întreg fără a viola condiţiile funcţionale sau de nenegativitate. În cadrul *fazei 2*, căutarea lui $\bar{x}^{(2)}$ plecând de la $\bar{x}^{(1)}$ furnizează măcar o soluţie rotunjită care satisface condiţiile problemei, deoarece $\bar{x}^{(2)}$ le satisface.

În mod asemănător, în cadrul *metodei 2* are loc înlocuirea fiecărui b'_i cu:

https://biblioteca-digitala.ro / https://unibuc.ro

$$b_i'^{(2)} = b_i' - \sqrt{N} / 2,$$

în cazul când condiția funcțională i are loc pentru $\bar{x}^{(1)}$, după care se calculează noua soluție $\bar{x}^{(2)}$ pentru această bază. Motivarea pentru această alegere a lui $\bar{x}^{(2)}$ este dublă. Mai întâi, această soluție are aceeaşi propietate ca şi în *metoda 1* şi anume variabilele ei de bază pot fi rotunjite până la cel mai apropiat întreg, fără a viola nici una dintre condițiile funcționale pentru $\bar{x}^{(1)}$. Acest lucru are loc datorită faptului că distanța dintre hiperplanul:

$$\sum_{j=1}^{n} a_{ij}' = b_i'$$

şi planul corespunzător pe care este situată extremitatea vectorului $\bar{x}^{(2)}$ este egală cu $\sqrt{N}$ / 2, deoarece fiecare dintre cele N variabile de bază se schimbă cu cel mult 1/2. A doua motivare constă în faptul că punctul $\bar{x}^{(2)}$ este echidistant la hiperplanurile:

$$\sum_{j=1}^{n} a_{ij}' x_j = b_j',$$

la intersecția cărora se află extremitatea vectorului $\bar{x}^{(1)}$.

6.1.2. Metodele fazei 2

Metoda 1 a *fazei 2* constă în deplasarea continuă dealungul segmentului de la $\bar{x}^{(1)}$ la $\bar{x}^{(2)}$ efectuând rotunjirea către cea mai apropiată soluție întreagă, până când se obține o soluție a problemei. Obținerea soluțiilor succesive întregi are loc pe calea identificării punctelor, de pe segmentul menționat, în care soluțiile rotunjite se schimbă. În acest scop, fiecare punct $\bar{x}$ al segmentului se reprezintă sub forma:

$$\bar{x} = (1-\alpha)\,\bar{x}^{(1)} + \alpha\,\bar{x}^{(2)}, \text{ unde } 0 \le \alpha \le 1.$$

Pentru componenta x_j a vectorului $\bar{x}$ rezultă expresia:

$$x_j = (1-\alpha)\,x_j^{(1)} + \alpha\,x_j^{(2)} = x_j^{(1)} + \alpha\left(x_j^{(2)} - x_j^{(1)}\right),$$

cu ajutorul căreia se pot calcula valorile succesive ale lui α pentru care x_j este coordonata de indice j a mijlocului segmentului definit de două valori întregi succesive. Metoda se încheie când α a atins o valoare mai mică sau egală cu 0 sau dacă depăşeşte valoarea 1. Datorită modului în care a fost ales punctul $\bar{x}^{(2)}$, soluția rotunjită ce satisface condițiile la limită satisfăcute de $\bar{x}^{(1)}$ se obține pentru o valoare a lui $\alpha \le 1$. Nu este însă sigur că o asemenea soluție satisface şi celelalte condiții, aşa încât metoda propusă nu garantează obținerea unei soluții corecte.

În cadrul *metodei 2* se efectuează creşterea succesivă a lui α cu pas constant. Pentru fiecare valoare a lui α, începând cu α = 0, primul pas în căutarea soluției constă în aplicarea unei rotunjiri în componentele lui $\bar{x}$ în vederea identificării unei

https://biblioteca-digitala.ro / https://unibuc.ro

cele mai apropiate soluţii întregi nenegative. Dacă o astfel de soluţie rotunjită nu este realizabilă, se încearcă dacă, în urma creşterii sau descreşterii unei variabile oarecare cu 1, are loc descreşterea, în modul definit mai jos, a *nerealizabilităţii* q. Dacă nu există asemenea variabile, se trece la o nouă valoare a lui α. Dacă însă există o singură asemenea variabilă, atunci se efectuează schimbarea cu 1 a valorii ei. Dacă soluţia obţinută nu este realizabilă, se repetă încercarea precedentă pentru celelalte variabile. Este posibil să existe mai multe asemenea variabile, caz în care se efectuează schimbarea care maximizează *îmbunătăţirea*, definită mai jos; dacă soluţia obţinută este în continuare nerealizabilă, se reia parcurgerea celorlalte variabile. Metoda se încheie când s-a obţinut o soluţie realizabilă sau dacă α a depăşit 1.

O măsură cu o singură valoare a gradului de nerealizabilitate a unei soluţii $\bar{x}$ se poate obţine plecând de la două definiţii distincte ale *nerealizabilităţii* q. Prima dintre ele este:

$$q = \sum_{i=1}^{m}\left(\sum_{j=1}^{n} a'_{ij}x_j - b'_i\right)_+$$

adică suma distanţelor euclidiene dintre $\bar{x}$ şi fiecare dintre hiperplanurile pentru care condiţia (6.1) nu este respectată. O a doua definiţie este:

$$q = \max_{i\in\{1,\dots,m\}}\left\{\sum_{j=1}^{n} a'_{ij} - b'_i\right\},$$

adică distanţa euclidiană maximă dintre $\bar{x}$ şi hiperplanurile pentru care condiţia (6.1) nu este respectată. Pentru o valoare aleasă a lui q, o măsură naturală a *îmbunătăţirii*, obţinută pe calea schimbării valorii variabilei x_j, este:

$$p = -\Delta q,$$

unde Δq este schimbarea lui q datorată schimbării lui x_j. O altă alternativă este:

$$p = c_j \Delta x_j / (-\Delta q),$$

unde Δx_j este schimbarea efectuată a valorii x_j.

Definiţiile precedente generează patru criterii distincte pentru alegerea variabilei a cărei valoare urmează să fie schimbată. Aceste criterii sunt:

Criteriul A: prima definiţie a lui p, prima definiţie a lui q.

Criteriul B: prima definiţie a lui p, a doua definiţie a lui q.

Criteriul C: a doua definiţie a lui p, prima definiţie a lui q.

Criteriul D: a doua definiţie a lui p, a doua definiţie a lui q.

Alte două metode au la bază ideea rotunjirii componentelor lui $\bar{x}^{(1)}$. Prima dintre ele, folosită numai în cazul în care toate valorile a_{ij} sunt nenegative, constă în înlocuirea lui x_j cu $\lceil x_j^{(1)} \rceil$, pentru $j = 1, 2, \dots, m$, unde $\lceil y \rceil$ este cel mai mare întreg mai mic sau egal cu y. Cealaltă metodă constă în parcurgerea tuturor variantelor de

https://biblioteca-digitala.ro / https://unibuc.ro

rotunjire a componentelor lui $\overline{x}^{(1)}$ folosind atât rotunjirea precedentă cât şi rotunjirea $\lfloor x_j^{(1)} \rfloor$, pentru $j = 1, 2, ..., m$, unde $\lfloor y \rfloor$ este cel mai mic întreg mai mare sau egală cu y.

6.1.3. Metodele fazei 3

Scopul *fazei 3* constă în căutarea unei soluţii mai bune decât soluţia $\overline{x}^{(F)}$ obţinută în *faza 2*. Această căutare se efectuează în două moduri. Primul dintre ele constă în creşterea sau descreşterea succesivă cu 1 a valorii unei variabile astfel încât noua soluţie să fie realizabilă şi mai bună, adică să corespundă la o valoare mai mare a lui x_0. Al doilea mod constă în căutarea unei soluţii mai bune pe calea schimbării succesive a valorilor a două variabile. Aceste două strategii se aplică alternativ până când nu se poate realiza o soluţie mai bună.

În cazul primului mod sunt folosite două criterii privind alegerea variabilei a cărei valoare urmează să fie schimbată. Primul criteriu se bazează pe ideea de a evita apropierea de mulţimea frontieră a soluţiilor, în vederea asigurării unor şanse mai mari următoarelor îmbunătăţiri. Al doilea criteriu de alegere constă în determinarea variabilelor ce asigură o apropiere cât mai mare de valoarea x_0 a funcţiei obiectiv.

Cele două metode utilizate în *faza 3* se deosebesc prin aceea că în prima dintre ele se impune restricţia ca valorile mărimilor ce participă la procesul de ameliorare a soluţiei $\overline{x}^{(F)}$ să fie întregi.

6.1.4. Exemplu de programare liniară întreagă

Exemplul 6.1. Se cere maximizarea lui

$$x_0 = x_1 + 5\, x_2,$$

cu restricţiile:

$$x_1 + 10\, x_2 \le 20, \quad (6.3)$$

$$x_1 \le 2, \quad (6.4)$$

$$x_1 \ge 0, x_2 \ge 0,$$

$$x_1 \text{ şi } x_2 \text{ sunt întregi.} \quad (6.5)$$

Reprezentarea grafică a restricţiilor, dată în figura 6.1., permite identificarea soluţiei optimale a problemei date, care este $(x_1, x_2) = (0, 2)$, cu $x_0 = 10$.

În vederea aplicării fazei 1 a algoritmului, restricţia (6.5) este eliminată, soluţia neîntreagă furnizată fiind $\overline{x}^{(1)}$=(2,9/5). Rezultă că baza este B={1,2} şi N=2. În cadrul metodei 1 de identificare a lui $\overline{x}^{(2)}$, termenii din dreapta ai restricţiilor (6.3) şi (6.4) sunt înlocuiţi cu $b_1^{(2)}$=20-(1/2) (1+10)=14.5, respectiv $b_2^{(2)}$=2-(1/2) (1)=3/2. Noua soluţie obţinută este $\overline{x}^{(2)}$=(3/2,13/10).

https://biblioteca-digitala.ro / https://unibuc.ro

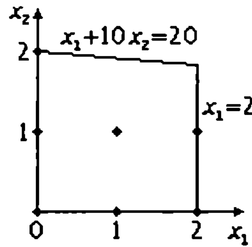


Fig. 6.1.

Dacă se aplică metoda 2, este necesar să se normalizeze restricțiile (6.3) și (6.4), care au acum forma:

$$(1/\sqrt{101})\,x_1 + (10\sqrt{101})\,x_2 \le 20/\sqrt{101} - \sqrt{2}/2,$$
$$x_1 \le 2 - 2/\sqrt{2}$$

și soluția optimală neîntreagă este:

$$\bar{x}^{(2)} = (2 - \sqrt{2}/2,\ 9/5 - (\sqrt{202} - \sqrt{2})\,/\,20) \approx (1.29,\ 1.16).$$

Cu aceasta se încheie faza 1.

Toate trei metodele fazei 2 încep cu rotunjirea soluției $\bar{x}^{(1)} = (2, 9/5)$ la cea mai apropiată soluție întreagă care este $(x_1, x_2) = (2, 2)$ și nu satisface condițiile date. În continuare, metoda 1 identifică o nouă soluție plecând de la $\bar{x}^{(2)} = (3/2, 13/10)$. Pentru valori succesive ale lui α, obținute din $\alpha = 0$ cu pasul 0.1, se constată că pentru $\alpha = 0.6$, valorile lui x_1 și x_2 sunt respectiv egale cu:

$$x_1 = (1 - 0.6)\,2 + (0.6)\,3/2 = 17/10,\ x_2 = (1 - 0.6)\,9/5 + (0.6)\,13/10 = 3/2.$$

O nouă soluție rotunjită, diferită de cea precedentă, este $(x_1, x_2) = (2, 1)$. Deoarece această soluție satisface restricțiile problemei, metoda 1 se încheie.

În cadrul metodelor 2 și 3, soluția rotunjită inițială (2, 2) nu este abandonată și se încearcă obținerea din ea a uneia ce satisface condițiile. Această soluție rotunjită nu satisface doar restricția (6.3) și valoarea corespunzătoare a lui q este $2/\sqrt{101}$. Dacă valoarea lui x_1 se micșorează cu 1, atunci valoarea lui q scade cu $1/\sqrt{101}$ și $c_1\,\Delta x_1 = -1$. Dacă se scade x_2 cu 1, atunci q scade cu $2/\sqrt{101}$ și $c_2\,\Delta x_2 = -5$. Deoarece în al doilea caz se obține o scădere mai mare pentru q, criteriile A și B vor alege scăderea lui x_2 cu 1, drept rezultat obținându-se soluția realizabilă $(x_1, x_2) = (2, 1)$. Deoarece $-1\,/\,(1\,/\,\sqrt{101}) > -5\,/\,(2\,/\,\sqrt{101})$, criteriile C și D vor alege scăderea cu 1 a lui x_1, drept rezultat obținându-se soluția nerealizabilă $(x_1, x_2) = (1, 2)$.

Faza 3 începe cu soluția realizabilă (2, 1) obținută în faza 2. Încercările bazate pe schimbarea valorii unei singure variabile nu conduc la obținerea unei soluții mai bune. Dacă se încearcă schimbarea valorilor a două variabile, se obține soluția $\bar{x} = (0, 2)$ care este și cea optimală. Dacă în faza 3 se începe cu soluția realizabilă (1, 2) se obține soluția optimală schimbând o singură valoare. ■

https://biblioteca-digitala.ro / https://unibuc.ro

6.2. Algoritm aproximativ pentru programarea 0-1

În cele ce urmează vom folosi următoarele notații:
m - numărul de proiecte (lucrări),
n - numărul resurselor restrânse,
P_i - proiectul i, $i = 1, 2, ..., m$,
R_j - resursa restrânsă j, $j = 1, 2, ..., n$,
K_i - profitul proiectului P_i,
L_j - limita superioară a resursei R_j,
H_{ij} - cantitatea din R_j solicitată de P_i,
F_{ij} - raportul dintre H_{ij} şi L_j, adică $F_{ij} = H_{ij} / L_j$,
Z - funcţia obiectiv ce trebuie maximizată,
X_i - variabile binare, adică $X_i = 0$ sau 1,
T - mulţimea tuturor proiectelor,
T_u - mulţimea proiectelor acceptate,
T_D - mulţimea proiectelor ce nu aparţin lui T_u, adică $T_D = T - T_u$,
$\overline{P_i}$ - vectorul cantităţilor de resurse pentru P_i, adică $\overline{P_i} = (F_{i1}, F_{i2}, ..., F_{in})$,
C_j - cantitatea din R_j cerută de mulţimea proiectelor acceptate, adică $C_j = \sum_{P_i \in T_u} F_{ij}$,
$\overline{P_u}$ - vectorul cumulativ total, adică $\overline{P_u} = (C_1, C_2, ..., C_n)$,
$|\overline{P_u}|$ - modulul (lungimea) lui $\overline{P_u}$, adică $|\overline{P_u}| = \left(\sum_{j=1}^{n} C_j^2\right)^{1/2}$,
$\overline{B}$ - vectorul limită normalizat al resurselor, adică $\overline{B} = (1, 1, ..., 1)$,
T_C - mulţimea proiectelor candidat, adică $T_C = \{P_i \mid P_i \in T_D, \overline{P_i} \le \overline{B} - \overline{P_u}\}$,
$\overline{E}$ - vectorul unitate al lui $\overline{P_u}$,
U_i - proiecţia lui $\overline{P_i}$ pe $\overline{E}$, adică $U_i = \overline{P_i} \cdot \overline{E}$,
G_i - gradientul efectiv al lui $\overline{P_i}$ (definit în textul algoritmului).

Punerea problemei. Problema programării 0-1 are enunţul:

Problema 1:

$$\text{Maximizează } Z = \sum_{i=1}^{m} K_i X_i,$$

cu condiţiile:

$$\sum_{i=1}^{m} H_{ij} X_i \le L_j, \text{ pentru } j = 1, 2, ..., n,$$

$$X_i = 0 \text{ sau } 1, \text{ pentru } i = 1, 2, ..., m. \blacksquare$$

În continuare se presupune $K_i > 0$, $H_{ij} \ge 0$ şi $L_j > 0$. O metodă pentru rezolvarea Problemei 1 este prezentată în lucrarea [SEN68] şi este numită *metoda duală a gradientului efectiv*, deoarece începe cu o soluţie nerealizabilă, urmată de

https://biblioteca-digitala.ro / https://unibuc.ro

identificarea unei soluţii realizabile. Metoda descrisă în continuare se numeşte *metoda primală a gradientului efectiv*, deoarece începe cu o soluţie realizabilă urmată de îmbunătăţirea ei.

Metoda primală a gradientului efectiv. Problema 1 poate fi reformulată astfel:

Problema 2:

$$\text{Maximizează } Z = \sum_{i=1}^{m} K_i X_i,$$

cu condiţiile:

$$\sum_{i=1}^{m} F_{ij} X_i \leq 1, \text{ pentru } j = 1, 2, ..., n, \tag{6.6}$$

$$X_i = 0 \text{ sau } 1 \text{ pentru } i = 1, 2, ..., m.$$

În procesul de selectare a unui proiect, este necesar să se selecteze un proiect cât mai profitabil. În cazul când proiectele considerate necesită o singură resursă, profiturile dorite se pot calcula uşor, acceptând drept indice de profitabilitate profitul pe unitatea de resursă. Problema devine dificilă în cazul a mai multor resurse limitate. În acest caz se introduce noţiunea de *vector de penalizare*, care furnizează un singur indice, adică un indice global, calculat cu ajutorul mai multor resurse limitate. Drept vector de penalizare se consideră vectorul $\overline{P}_u$.

Exemplul 6.2. Fie cazul n=2 cu $\overline{P}_u$=(0.6;0.3), $\overline{P}_1$=(0.1;0.4) şi $\overline{P}_2$=(0.4;0.1). Penalizările (totale) ale proiectelor P_1 şi P_2 sunt:

$$\text{pentru } P_1 : 0.1 \cdot 0.6 + 0.4 \cdot 0.3 = 0.18,$$

$$\text{pentru } P_2 : 0.4 \cdot 0.6 + 0.1 \cdot 0.3 = 0.27,$$

deci $P_1 < P_2$. După cum se vede, cele două penalizări reprezintă produsele scalare $\overline{P}_1 \cdot \overline{P}_u$, respectiv $\overline{P}_2 \cdot \overline{P}_u$. Ţinând seama că vectorul unitate al lui $\overline{P}_u$ este $\overline{E} = \overline{P}_u / |\overline{P}_u|$, rezultă că:

$$(\overline{P}_i \cdot \overline{P}_u) / |\overline{P}_u| = \overline{P}_i \cdot (\overline{P}_u / |\overline{P}_u|) = \overline{P}_i \cdot \overline{E} = U_i,$$

deci U_i reprezintă proiecţia lui $\overline{P}_i$ pe direcţia lui $\overline{P}_u$. Figura 6.2. ilustrează pe U_1 şi U_2 pentru cazul $\overline{P}_u$=(0.6;0.3), $\overline{P}_1$=(0.4;0.1) şi $\overline{P}_2$=(0.1;0.4). În cazul când se calculează probabilitatea unui proiect, se poate folosi gradientul efectiv al resursei definit prin:

$$G_i = K_i / U_i. \blacksquare$$

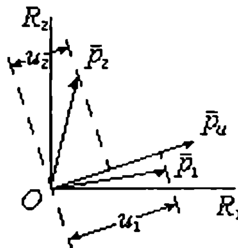


Fig. 6.2.

https://biblioteca-digitala.ro / https://unibuc.ro

Ideea de bază a euristicii pentru rezolvarea Problemei 2 constă în calculul profitabilității fiecărui proiect şi acceptarea celui mai profitabil, reluând procedura cât timp are loc (6.6). Soluțiile obținute în acest mod reprezintă, de obicei, aproximări foarte bune, fără a fi optimale.

Algoritmul pentru metoda primală a gradientului efectiv este următorul:

Algoritmul 6.1. (Metoda primală a gradientului)

1. [Inițializări] Se face $T_u=\varnothing$, $T_D=T$, $\overline{P}_u=(0,0,...,0)$, $Z=0$ şi $X_i=0$, pentru $i=1,2,...,m$.
2. [Proiecte candidat] Se determină mulțimea proiectelor candidat:
$$T_C = \{P_i \mid P_i \in T_D, \overline{P}_i \leq \overline{B} - \overline{P}_u\}.$$
3. [Terminare] Dacă $T_C = \varnothing$, atunci STOP (nu mai există proiecte candidat).
4. [Calcul gradienți] Se calculează gradienții efectivi ai proiectelor din T_C astfel: Dacă $\overline{P}_u = \overline{0}$, atunci $\overline{E}$ nu poate fi calculat şi se calculează $G_i = K_i \cdot n^{1/2} / \sum_{j=1}^{n} F_{ij}$, pentru $P_i \in T_C$. (Această variantă de calcul corespunde cazului $\overline{P}_u = (1, 1, ..., 1)$, adică situației în care toate resursele au aceeaşi penalizare. Această situație apare, de cele mai multe ori, numai la inițializarea procedurii.) Altfel se calculează:
$$\left|\overline{P}_u\right| = \left(\sum_{j=1}^{n} C_j^2\right)^{1/2}, \quad \overline{E} = \overline{P}_u / \left|\overline{P}_u\right|, \quad U_i = \overline{P}_i \cdot \overline{E} = \left(\overline{P}_i \cdot \overline{P}_u\right) / \left|\overline{P}_u\right| = \left(\sum_{j=1}^{n} F_{ij} C_j\right) / \left(\sum_{j=1}^{n} C_j^2\right)^{1/2},$$
$$G_i = K_i / U_i = K_i \left(\sum_{j=1}^{n} C_j^2\right)^{1/2} / \left(\sum_{j=1}^{n} F_{ij} C_j\right),$$ pentru $P_i \in T_C$; în cazul când $\sum_{j=1}^{n} F_{ij} C_j = 0$, se atribuie lui G_i o valoare foarte mare, deoarece un asemenea proiect nu are o penalizare şi deci este foarte profitabil.
5. [Alegere] Se alege un P_k cu cel mai mare gradient efectiv.
6. [Modificări] Se face $T_u = T_u + \{P_k\}$, $\overline{P}_u = \overline{P}_u + \overline{P}_k$, $Z = Z + K_k$, $T_D = T_D - \{P_k\}$ şi $X_k = 1$.
7. [Ciclare] Se trece la pasul 2. ■

Exemplul 6.3. Fie o mulțime de 8 proiecte cu datele din tabelul 6.1. care folosesc două resurse limitate. Resursele pot fi de orice tip ca de exemplu bani, numărul de ore de procesare, numărul specialiştilor, numărul maşinilor şi altele. Aplicând algoritmul prezentat, rezultă:

Pasul 1. $T = \{P_1, P_2, ..., P_8\}$, $T_u = \varnothing$, $T_D = T = \{P_1, P_2, ..., P_8\}$, $\overline{P}_u = \{0, 0\}$, $Z = 0$, $X_i = 0$, pentru $i = 1, 2, ..., 8$.

https://biblioteca-digitala.ro / https://unibuc.ro

P_i	H_{i1}	H_{i2}	K_i	F_{i1}	F_{i2}
P_1	6	2	100	0.250	0.067
P_2	2	8	400	0.083	0.267
P_3	6	5	600	0.250	[illegible]167
P_4	4	6	800	0.167	0.200
P_5	9	3	300	0.375	0.100
P_6	3	2	200	0.125	0.067
P_7	5	6	400	0.208	0.200
P_8	1	7	500	0.042	0.233
Total	36	49		1.500	1.300
Limita superioară	24	30		1.000	1.000

Tabelul 6.1.

Pasul 2. $T_C = \{P_i \mid P_i \in T_D, \overline{P_i} \le \overline{B} - \overline{P_u}\} = \{P_1, P_2, ..., P_8\}$.

Pasul 3. Deoarece $T_C \neq \varnothing$, se trece la pasul 4.

Pasul 4. Deoarece $\overline{P_u} = \{0, 0\}$, gradientul efectiv se calculează după prima variantă, obținându-se: $G_1 = K_1 \cdot n^{1/2} / \sum_{j=1}^{n} F_{1j} = 100 \cdot 2^{1/2} / (0.250 + 0.067) = 446$. În mod analog se obțin $G_2 = 1616$, $G_3 = 2036$, $G_4 = 3085$, $G_5 = 893$, $G_6 = 1475$, $G_7 = 1386$ și $G_8 = 2571$.

Pasul 5. Proiectul cu gradientul efectiv maxim este P_4, deoarece $G_4 = 3085$ este valoarea maximă calculată în pasul 4.

Pasul 6. Se efectuează atribuirile: $T_u = T_u + \{P_4\} = \varnothing + \{P_4\} = \{P_4\}$, $\overline{P_u} = \overline{P_u} + \overline{P_4} = (0, 0) + (0.167, 0.200) = (0.167, 0.200)$, $Z = Z + K_4 = 0 + K_4 = 800$, $T_D = T_D - \{P_4\} = \{P_1, P_2, P_3, P_5, P_6, P_7, P_8\}$, $X_4 = 1$.

Pasul 7. Se trece la pasul 2.

Pasul 2. $T_C = \{P_i \mid P_i \in T_D, \overline{P_i} \le \overline{B} - \overline{P_u}\} = \{P_1, P_2, P_3, P_5, P_6, P_7, P_8\}$.

Pasul 3. Deoarece $T_C \neq \varnothing$, se trece la pasul 4.

Pasul 4. Deoarece $\overline{P_u} \neq 0$, gradienții efectivi se calculează după varianta a doua:

$$G_1 = K_1 \left(\sum_{j=1}^{n} C_j^2 \right)^{1/2} / \left(\sum_{j=1}^{n} F_{1j} C_j \right) = 100/(0.167^2 + 0.200^2)^{1/2}/(0.250 \cdot 0.167 + 0.067 \cdot 0.200) = 473.$$

În mod analog se obțin: $G_2 = 1549$, $G_3 = 2083$, $G_5 = 947$, $G_6 = 1524$, $G_7 = 1394$ și $G_8 = 2429$.

Pasul 5. Proiectul cu gradientul efectiv maxim este P_8.

Pasul 6. Se efectuează atribuirile: $T_u = T_u + \{P_8\} = \{P_4, P_8\}$, $\overline{P_u} = \overline{P_u} + \overline{P_8} = (0.167, 0.200) + (0.042, 0.233) = (0.209, 0.433)$, $Z = Z + K_8 = 1300$, $T_D = T_D - \{P_8\} = \{P_1, P_2, P_3, P_5, P_6, P_7\}$, $X_8 = 1$.

https://biblioteca-digitala.ro / https://unibuc.ro

Pasul 7. Se trece la pasul 2.

După patru reluări ale procedurii, T_C devine vidă şi algoritmul se încheie. Soluția finală este: $Z = 2600$, $X_1 = X_2 = X_3 = X_4 = X_6 = X_8 = 1$, $X_5 = X_7 = 0$, $T_u = \{P_1, P_2, P_3, P_4, P_6, P_8\}$, $T_D = \{P_5, P_7\}$ şi $\overline{P}_4 = (0.917, 1.000)$. Tabelul 6.2. şi figura 6.3. ilustrează procesul acceptări proiectelor la fiecare iterație.

G_i	1	2	3	4	5	6
G_1	446	473	596	488	446	**501**
G_2	1616	1549	1447	1524	**1565**	
G_3	2036	2083	**2320**			
G_4	**3085**					
G_5	893	947	1187			
G_6	1475	1524	1750	**1552**		
G_7	1386	1394	1479	1402	1390	
G_8	2571	**2429**				
$\overline{P}_u$	(0.167,0.2)	(0.209,0.433)	(0.455,0.6)	(0.583,0.667)	(0.667,0.933)	(0.917,1.0)

Tabelul 6.2.

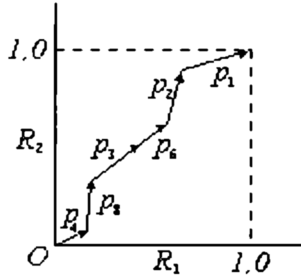


Fig. 6.3.

Soluția obținută coincide cu cea optimală, găsită prin enumerarea totală. ■

6.3. Programarea liniară a comenzilor cu variabile 0-1

Punerea problemei. Fiind date mai multe comenzi, fiecare dintre ele putând fi acceptată sau neaccepatată, se caută identificarea unei ordini de efectuare optimală a acestor comenzi, unele dintre ele putând fi efectuate parțial. ■

Pentru formalizarea problemei, se folosesc următoarele notații:

m - numărul de comenzi;

n - numărul de resurse;

https://biblioteca-digitala.ro / https://unibuc.ro

K_i - creşterea profitului asigurată de comanda i, $i = 1, 2, ..., m$;
L_j - capacitatea limită a resursei j, $j = 1, 2, ..., n$;
H_{ij} - cantitatea necesară din resursa j pentru satisfacerea comenzii i;
X_i - variabile ce urmează a fi determinate ($X_i = 0$ sau 1);
Z - funcţia obiectiv ce urmează a fi maximizată (minimizată).

Problema de rezolvat constă în identificarea unei ordini de efectuare a comenzilor care maximizează valoarea:

$$Z = K_1 X_1 + K_2 X_2 + ... + K_m X_m,$$

satisfăcând restricţiile:

$$X_i = 0 \text{ sau } 1,$$
$$\sum_i H_{ij} X_i \le L_j.$$

Acesta este un caz special al problemei 0-1 a rucsacului.

Exemplul 6.4. Fie cazul a două procesoare ce urmează să furnizeze produsele comandate de opt clienţi. Duratele efectuării comenzilor şi profiturile corespunzătoare sunt prezentate în tabelul 6.3., capacităţile limită ale celor două procesoare fiind 24, respectiv, 30 de ore de lucru.

Ordinea propusă	Procesorul I (ore)	Procesorul II (ore)	Profitul
p_1	6	2	100
p_2	2	8	400
p_3	3	2	200
p_4	4	6	800
p_5	9	3	300
p_6	6	5	600
p_7	5	6	400
p_8	1	7	500
Total	36	39	3300
Capacităţi limită	24	30	
Depăşirea capacităţii	12	9	

Tabelul 6.3.

Având în vedere faptul că există două procesoare, este convenabil să se folosească în calitate de notaţii vectori bidimensionali. Cu aceste notaţii, comenzilor p_1, p_2 etc. le corespund vectorii (6, 2), (2, 8) etc.

Notând cu $\overline{R}$ vectorul rezultat al tuturor comenzilor, cu $\overline{L}$ vectorul capacitate maximală şi cu $\overline{S}$ surplusul de ore pe cele două procesoare, rezultă următoarele ecuaţii vectoriale:

https://biblioteca-digitala.ro / https://unibuc.ro

$$\bar{R} = \bar{p}_1 + \bar{p}_2 + \ldots + \bar{p}_8,$$
$$\bar{S} = \bar{R} - \bar{L},$$
$$\bar{L} = (24, 30),$$

ilustrate pe figura 6.4. Zona dreptunghiulară definită de punctele 0, L_1, L, L_2 conţine puncte cărora le corespund extremităţi ale vectorilor $\bar{R}$ ce definesc comenzi efectuabile de către cele două procesoare.

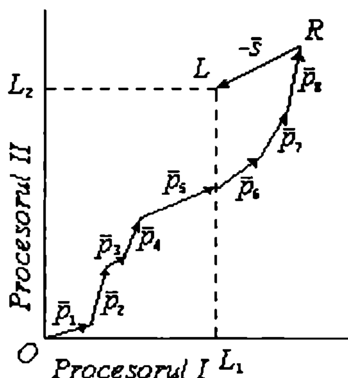


Fig. 6.4.

Dacă se elimină comanda p_8, punctul R se deplasează în extremitatea vectorului $\bar{p}_7$ şi are loc o pierdere egală cu 500. Contribuţia comenzii p_8 la procesul de apropiere de punctul L (dealungul vectorului $\bar{S}$) este egală cu lungimea proiecţiei vectorului $\bar{p}_8$ pe vectorul $\bar{S}$. Este preferabil să se elimine acele comenzi a căror profit este mic faţă de lungimea proiecţiei pe vectorul $\bar{S}$ a vectorului corespunzător comenzii considerate. Drept măsură a contribuţiei unei comenzi, numită în continuare *gradient efectiv*, se consideră raportul dintre profitul comenzii p_i şi proiecţia pe $\bar{S}$ a vectorului corespunzător $\bar{p}_i$.

Dacă se notează cu $\bar{u}$ versorul vectorului $\bar{S}$, atunci proiecţia u_i a vectorului $-\bar{p}_i$ pe $-\bar{u}$ este definită de produsul scalar:

$$u_i = \left(-\bar{p}_i\right)\left(-\bar{u}\right),$$

unde, în cazul exemplului studiat, $\bar{u}$ are expresia:

$$\bar{u} = \bar{S} / |\bar{S}| = \left(12 / \left(12^2 + 9^2\right)^{1/2}, 9 / \left(12^2 + 9^2\right)^{1/2}\right).$$

Pentru u_1 se obţine:

$$u_1 = \bar{p}_1 \cdot \bar{u} = 6 \cdot 12 / \left(12^2 + 9^2\right)^{\frac{1}{2}} + 2 \cdot 9 / \left(12^2 + 9^2\right)^{\frac{1}{2}} = 90 / \left(12^2 + 9^2\right)^{\frac{1}{2}}.$$

Gradientul efectiv G_i al vectorului $\bar{p}_i$ are expresia:

https://biblioteca-digitala.ro / https://unibuc.ro

$$G_i = K_i / u_i = K_i / \left(\bar{p}_i \cdot \bar{u}\right),\ i = 1, 2, ..., m.$$

Pentru exemplul studiat, se obțin următoarele valori ale gradienților efectivi:

$$G_1 = 1.1 \cdot S,\ G_2 = 4.2 \cdot S,\ G_3 = 3.7 \cdot S,\ G_4 = 7.8 \cdot S,$$
$$G_5 = 2.2 \cdot S,\ G_6 = 5.1 \cdot S,\ G_7 = 3.5 \cdot S,\ G_8 = 6.6 \cdot S.$$

Ordonând comenzile p_i crescător față de gradienții efectivi, se obține șirul:

$$p_1, p_5, p_7, p_3, p_2, p_6, p_8, p_4.$$

Identificarea unei soluții a problemei studiate constă în eliminarea succesivă a comenzilor până când depășirile capacităților celor două procesoare devin negative, urmată de adăugarea, dacă e posibil, a unor comenzi cu gradient efectiv mic astfel încât depășirile capacității procesoarelor să nu devină pozitive. Modul în care se obține o soluție pentru exemplul dat este ilustrat în tabelul 6.4. Rezultă că dacă se elimină comenzile p_5 și p_7 se obține un profit egal cu: 3300 - 300 - 400 = 2600.

	Procesorul 1	Procesorul 2
Surplusul inițial de ore	12	9
Eliminarea comenzii $p_1(6, 2)$	6	7
Eliminarea comenzii $p_5(9, 3)$	-3	4
Eliminarea comenzii $p_7(5, 6)$	-8	-2
Adăugarea comenzii $p_1(6, 2)$	-2	0

Tabelul 6.4.

Altă soluție cu același profit se obține eliminând comenzile p_1, p_3 și p_7 ∎

Vom prezenta în continuare unele îmbunătățiri și generalizări ale acestei proceduri pentru planificare.

Se observă mai întâi că:

$$G_i = K_i / \left(\bar{p}_i \cdot \bar{u}\right) = K_i / \left(\bar{p}_i \cdot \bar{S} / S\right) = K_i \cdot S / \left(\bar{p}_i \cdot \bar{S}\right),$$

unde S figurează ca factor comun în toți gradienții efectivi. Rezultă că este convenabilă utilizarea în calcule în locul lui G_i a valorii G_i' definită prin:

$$G_i' = K_i / \left(\bar{p}_i \cdot \bar{S}\right),\ i = 1, 2, ..., m.$$

Valoarea G_i' reprezintă *gradientul efectiv relativ* și este numită tot *gradient efectiv*.

Dacă punctul rezultat R este situat în modul ilustrat pe figura 6.5., calculul gradientului efectiv trebuie efectuat dealungul direcției S_1 și nu S_2, deoarece punctul reprezentativ al soluției optimale nu poate avea o ordonată mai mare decât cea a lui R.

Uneori, procesul de eliminare a unora dintre comenzi aduce succesiv punctul R într-o regiune (vezi figura 6.6.) în care restricțiile relative la procesorul I nu mai sunt aplicabile. Această situație are loc dacă punctul figurativ A al soluției optimale are poziția din figura 6.6.

https://biblioteca-digitala.ro / https://unibuc.ro

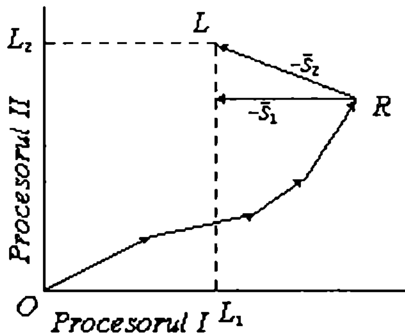


Fig. 6.5.

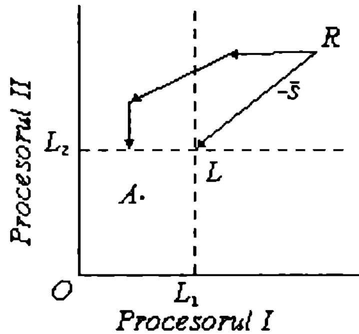


Fig. 6.6.

În asemenea cazuri se poate încerca reluarea eliminării, plecând de la punctul R, a altor comenzi astfel încât intrarea în zona dreptunghiulară definită de punctele O, L_1, L, L_2 să se efectueze pe un drum cât mai apropiat de L. Această strategie nu conduce însă întotdeauna la o îmbunătăţire a soluţiei. În afară de aceasta, atunci când punctul R este apropiat de L, direcţia vectorului $\overline{S}$ poate deveni instabilă, schimbându-se mult după fiecare eliminare a comenzilor. Pentru evitarea dificultăţilor menţionate, în cazul unui număr mare de procesoare şi comenzi, se recomandă următoarele strategii de recalculare a gradienţilor efectivi:

Metoda I: Când are loc eliminarea depăşirii capacităţii măcar a unui procesor.

Metoda II: Când are loc eliminarea a d sau sau mai multe depăşiri, unde d este un număr mic faţă de numărul total de procesoare.

Metoda III: Când a avut loc eliminarea unei comenzi.

Metoda IV: Când a avut loc eliminarea a c comenzi, unde c este un număr mic faţă de numărul total de comenzi.

https://biblioteca-digitala.ro / https://unibuc.ro

Rezultatele experimentale dovedesc o foarte bună eficiență a euristicii propuse, în multe cazuri fiind obținute soluții optimale sau aproape optimale.

6.4. Problema depozitării

Punerea problemei. Se consideră depozitarea unor produse în depozite cu capacitate nelimitată. Trebuie să se minimizeze $Z = \sum_{i,j} C_{ij}X_{ij} + \sum_i F_iY_i$ cu condițiile $\sum_i X_{ij} = 1$, pentru $j = 1, 2, ..., n$; $0 \leq X_{ij} \leq Y_i \leq 1$, $Y_i = 0$ sau 1, pentru $i = 1, 2, ..., m$ și $j = 1, 2, ..., n$, unde m este numărul depozitelor; n este numărul clienților; $C_{ij} = t_{ij} D_j$; t_{ij} este cost ce include costul utilizării depozitului i, costul operațiilor de depozitare și transportul de la depozit la clientul j; D_j este cererea consumatorului j; X_{ij} este fracțiunea din D_j furnizată de depozitul i; F_i este costul fix asociat depozitului i. ■

Termenii de depozit *deschis*, *închis* și *liber* au semnificațiile de depozit ce poate fi folosit, depozit ce nu poate fi folosit, respectiv depozit potențial, nedeclarat încă deschis sau închis.

Euristica propusă are la bază ideea identificării unui drum în arborele generat de metoda B&B (Branch and Bound) utilizând mai multe reguli de ramificare. Cea mai bună dintre soluțiile identificate este acceptată drept soluție a problemei.

Problema este mai întâi rezolvată ca o problemă de programare liniară, fără restricții privind integritatea valorilor Y_i. Fie Z_0 o soluție obținută astfel. Dacă toți Y_i sunt întregi, atunci problema este rezolvată. Altfel Z_0 devine marginea inferioară a problemei. În continuare se alege un Y_k neîntreg și se rezolvă două probleme auxiliare cu $Y_k = 0$, adică depozitul k este închis, respectiv $Y_k = 1$, adică depozitul k este deschis. În reprezentarea pe arbore aceasta corespunde la două arce divergente din nodul asociat soluției Z_0. Dacă Z_1 și Z_2 sunt cele două noi soluții (evident că $Z_1 \geq Z_0$ și $Z_2 \geq Z_0$) atunci $\overline{Z} = \min(Z_1, Z_2)$ este noua margine inferioară pentru problema studiată. Procedura continuă până ce marginea superioară curentă devine egală cu marginea inferioară curentă.

Pentru accelerarea procesului de identificare a soluției se folosesc trei simplificări descrise în continuare.

Prima simplificare determină o margine minimală pentru a decide un depozit ca fiind deschis. Dacă această margine este pozitivă, depozitul respectiv este declarat deschis. Cu K_0, K_1 și K_2 se notează mai jos mulțimile de indici ale depozitelor declarate închise, deschise, respectiv libere. Marginea menționată Δ_i se calculează cu formula următoare:

https://biblioteca-digitala.ro / https://unibuc.ro

$$\Delta_i = \sum_{j \in P_i} \left\{ \min_k \left[\max\left(C_{kj} - C_{ij}, 0 \right) \right] \right\} - F_i,$$

unde $i \in K_2$, $k \in K_1 \cup K_2$, $k \in N_j$, $k \neq i$, cu N_j se notează mulțimea depozitelor ce pot deservi clientul j, P_i este mulțimea clienților ce pot fi deserviți de depozitul i și numărul de elemente din P_i se notează în continuare cu n_i.

A doua simplificare constituie un mijloc de reducere a lui n_i în felul următor: dacă pentru $i \in K_2$, $j \in P_i$ are loc $\min(C_{kj} - C_{ij}) < 0$, unde $k \in K_i \cap N_j$, atunci n_i se reduce cu 1. Dacă inegalitatea are loc pentru toți $j \in P_i$, atunci $n_i = 0$ și $Y_i = 0$, pentru toate ramurile divergente din nodul corespunzător.

A treia simplificare determină o margine maximă a reducerii costului deschiderii unui depozit. Pentru $i \in K_2$, $j \in P_i$ această margine este calculată cu formula următoare:

$$\Omega_i = \sum_{j \in P_i} \left\{ \min_k \left[\max\left(C_{kj} - C_{ij}, 0 \right) \right] \right\} - F_i,$$

unde $k \in K_1$, $k \in N_j$. Dacă $\Omega_i < 0$, atunci $Y_i = 0$, adică depozitul i se consideră închis pentru toate ramurile divergente din modul examinat.

Cele opt reguli de ramificare sunt:

1. Regula Delta maxim	5. Regula Delta minim
2. Regula Omega maxim	6. Regula Omega minim
3. Regula Cererii maxime	7. Regula Cererii minime
4. Regula Y maxim	8. Regula Y minim

Regulile 1 ÷ 4 (respectiv 5 ÷ 8) se numesc de *tip 1* (respectiv de *tip 2*).

Regula 1 (respectiv 5) alege depozitul liber cu cel mai mare (cel mai mic) Δ_i din mulțimea de depozite libere din nodul în care $\Delta_i < 0$.

Regula 2 (respectiv 6) alege depozitul liber cu cel mai mare (cel mai mic) Ω_i din mulțimea de depozite libere din nodul în care $\Omega_i < 0$.

Regula 3 (respectiv 7) alege depozitul liber cu cel mai mare (cel mai mic) Y_i din mulțimea de depozite libere din nodul în care Y_i nu este întreg.

Regula 4 (respectiv 8) alege depozitul liber care poate satisface cererea totală maximă (cererea totală minimă) corespunzătoare nodului considerat.

Corespunzător celor opt reguli de ramificare sunt utilizate opt euristici. Fiecare euristică alege un depozit k liber (cu Y_k neîntreg), după care acest depozit este declarat atât deschis ($Y_k = 1$) cât și închis ($Y_k = 0$), celor două declarații corespunzându-le două noduri adiționale, ramurile corespunzătoare fiind numite ramură *deschisă*, respectiv *închisă*. Ideea de bază a euristicii poate fi formulată astfel: se trasează drumul preferat

https://biblioteca-digitala.ro / https://unibuc.ro

ce rezultă din aplicarea unei reguli de ramificare particulare. Această idee este ilustrată în figura 6.7.

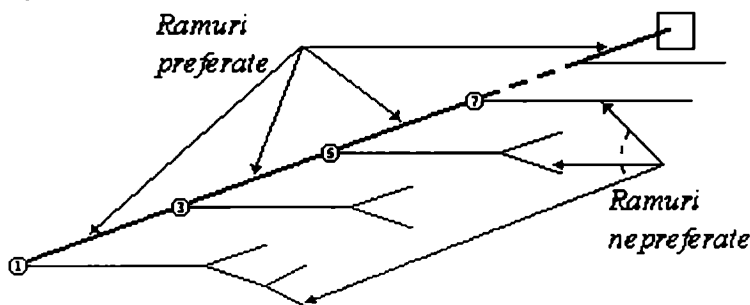


Fig. 6.7.

Euristica propusă are structura ce urmează.

Algoritmul 6.2. (***Depozitare***)

1. [Inițializare] Se inițializează mulțimile N_j, P_i, K_0, K_1, K_2 pentru primul nod al arborelui.
2. [Simplificări] Se parcurg ciclic simplificările pentru nodul inițial cu metodele de simplificare. Dacă $K_2 = \varnothing$, atunci se trece la pasul 3, altfel se trece la pasul 4.
3. [Soluția reală] Rezolvă problema de programare liniară *PL*. Dacă problema *PL* nu are soluție, atunci STOP. Dacă problema *PL* are o soluție terminală ($Y_k = 0$ sau $Y_k = 1$, pentru orice k), atunci aceasta este soluția optimală și STOP.
4. [Terminare] Dacă au fost aplicate toate regulile de ramificare (regulile 1÷8), atunci s-a obținut o soluție euristică a problemei și STOP.
5. [Alegere regulă] Se selectează o regulă *R* de ramificare încă nefolosită. Dacă nodul curent este cel inițial, se rezolvă problema *PL* pentru a găsi mulțimea *Y*.
6. [Alegere depozit] Se alege un depozit conform cu regula selectată. Dacă regula este de tip 1, atunci se consideră depozitul ales deschis; altfel (regulă de tip 2) se consideră depozitul ales închis.
7. [Simplificări] Se parcurg ciclic simplificările utilizând metodele de simplificare. Dacă s-a întâlnit o condiție nerealizabilă, atunci se trece la pasul 4. (O condiție nerealizabilă are loc dacă un depozit deschis este constrâns să fie închis fie datorită tipului regulii de ramificare fie în ciclul de ramificare).
8. [Determinare *Y*] Dacă a fost utilizată regula de ramificare *R*, atunci se identifică mulțimea *Y*.
9. [Ajustări] Dacă $K_2 = \varnothing$, se rezolvă problema *PL* și se schimbă soluția euristică dacă este necesar, adică dacă această soluție este mai bună decât soluția obținută anterior, apoi se trece la pasul 4. Altfel se trece la pasul 6. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Metodele de simplificare pot fi sistematizate ca în următorul algoritm.

Algoritmul 6.3. (***Simplificare***)

1. [Test rădăcină] Dacă nodul curent nu este rădăcina arborelui, atunci se trece la pasul 4.
2. [Calcule Δ_i] Se calculează Δ_i pentru depozitele libere. Se alege un depozit i cu $\Delta_i > 0$. Dacă nu există un astfel de depozit, atunci STOP.
3. [Poziționare] Se consideră depozitul i deschis, se reinițializează mulțimile K_1 și K_2 și se trece la pasul 5.
4. [Test ramură închisă] Dacă ramura este închisă, atunci se trece la pasul 6.
5. [Simplificările 2 și 3] Dacă $K_2 = \varnothing$, atunci STOP. Altfel, se reinițializează P_i, n_i, K_0, K_2 și dacă $K_2 = \varnothing$, atunci STOP. Altfel se calculează Ω_i pentru toate depozitele libere și se alege un depozit i pentru care $\Omega_i \leq 0$. Dacă nu există un astfel de depozit, atunci STOP. Altfel se consideră depozitul i închis și se reinițializează K_0 și K_2.
6. [Eliminări clienți] Dacă $K_2 = \varnothing$, atunci STOP. Altfel, dacă există clienți deserviți de depozite ce au fost închise care să fie mai bine deserviți de depozite deschise, atunci se elimină acei clienți din mulțimile P_i de depozite libere.
7. [Ciclare] Se trece la pasul 2. ■

Subrutina de simplificare este descrisă în [KHU72].

6.5. Euristică liniară de procesare pe mașini paralele independente

Punerea problemei. Fiecare dintre cele n lucrări trebuie procesată fără întrerupere pe unul din cele m procesoare paralele independente, obiectivul fiind minimizarea timpului maxim de încheiere. ■

Este evident că dacă o procesare dată este modificată pe calea reordonării lucrărilor pe oricare dintre mașini, timpul maxim de încheiere nu se modifică. Rezultă că problema procesării se reduce la asignarea lucrărilor pe mașini.

O formulare a problemei utilizează variabile x_{ij} ce iau valorile 0, 1, pentru $i = 1, 2, ..., n; j = 1, 2, ..., m$, unde:

$$x_{ij} = \begin{cases} 1 & \text{dacă lucrarea } i \text{ este atribuită mașinii } j, \\ 0 & \text{altfel,} \end{cases}$$

și variabila C_{max} ce reprezintă timpul maxim de încheiere. Variabilele x_{ij} se numesc *variabile de asignare*. Problema poate fi formulată astfel:

https://biblioteca-digitala.ro / https://unibuc.ro

să se minimizeze C_{max},

cu condiţiile:

$$\sum_{i=1}^{n} t_{ij} x_{ij} \le C_{max}, \text{ pentru } j = 1, 2, ..., m, \quad (6.7)$$

$$\sum_{j=1}^{m} x_{ij} = 1, \text{ pentru } i = 1, 2, ..., n, \quad (6.8)$$

$$x_{ij} \in \{0, 1\}, \text{ pentru } i = 1, 2, ..., n; j = 1, 2, ..., m, \quad (6.9)$$

unde t_{ij} reprezintă timpul de procesare a lucrării i pe procesorul j.

Din condiţiile (6.7) rezultă că C_{max} este cel puţin egală cu timpul total de procesare pe orice maşină şi condiţiile (6.8) şi (6.9) asigură procesarea fiecărei lucrări pe o singură maşină. În continuare, se notează cu C^*_{max} valoarea unei soluţii optimale.

Fie *programarea liniară relaxată* în care condiţiile (6.9) sunt eliminate şi înlocuite cu:

$$x_{ij} \ge 0, x_{ij} \le 1, \text{ pentru } i = 1, 2, ..., n; j = 1, 2, ..., m,$$

unde restricţia $x_{ij} \le 1$ rezultă din (6.8). Efectul relaxării constă în posibilitatea procesării unei lucrări simultan pe mai multe maşini. O *programare parţială* poate fi obţinută din soluţia programării liniare relaxate pe calea asignării lucrării i maşinii j dacă $x_{ij} = 1$, pentru $i = 1, 2, ..., n$ şi $j = 1, 2, ..., m$. Lucrările ce nu figurează ı această programare parţială sunt numite *lucrări fracţionate*, datorită valorii neîntregi a variabilelor de asignare. În caz favorabil, soluţia programării liniare poate coincide cu cea a problemei originale, atunci când variabilele x_{ij} iau valori întregi. În continuare se deduce o margine superioară a numărului lucrărilor fracţionate.

Problema programării liniare conţine $m + n$ restricţii în plus faţă de condiţiile privind nenegativitatea. Rezultă că există o soluţie optimală având $m + n$ variabile de bază ce pot lua valori pozitive în timp ce celelalte variabile, care nu sunt de bază, iau valoarea zero. Totuşi, dacă se aplică un algoritm de programare liniară pentru obţinerea unei soluţii optimale în care mai mult de $m + n$ variabile iau valori pozitive, se poate folosi o procedură în timp polinomial [HAD62] pentru a transforma soluţia obţinută într-o soluţie optimală de bază. Într-o soluţie optimală de bază $C_{max} > 0$, deci C_{max} este o variabilă de bază. Rezultă că cel mult $m + n - 1$ variabile de asignare sunt de bază. Deoarece nici o pereche de condiţii (6.8) nu conţine o aceeaşi variabilă, este posibilă selectarea unei mulţimi B de n variabile de asignare de bază, astfel încât fiecare restricţie (6.8) să conţină exact o variabilă de bază din această mulţime.

Numărul de variabile de asignare de bază este $m - 1$, mulţimea lor fiind notată în continuare cu B'. Dacă $n \ge m - 1$, cel mult $m - 1$ restricţii din (6.8) conţin o variabilă din B' în plus faţă de o variabilă de bază din B, în timp ce $n - m + 1$ dintre

https://biblioteca-digitala.ro / https://unibuc.ro

ele conţin o variabilă de bază din B şi nici una din B'. În restricţiile ce conţin o singură variabilă de bază, această variabilă are valoarea unu deoarece celelalte variabile, care nu sunt de bază, iau valoarea zero. Rezultă că, pentru $n \geq m - 1$, cel puţin $n - m + 1$ variabile iau valoarea unu şi în consecinţă cel puţin $n - m + 1$ lucrări figurează în programarea parţială şi $m - 1$ lucrări sunt neprogramate.

Ideea unui algoritm euristic este următoarea. Mai întâi se rezolvă problema programării liniare relaxate şi, dacă este necesar, soluţia obţinută este transformată într-o soluţie optimală de bază. Programarea parţială este definită de variabilele de asignare ce iau valoarea unu. Pentru $n \leq m - 1$ soluţia parţială poate fi vidă. Programarea completă se obţine adăugând lucrările fracţionate la programarea parţială. Sunt considerate toate asignările posibile ale lucrărilor fracţionate şi se alege asignarea ce asigură cel mai mic timp maximal de procesare. Această euristică, ce constă din programare liniară şi apoi enumerare, este notată în continuare cu *PLE*.

Pentru deducerea complexităţii euristicii propuse se poate observa, mai întâi, că problema programării liniare cu $m \bullet n + 1$ variabile şi $m + n$ restricţii, poate fi rezolvată polinomial cu algoritmul lui Khachian [KHA79]. Orice transformare a soluţiei într-una de bază necesită de asemenea un timp polinomial. Deoarece există cel mult $m - 1$ lucrări fracţionate ce urmează să fie asignate maşinilor, pe calea enumerării complete, rezultă că sunt generate cel mult m^{m-1} programări ce unt apoi comparate în etapa a doua a algoritmului. În consecinţă, pentru un m fixat, complexitatea euristicii este polinomială şi pentru m arbitrar complexitatea este exponenţială.

Teorema 6.1. Pentru $m \geq 3$, $C_{\max}^{PLE} / C_{\max}^{*} \leq 2$ şi această margine este cea mai bună posibil.

Demonstraţie. Fie $C_{\max}^{PL}$ valoarea soluţiei optimale a programării liniare relaxate şi $C_{\max}^{E}$ timpul maxim de elaborare a programării lucrărilor fracţionate, considerate separat. Deoarece programarea liniară relaxată furnizează o margine inferioară, rezultă că:

$$C_{\max}^{*} \geq C_{\max}^{PL}. \tag{6.10}$$

De asemenea şi programarea optimală a lucrărilor fracţionate furnizează o margine inferioară a timpului maxim de încheiere a programării. Prin urmare:

$$C_{\max}^{*} \geq C_{\max}^{E}. \tag{6.11}$$

O programare posibilă ce poate fi generată de *PLE* constă în adăugarea la programarea optimală a lucrărilor fracţionate, considerată independent, a programării parţiale obţinută drept soluţie a programării liniare relaxate. Rezultă atunci că:

https://biblioteca-digitala.ro / https://unibuc.ro

$$C_{\max}^{PLE} \leq C_{\max}^{PL} + C_{\max}^{E}. \tag{6.12}$$

Din (6.10), (6.11) şi (6.12) rezultă că $C_{\max}^{PLE} / C_{\max}^{*} \leq 2$.

Faptul că marginea este cea mai bună posibil rezultă din exemplul 6.5. ■

Exemplul 6.5. Fie un exemplu în care cele m maşini sunt uniforme. Numărul de lucrări este m şi $t_{ij} = t_i / q_j$, pentru $i, j = 1, 2, ..., m$, unde $q_1 = m - 1$, $q_j = 1$, pentru $j = 2, 3, ..., m$, $t_1 = (m - 1)\, t$ şi $t_i = t$, pentru $i = 2, 3, ..., m$, pentru orice [illegible] pozitiv. O programare optimală se obţine dacă lucrarea i este asignată maşinii i, pentru $i = 1, 2, ..., m$, în acest caz rezultând $C_{\max}^{*} = t$. O soluţie, nu unica, a programării liniare relaxate este $x_{11} = 0$, $x_{i1} = 1$, $i = 2, ..., m$, $x_{1j} = 1 / (m - 1)$, pentru $j = 2, 3, ..., m$, şi $x_{ij} = 0$, pentru $i, j = 2, 3, ..., m$. Pentru cazul particular $m = 3$, cele două matrici definite de t_{ij} şi x_{ij} sunt:

$$\underset{t_{ij}}{i \overset{j}{\begin{bmatrix} t & t/2 & t/2 \\ 2t & t & t \\ 2t & t & t \end{bmatrix}}} \qquad \underset{x_{ij}}{i \overset{j}{\begin{bmatrix} 0 & 1 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 0 & 0 \end{bmatrix}}}.$$

Programarea parţială asignează lucrările 2, 3, ..., m maşinii 1 şi lucrarea 1 este lucrarea fracţionată. Enumerarea arată că lucrarea 1 este programată pe maşina 1 de unde rezultă $C_{\max}^{PLE} = 2t$ şi în consecinţă $C_{\max}^{PLE} / C_{\max}^{*} = 2$. ■

https://biblioteca-digitala.ro / https://unibuc.ro

7 ALTE APLICAȚII

7.1. Euristici de ajustare a costurilor în cazul incertitudinii cererilor

Punerea problemei. Pentru problema alegerii dinamice a valorii loturilor se propune o euristică bazată pe ajustarea stabilirii costurilor în cazul când timpul la care se efectuează cererile este incert. ■

Algoritmul lui Zangwill [ZAN69] rezolvă problema alegerii dinamice a valorii loturilor ținând seama de rezerve sau lipsuri. În cadrul extensiei acestei probleme, prezentate în continuare, se consideră că producția este planificată pentru *n* perioade în vederea satisfacerii comenzilor $L_1, L_2, \ldots, L_m$, fiecare dintre ele fiind efectuată într-o singură perioadă, definită de o distribuție de probabilități pe cele *n* perioade. Costul producerii cantității X_t în perioada *t* este dat de $a_tX_t+b_t\delta(X_t)$, unde $\delta(x)=0$ dacă $x=0$ și $\delta(x)=1$ dacă $x\neq0$. Constantele a_t și b_t sunt cunoscute pentru fiecare perioadă. Cantitatea D_t comandată în fiecare perioadă este o variabilă aleatoare ce depinde de distribuția comună a perioadelor în care apare comanda L_k. Inventarul inițial I_0 este cunoscut și inventarul de la sfârșitul fiecărei perioade este dat de $I_t=I_{t-1}+X_t-D_t$. Valoarea inventarului pentru fiecare perioadă are expresia $h_tI_t^+ + s_tI_t^-$, unde $I_t^+=\max(I_t,0)$ și $I_t^- = I_t^+ - I_t$. Costurile depozitării h_t și pierderile provocate de lipsa de materiale s_t sunt cunoscute pentru fiecare perioadă. Deoarece deciderea producției pentru fiecare perioadă este luată conform cererilor din toate perioadele anterioare, X_t poate depinde de toate cererile din perioadele 1 până la t-1.

Sunt propuse pentru rezolvarea acestei probleme trei euristici bazate pe *algoritmul Zangwill*. Pentru fiecare dintre ele, problema dată este modificată într-una fără incertitudini, în cadrul căreia fiecare cerere L_k apare într-o perioadă specificată, după care se aplică algoritmul Zangwill, care este optimal.

Soluția deterministă ignoră efectiv incertitudinea pe calea atribuirii fiecărei cereri L_k cu o perioadă scadentă, mai precis, cererea L_k este programată în prima

https://biblioteca-digitala.ro / https://unibuc.ro

perioadă t, pentru care probabilitatea ca această cerere să apară în timpul sau înaintea perioadei scadente este cel puţin 1/2. *Euristica deterministă* este o procedură cu următorii doi paşi:

1. Se specifică faptul că fiecare cerere apare în perioada scadentă.
2. Se aplică *algoritmul Zangwill* pentru obţinerea soluţiei deterministe, adică un plan de producţie fix. ■

Euristica timpului de asignare a priorităţii cuprinde următorii trei paşi·

1. Se caută perioada scadentă pentru fiecare cerere, ca şi în cazul euristicii deterministe.
2. Se verifică faptul că fiecare cerere apare cu s perioade înaintea perioadei scadente sau în prima perioadă mai mare (mai lungă).
3. Se aplică *algoritmul Zangwill* în vederea obţinerii unei soluţii cu timp de asignare a priorităţii. ■

Euristica ajustării costului cuprinde următorii trei paşi:

1. Se specifică faptul că fiecare cerere apare în perioada scadentă, definită ca şi în cazul euristicii deterministe.
2. Se defineşte un nou cost de furnizare b_t^* pentru fiecare perioadă, acest cost fiind egal cu costul de furnizare original plus costul aşteptat al lipsei de materiale provocate de cererea scadentă (cererile scadente) într-o perioadă t anterioară (definiţia lui b_t^* este dată mai jos).
3. Se aplică *algoritmul Zangwill* pentru obţinerea soluţiei euristicii de ajustare a costului. ■

În vederea definirii costului ajustat b_t^*, fie variabila aleatoare H_{kt} ce ia valoarea 1 dacă cererea L_k apare în perioada t şi valoarea 0 în caz contrar. Costul ajustat are expresia:

$$b_t^* = b_t + \sum_i \left(\sum_{j=1}^{t-1} s_j \cdot E\left[\sum_{k=1}^{j} H_{ik} \right] \right),$$

unde i parcurge toate comenzile programate în peioada t.

Euristicile descrise au fost testate pe mai multe exemple, unul dintre ele fiind descris în continuare.

Exemplul 7.1. Considerăm un număr de 12 perioade, în fiecare dintre ele fiind programată câte o cerere, conform tabelului 7.1.

Perioada	1	2	3	4	5	6	7	8	9	10	11	12
Cererea	31	68	32	81	40	35	32	70	91	30	30	34

Tabelul 7.1.

https://biblioteca-digitala.ro / https://unibuc.ro

Fiecare cerere apare cu următoarele probabilități: 0.2 cu două periode mai devreme, 0.2 cu o perioadă mai devreme, 0.3 în perioada programată, 0.3 cu o perioadă mai târziu. Tabelul 7.2. cuprinde costurile, care sunt propuse staționare.

Costul de producție (a_t):	0
Costul de furnizare (b_t):	100
Costul pierderilor (s_t):	5
Costul depozitării (h_t)	1

Tabelul 7.2.

Tabelul 7.3. cuprinde soluțiile pentru exemplul studiat. Tabelul 7.4. cuprinde rezultatele furnizate de cele trei euristici și un algoritm optimal [BUR84]. ■

	Perioada											
	1	2	3	4	5	6	7	8	9	10	11	12
Soluția deterministă	31	100	0	121	0	67	0	70	121	0	64	0
Asigurarea priorității	131	0	121	0	67	0	70	121	0	64	0	0
Ajustarea costului	99	0	188	0	0	0	193	0	0	94	0	0

Tabelul 7.3.

Metoda	Costul așteptat
Deterministă	1799
Asigurarea priorității	1342
Ajustarea costului	1298
Optimală	1237

Tabelul 7.4.

7.2. Euristici pentru problema localizării deservirii

Punerea problemei. Fiind dată o mulțime localizată de consumatori, se cere localizarea punctelor de deservire a lor astfel încât să se minimizeze costurile. ■

Forma generală a problemei localizării deservirii are enunțul:

$$\min_{S \subset J} Z(S), \text{ unde } Z(S) = \sum_{i \in I} \min_{j \in S} d_{ij} + \sum_{j \in S} f_j,$$

unde $I = \{1, 2, ..., n\}$ este mulțimea consumatorilor, $J = \{1, 2, ..., m\}$ este mulțimea facilităților, d_{ij} reprezintă costul deservirii consumatorului i de facilitatea j.

Orice soluție S a problemei localizării deservirii induce o partiție a mulțimii I după cum urmează:

$$i \in P_s \text{ dacă } d_{is} = \min_{j \in S} d_{ij},\ I = \bigcup_{s \in S} P_s.$$

Reciproc, orice partiție $\{P_s\}_{s \in S}$ a mulțimii I induce o soluție a problemei localizării deservirii și anume, pentru fiecare mulțime P_s se rezolvă problema

https://biblioteca-digitala.ro / https://unibuc.ro

$$C_s = \min_{j^* \in J}\left(\sum_{i \in P_s} d_{ij^*} + f_{j^*}\right) \quad (7.1)$$

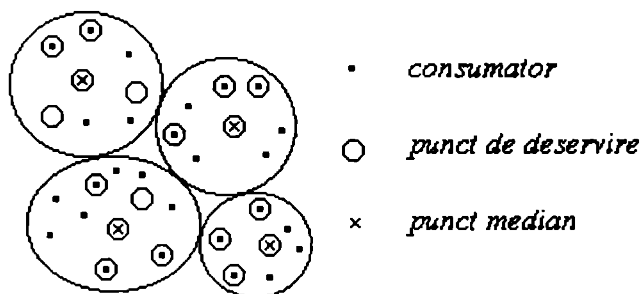


Fig. 7.1.

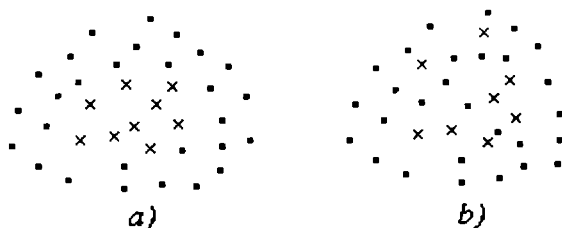
Rezolvarea optimală a problemei localizării deservirii este echivalentă cu rezolvarea optimală a problemei partiţionării cu ajutorul mulţimilor $P = \{P_j\}_{j=1}^{N}$, adică mulţimea putere a lui I, unde $N = 2^n - 1$. Fiecărei mulţimi P_j i se asociază o pondere C_j definită de (7.1). Problema studiată constă în identificarea unei acoperiri a mulţimii I de cost minim. O acoperire de cost minim este de fapt o partiţie deoarece $P_1 \subset P_2 \Rightarrow C_1 \leq C_2$. Figura 7.1. ilustrează medianele induse de partiţie.

În termeni de *acoperire cu mulţimi* problema localizării deservirii poate fi formulată astfel:

$$\min \sum_{j=1}^{N} C_j x_j \text{, cu condiţia } \sum_{j=1}^{N} a_{ij} x_j \geq 1,\ i \in I, \text{ unde } x_j \in \{0, 1\}, j \in P.$$

Definiţia 7.1. O mulţime $S \subset I$ este *circular convexă* dacă $\exists j' \in J$ astfel încât pentru toate perechile s, t cu $s \in S$, $t \in I$, $d_{j's} > d_{j't} \Rightarrow t \in S$. ■

Pentru cazul în care costurile sunt egale cu distanţele euclidiene în plan, în figura 7.2. sunt reprezentate două mulţimi, prima fiind circular convexă şi a doua nu.



• punct $\in I$, × punct $\in S$.

Fig.7.2.

În cazul când mulţimea J este discretă, o clasă restrictivă de mulţimi circular convexe se generează astfel: pentru fiecare $j \in J$ se identifică şirul $d_{ji_1} \leq d_{ji_2} \leq \ldots \leq d_{ji_n}$,

https://biblioteca-digitala.ro / https://unibuc.ro

unde $i_k \in I$; în cazul egalităţii a două elemente ordinea este arbitrară; pentru $j \in J$ se generează secvenţial mulţimile circular convexe pentru care j este *centru*: $\{i_1\}$, $\{i_1, i_2\}$, ..., $\{i_1, ..., i_{n-1}\}$, $\{i_1, ..., i_n\}$.

Rezultă că numărul de mulţimi convexe centrate în j este n.

Pentru rezolvarea problemei acoperirii cu mulţimi se foloseşte, în calitate de procedură, *algoritmul Chvatal* [CHV79] prezentat în continuare.

Algoritmul 7.1. (***Chvatal***)

1. [Iniţializare] Se face $ACOPERIRE = \emptyset$.
2. [Terminare] Dacă $P_j = \emptyset$ pentru orice $j \in \{1, 2, ..., N\}$, atunci STOP.
3. [Alegere] Se determină k astfel încât: $\frac{|P_k|}{C_k} = \max\limits_{j \in \{1, ..., N\}} \frac{|P_j|}{C_j}$.
4. [Ajustare] Se face $ACOPERIRE = ACOPERIRE \cup \{k\}$, $P_j = P_j - P_k$ pentru orice j.
5. [Ciclare] Se trece la pasul 2. ■

Paşii 2 - 5 necesită un număr de operaţii proporţional cu N. Aceşti paşi se repetă până când sunt acoperite toate cele n puncte ale mulţimii I. Deoarece la fiecare iteraţie mulţimea selectată conţine măcar un element, rezultă că numărul maxim de iteraţii este n, deci complexitatea temporală a algoritmului este $O(n \cdot N)$.

Ţinând seama că $|J| = m$ şi că numărul de mulţimi circular convexe centrate în $j \in J$ este $n = |I|$, rezultă că numărul total de mulţimi circular convexe este $m \cdot n$.

Un prim algoritm pentru rezolvarea problemei acoperirii cu mulţimi este prezentat în continuare.

Algoritmul 7.2. (***Acoperire***)

1. [Iniţializare] Se identifică toate mulţimile circular convexe $P_j \subset I$ şi se face $I' = I$ şi $ACOPERIRE = \emptyset$.
2. [Terminare] Dacă $I' = \emptyset$ sau $P_j = \emptyset$, pentru orice $j \in \{1, 2, ..., N\}$, atunci STOP.
3. [Alegere] Se determină k astfel încât: $\frac{|P_k|}{C(P_k)} = \max\limits_{P_j \neq \emptyset} \frac{|P_j|}{C(P_j)}$.
4. [Ajustare] Se face $ACOPERIRE = ACOPERIRE \cup \{k\}$, $P_j = P_j - P_k$ şi $C(P_j) = C(P_j - P_k)$, pentru orice j, $I' = I' - P_k$.
5. [Ciclare] Se trece la pasul 2. ■

Complexitatea algoritmului este $O(n^2 \cdot m)$.

O altă formă de implementare a problemei acoperirii cu mulţimi se deosebeşte de cea precedentă prin aceea că la fiecare iteraţie sunt generate toate mulţimile circular convexe.

https://biblioteca-digitala.ro / https://unibuc.ro

În cel mai nefavorabil caz, pentru Algoritmul 7.3., timpul de lucru are acelaşi ordin. Memoria necesară pentru Algoritmul 7.3. este matricea punctelor ordonate ale lui I şi matricea distanţelor, adică $O(m \cdot n)$. Pentru Algoritmul 7.2. este necesară memorarea matricei distanţelor şi a tuturor mulţimilor circular convexe. Deoarece fiecare mulţime circular convexă poate conţine până la n puncte, rezultă că memoria necesară este $O(n^2 m)$. Rezultă că Algoritmul 7.3. este mai eficient.

Algoritmul 7.3. (*Acoperire eficientă*)

1. [Iniţializare] Pentru orice $j \in J$, se sortează elementele lui I faţă de distanţa crescătoare la j şi se face $I' = I$ şi $ACOPERIRE = \emptyset$.
2. [Terminare] Dacă $I' = \emptyset$, atunci STOP.
3. [Evaluare] Pentru orice $j \in J$, se evaluează toate submulţimile circular convexe ale lui I' în conformitate cu ordinea din pasul 1. Pentru fiecare mulţime se calculează $|P|/C(P)$. Se memorează numai P_k şi $C(P_k)$ care maximizează valoarea de mai sus.
4. [Ajustare] Se face $ACOPERIRE = ACOPERIRE \cup \{k\}$ şi $I' = I' - P_k$.
5. [Ciclare] Se trece la pasul 2. ■

Complexitatea algoritmului este cel mult $O(n^2 \cdot m)$.

7.3. Problema reaprovizionării comune

Punerea problemei. Se cere identificarea unei strategii eficiente de reaprovizionare a unei firme furnizoare pentru satisfacerea tuturor cererilor mai multor clienţi. ■

Pentru problema reaprovizionării comune se poate arăta (vezi [VEI69]) că o strategie optimală constă în comanda fiecărui articol într-un interval de timp t, dacă inventarul lui la sfârşitul perioadei $(t - 1)$ este nul. Se presupune că intervalele de timp succesive au o durată egală cu 1. Fie un articol i pentru care se efectuează două comenzi succesive în perioadele s şi t, unde $1 \le s < t \le T + 1$, T fiind orizontul temporal al reaprovizionării. Rezultă că reaprovizionarea trebuie să acopere cererile din perioadele s până la $(t - 1)$. Fie în continuare următoarele definiţii:

C_{ist} - costul articolului i solicitat în perioadele s până la $(t - 1)$, dacă acest articol este comandat numai în perioadele s şi t = $K_i + H_i \sum_{\tau=s+1}^{t-1} (\tau - s) d_{i\tau}$, unde K_i este costul comenzii articolului i, H_i este costul inventarierii pe unitatea de timp a articolului i şi $d_{i\tau}$ este cererea articolului i în perioada de timp τ,

$$X_{ist} = \begin{cases} 1 & \text{dacă articolul } i \text{ este comandat numai în perioadele } s \text{ şi } t, \\ 0 & \text{altfel,} \end{cases}$$

$$X_t = \begin{cases} 1 & \text{dacă orice articol este comandat în perioada } t, \\ 0 & \text{altfel.} \end{cases}$$

https://biblioteca-digitala.ro / https://unibuc.ro

Fie de asemenea notațiile:
K_0 - costul comun de efectuare a unei comenzi,
N - numărul total de articole.

Cu aceste definiții și notații, problema reaprovizionării comune poate fi formulată în termeni de programare liniară astfel:

$$\text{Minimizează } \sum_{i=1}^{N}\sum_{s=1}^{T}\sum_{t=s+1}^{T+1} C_{ist}X_{ist} + \sum_{t=1}^{T} K_0 X_t,$$

cu condițiile:

$$\sum_{t>s} X_{ist} - \sum_{k<s} X_{iks} = 0, \text{ pentru } i = 1, 2, ..., N, s = 2, 3, ..., T,$$

$$\sum_{s>1} X_{i1s} = 1, \text{ pentru } i = 1, 2, ..., N,$$

$$X_t - \sum_{t>s} X_{ist} \geq 0, \text{ pentru } i = 1, 2, ..., N, s = 1, 2, ..., T,$$

$$X_{ist}, X_t = 0 \text{ sau } 1.$$

O metodă eficientă de obținere a unei soluții optimale a problemei enunțate mai sus este prezentată în continuare.

Ideea de bază a euristicii constă în echilibrarea costului individual de inventariere a unui articol cu costul necesar comenzii acestui articol. Mai precis, ori de câte ori costul total de inventariere a unui articol, începând cu ultima sa comandă, depășește costul individual al comenzii acestui articol, el devine candidat pentru efectuarea unei noi comenzi. O comandă comună este efectuată atunci când costul total de inventariere a tuturor articolelor candidat depășește costul total al comenzii lor plus costul comun al comenzii.

Fie t_i ultima perioadă în care a fost comandat articolul i și T_{it} costul total de inventariere al acestui articol în perioadele t_i, t_{i+1}, ..., t, presupunând că nici o comandă nu a fost efectuată în perioada t. Atunci:

$$T_{it} = H_i \sum_{j=1}^{t-t_i} j d_{i,t_i+j}.$$

În fiecare perioadă t în cadrul euristicii se calculează T_{it} pentru fiecare articol. Dacă T_{it} depășește pe K_i, atunci articolul i devine candidat al comenzii lui. Totuși, comanda nu este efectuată decât dacă acești candidați pot acoperi costul comenzii comune. Rezultă că în perioada t se aplică următoarea regulă.

Regula 1. Se comandă toate articolele candidat, adică cele pentru care $T_{it} \geq K_i$, în perioada t dacă $\sum_{i=1}^{N}\left(T_{it} - K_i\right)^+ \geq K_0$. În continuare are loc trecerea la următoarea perioadă de timp.

Fie t perioada în care a fost efectuată o comandă comună în conformitate cu regula 1. Este posibil ca un articol, care nu este un candidat, să impună o nouă

https://biblioteca-digitala.ro / https://unibuc.ro

comandă, foarte curând după perioada t. În acest caz, poate fi mai avantajos să se comande acest articol în perioada t în vederea evitării costului suplimentar în cazul unei comenzi separate. Pentru ilustrarea acestei idei considerăm următorul exemplu.

Exemplul 7.2. Fie două articole cu $H_1 = H_2 = 1$ şi comenzile:

$d_{1,t} = 0$, pentru $t = 1, 2, ..., (m - 1)$,

$d_{1,m} = (K_0 + K_1) / (m - 1)$,

$d_{2,t} = 0$, pentru $t = 1, 2, ..., m$,

$d_{2,m+1} = (K_0 + K_2) / m$.

În conformitate cu regula 1 articolul 1 va fi comandat în perioada m şi articolul 2 în perioada $m + 1$. Dacă însă cele două articole sunt comandate în perioada m, atunci pentru articolul 2 rezultă un extracost de inventariere egal cu $(K_0 + K_2) / m$ însoţit de o reducere în costul comun K_0. Dacă m este destul de mare, reducerea va fi mai mare decât extracostul de inventariere. Pe lângă aceasta, dacă numărul N de articole este mare, atunci într-un exemplu similar, euristica va comanda fiecare articol într-o perioadă distinctă, astfel încât costul comun va trebui achitat de N ori. Dacă toate articolele sunt comandate în aceeaşi perioadă cu primul, atunci se asigură o economie egală cu $(N - 1) K_0$ în costul comenzilor cu preţul unei creşteri mici a costului inventarierii. Rezultă că, în forma descrisă, euristica poate avea o performanţă oricât de rea dacă numărul de articole este mare. ■

Fie t_0 perioada în care a fost efectuată o comandă comună şi i un articol ce nu a fost candidat să fie comandat în această perioadă deoarece $T_{it_0} < K_i$. Pentru acest articol, ultima perioadă t_i în care a fost comandat satisface condiţia $t_i < t_0$. Fie S_{it} mărimea definită prin:

$$S_{it} = H_i (t_0 - t_i) \sum_{j=t_0}^{t} d_{ij} ,$$

care reprezintă economia în inventarierea articolului i dacă o comandă a acestui articol este plasată în intervalul t_0. Evident că efectuarea acestei comenzi este indicată dacă economia în inventariere este mai mare decât costul comenzii K_i a articolului i. De aici rezultă:

Regula 2. Dacă la momentul t ultima comandă comună a avut loc în intervalul t_0 şi articolul i n-a fost comandat în acest interval şi $S_{it} \geq K_i$, atunci şi articolul i este comandat în intervalul t_0.

Reflectarea acestei reguli constă în recalcularea lui T_{it}. Adică T_{it} se calculează începând cu t_0, în loc de t_i.

Exemplul 7.3. Pentru a vedea de ce schimbarea ordinei, descrisă mai sus, previne comportarea neeficientă a euristicii, fie din nou datele din exemplul 7.2.

https://biblioteca-digitala.ro / https://unibuc.ro

La momentul m+1, pentru articolul 2 cantitatea $S_{2,m+1}$ este $(m-1)(K_0+K_2)/m$. Rezultă că $S_{2,m+1} \le K_2$ dacă $K_0 \le K_2/(m-1)$ şi, în consecinţă, euristica nu schimbă ordinea comenzii articolului 2 dacă fie K_0, fie m au valori mici. Dacă însă K_0 sau m au valori mari, atunci euristica schimbă ordinea articolului 2 şi asigură o strategie optimală. ■

În continuare se prezintă euristica cu observaţia că, pentru simplificare, se introduce intervalul fictiv $T + 1$ de la sfârşitul orizontului temporal, în cadrul căruia comanda pentru fiecare articol este nulă.

Algoritmul 7.4. (Euristica acoperirii costului)

1. [Iniţializări] Se face $t = 1$, $T_{it} = S_{it} = 0$, pentru orice i, şi $t_0 = t_i = 1$, pentru orice i. Se comandă în prima perioadă fiecare articol.
2. [Terminare] Se face $t = t + 1$ şi dacă $t > T$, atunci STOP.
3. [Determină candidaţi] Pentru fiecare articol i se calculează S_{it} şi dacă $S_{it} \ge K_i$ (Regula 2: schimbă comanda articolului i), atunci se plasează comanda articolului i în intervalul t_0 şi se efectuează atribuirea $t_i = t_0$. Pentru fiecare articol i se calculează T_{it} şi se consideră drept candidaţi acei i pentru care $T_{it} \ge K_i$.
4. [Regula 1] Dacă $\sum_{i=1}^{N}\left(T_{it} - K_i\right)^+ \ge K_0$ se plasează o comandă a tuturor candidaţilor în intervalul t. Se face $t_0 = t$ şi pentru toate articolele candidat se face $t_i = t$.
5. [Ciclare] Se trece la pasul 2. ■

Este uşor de văzut că T_{it} şi S_{it} se calculează în timp constant cu ajutorul lui $T_{i,t-1}$ şi $S_{i,t-1}$. Deci complexitatea temporală a algoritmului este $O(NT)$.

Se poate demonstra că raportul dintre soluţiile furnizate de euristica propusă şi cea optimală este cel mult 3. Problema studiată este NP-completă.

Exemplul 7.4. Aplicând algoritmul precedent pentru datele $K_0 = 1$, $K_1 = 1$, $K_2 = 4$; $H_1 = H_2 = 1$; $d_{11} = 2$, $d_{12} = 4$, $d_{13} = 1$, $d_{21} = 3$, $d_{22} = 2$, $d_{23} = 2$; $T = 3$; $N = 2$ se obţin următoarele rezultate intermediare.

Pasul 1: Se face $t = 1$, $T_{11} = S_{11} = T_{21} = S_{21} = 0$, $t_0 = t_1 = t_2 = 1$. Se efectuează comenzile $d_{11} = 2$ şi $d_{21} = 3$.

Pasul 2: Se face $t = 2$ şi deoarece $t < T$ algoritmul continuă.

Pasul 3: Se calculează:

$$S_{12} = H_1\,(t_0 - t_1) \sum_{j=1}^{2} d_{1j} = 0 < K_1 = 1 \text{ şi } S_{22} = H_2\,(t_0 - t_2) \sum_{j=1}^{2} d_{2j} = 0 < K_2 = 4,$$

deci nu au loc plasări conform Regulei 2. Se calculează:

$$T_{12} = H_1 \sum_{j=1}^{1} j \cdot d_{1,1+j} = 1 \cdot d_{12} = 4 > K_1 = 1, \text{ rezultă că 1 este candidat, şi}$$

$$T_{22} = H_2 \sum_{j=1}^{1} j \cdot d_{2,1+j} = 1 \cdot d_{22} = 2 < K_2 = 4, \text{ rezultă că 2 nu este candidat.}$$

https://biblioteca-digitala.ro / https://unibuc.ro

Pasul 4: Deoarece $\sum_{i=1}^{2}\left(T_{i2}-K_i\right)^+$=(4-1)$^+$+(2-4)$^+$=3+0=3>$K_0$=1, se plasează comanda candidat d_{12}=4 în intervalul t=2 şi se face t_0=2 şi t_1=2.

Pasul 2: Se face t = 3 şi deoarece t = T algoritmul continuă.

Pasul 3: Se găsesc: $S_{13}=H_1(t_0-t_1)\sum_{j=2}^{3}d_{1j}=0<K_1=1$ şi $S_{23}=H_2(t_0-t_2)\sum_{j=2}^{3}d_{2j}=d_{22}+d_{23}=$ $=4=K_2$, deci, conform Regulei 2, se plasează comanda d_{22}=2 în intervalul t=2 şi se face t_2=2. Se calculează $T_{13}=H_1\sum_{j=1}^{2}j\cdot d_{1,2+j}=1\cdot d_{13}=1=K_1$, rezultă că 1 este candidat, şi $T_{23}=H_2\sum_{j=1}^{2}j\cdot d_{2,2+j}=1\cdot d_{22}+2\cdot d_{23}=6>K_2=4$, rezultă că şi 2 este candidat.

Pasul 4: Deoarece $\sum_{i=1}^{2}\left(T_{i3}-K_i\right)^+$ = (1 - 1)$^+$ + (6 - 4)$^+$ = 0 + 2 = 2 > K_0 = 1, se plasează comenzile candidat d_{13} = 1 şi d_{23} = 2 în intervalul t = 3 şi se face t_0 = 3 şi $t_1 = t_2 = 3$.

Pasul 2: Se face t = 4 şi deoarece t > T algoritmul se termină.

Toate comenzile au fost plasate conform tabelului 7.5.

Interval	1	2	3
Comenzi	d_{11}, d_{21}	d_{12}, d_{22}	d_{13}, d_{23}

Tabelul 7.5.

Soluția obținută este optimală deoarece toate cele şase comenzi au fost plasate în intervale succesive. ■

7.4. Plasarea optimală a băncilor de conturi

Punerea problemei. Numărul de zile necesar pentru eliberarea unui CEC într-o bancă din oraşul j depinde de oraşul i în care cecul este încasat. Rezultă că, pentru maximizarea fondurilor disponibile, o companie ce achită note de plată la mai mulți clienți din localități diferite, poate considera ca fiind avantajos să dețină conturi în mai multe bănci, alese în mod strategic. Achitările notelor de plată clienților din oraşul i se efectuează de către o bancă din oraşul $j(i)$ ce dispune de cel mai mare timp de eliberare. ■

Fie I = {1, 2, ..., n} mulțimea locațiilor clienților, J = {1, 2, ..., n} mulțimea locaţiilor de conturi, d_j costul fix de menținere a unui cont în oraşul j, f_i volumul în dolari a cecurilor achitate în oraşul i, p_{ij} numărul de zile, tradus în valori monetare, pentru eliberarea unui CEC în oraşul j şi încasat în oraşul i şi K numărul maxim de conturi ce poate fi menținut. Toate aceste informaţii se presupun cunoscute şi $c_{ij}=f_i\,p_{ij}$ reprezintă valoarea achitată clienţilor din oraşul i dintr-un cont din oraşul j.

https://biblioteca-digitala.ro / https://unibuc.ro

Fie:

$$y_j = \begin{cases} 1 & \text{dacă un cont este menținut în orașul } j, \\ 0 & \text{altfel.} \end{cases}$$

Plata tuturor clienților din orașul i dintr-un singur cont se consideră optimală. Fie x_{ij}=1 dacă clienții din orașul i sunt deserviți de contul j, și $x_{ij} = 0$ altfel. Dacă y_i=0, atunci x_{ij}=0, pentru toți $i \in I$. Rezultă că problema locațiilor conturilor (a băncilor de conturi), numită în continuare (P), poate fi formulată în termeni de programare liniară întreagă (IP) astfel:

$$z = \max \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} - \sum_{j\in J} d_j y_j$$

$$\sum_{j\in J} x_{ij} = 1,\ i \in I \tag{7.2}$$

$$1 \le \sum_{j\in J} y_j \le K \tag{7.3}$$

$$0 \le x_{ij} \le y_j \le 1,\ i \in I, j \in J \tag{7.4}$$

$$x_{ij} \text{ și } y_j \text{ întregi},\ i \in I, j \in J \tag{7.5}$$

În lucrarea [GEO74] se propune o relaxare lagrangeană a problemei (P) în care restricțiile (7.2) sunt ponderate cu multiplicatori și introduse în funcția obiectiv. Fie x și y vectorii $x_{ij}, i\in I, j\in J$ și $y_j, j\in J$. Fie apoi $u=(u_1,u_2,...,u_m)$ multiplicatorii restricțiilor (7.2) și

$$L(x, y, u) = \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} - \sum_{j\in J} d_j y_j - \sum_{i\in I} u_i\left(\sum_{j\in J} x_{ij} - 1\right) =$$

$$= \sum_{j\in J}\left[\sum_{i\in I}(c_{ij} - u_i)x_{ij} - d_j y_j\right] + \sum_{i\in I} u_i .$$

Problema lagrangeană este definită de $z_D(u) = \max_{(x,y)} L(x, y, u)$, condițiile (7.3), (7.4) și (7.5) și problema duală este $z_D = \min_u z_D(u)$. Se cunoaște că $z_D \ge z$.

Pentru u fixat, problema determinării lui $z_D(u)$ poate fi rezolvată după cum urmează. Din a doua expresie a lui $L(x, y, u)$ și (7.4) rezultă că valorile optimale ale lui x_{ij} în problema lagrangeană sunt date de:

$$x_{ij} = y_j, \text{ dacă } c_{ij} - u_i \ge 0,$$
$$x_{ij} = 0, \text{ altfel.}$$

Definind:

$$\rho_j(u) = \sum_{i\in I} \max(0, c_{ij} - u_i) - d_j ,$$

se deduce că valorile optimale ale lui y_j rezolvă problema:

$$\max \sum_{j\in J} \rho_j(u) y_j \text{ cu condițiile } 1 \le \sum_{j\in J} y_j \le K, y_j = 0 \text{ sau } 1, j \in J.$$

În cadrul euristicii de tip Greedy propuse, se identifică mai întâi o locație ce rezolvă problema (P) pentru K = 1 și apoi se procedează recursiv. Fie $k < K$ locații selectate. Dacă există o locație neselectată care crește valoarea funcției obiectiv, atunci se caută una care asigură o creștere maximală; altfel STOP.

https://biblioteca-digitala.ro / https://unibuc.ro

Algoritmul 7.5. (***Plasare***)

1. [Inițializări] Se face $k = 1$, $J^* = \varnothing$ și $u_i^1 = \min_{j\in J} c_{ij}$, pentru $i \in I$.
2. [Alegere] Se face $\rho_j(u^k) = \sum_{i\in I}\max\left(0, c_{ij} - u_i^k\right) - d_j, j \notin J^*$. Se alege $j_k \in J^*$ astfel încât $\rho_k = \rho_{j_k}\left(u^k\right) = \max_{j\notin J^*} \rho_j\left(u^k\right)$. Dacă $\rho_k < 0$ și $|J^*| \geq 1$, se face $k = k - 1$ și se trece la pasul 4. Altfel se face $J^* = J^* \cup \{j_k\}$.
3. [Progresare] Dacă $|J^*| \neq K$, se face $k = k + 1$, $u_i^k = \max_{j\in J^*} c_{ij} = u_i^{k-1}$ + max(0, $c_{ij_{k-1}} - u_i^{k-1}$), pentru $i \in I$, și se trece la pasul 2.
4. [Terminare] STOP. (Soluția Greedy este dată de $y_j = 1$, pentru $j \in J^*$ și $y_j = 0$ altfel. Valoarea lui $|J^*|$ este k și valoarea soluției este $z_g = \sum_{i=1}^{m} u_i^1 + \sum_{j=1}^{k} \rho_j$.) ■

Exemplul 7.5. Fie $d = 0$, $K = 2$ și

$$C = \begin{bmatrix} 0 & 11 & 6 & 9 \\ 7 & 0 & 8 & 2 \\ 7 & 3 & 0 & 3 \\ 10 & 9 & 4 & 0 \end{bmatrix}.$$

Calculele pentru exemplul dat sunt următoarele.

Pasul 1. $k = 1$, $J^* = \varnothing$, $u^1 = (0, 0, 0, 0)$.

Pasul 2. $\rho_1(u^1) = \sum_{i\in I}\max\left(0, c_{i1} - u_i^1\right) - d_1$=max(0,0-0)+max(0,7-0)+max(0,7-0)+ +max(0,10-0)=24, $\rho_2(u^1) = \sum_{i\in I}\max\left(0, c_{i2} - u_i^1\right) - d_2$ = max(0,11-0)+max(0,0-0)+max(0, 7-0)+max(0,9-0)=23, $\rho_3(u^1) = \sum_{i\in I}\max\left(0, c_{i3} - u_i^1\right) - d_3$ = max(0,6-0) + max(0,8-0) + +max(0,0-0)+max(0,4-0)=18, $\rho_4(u^1) = \sum_{i\in I}\max\left(0, c_{i4} - u_i^1\right) - d_4$=max(0,9-0)+max(0,7-0)+ +max(0,3-0)+max(0,0-0)=19, $\max_{j\notin J^*} \rho_j\left(u^1\right) = \rho_1(u^1) = 24$, $j_1 = 1$ și $J^* = \{1\}$.

Pasul 3. $u_i^2 = u_i^1$+max(0,$c_{ij_1} - u_i^1$). u_1^2=0+max(0,c_{11}-0)=0, u_2^2=0+max(0,c_{12}-0)=7, u_3^2=0+max(0,c_{13}-0)=7, u_4^2=0+max(0,c_{14}-0)=10; u^2=(0,7,7,10).

Pasul 2. $\rho_1(u^2)$ nu se calculează deoarece $J^* = \{1\}$. $\rho_2(u^2) = \sum_{i\in I}\max\left(0, c_{i2} - u_i^2\right)$ = =max(0,11-0)+max(0,0-7)+max(0,3-7)+max(0,10-10)=11, $\rho_3(u^2) = \sum_{i\in I}\max\left(0, c_{i3} - u_i^2\right)$ = = max(0,6-0)+max(0,8-7)+max(0,0-7)+max(0,4-10) = 7, $\rho_4(u^2) = \sum_{i\in I}\max\left(0, c_{i4} - u_i^2\right)$ = = max(0,9-0)+max(0,2-7)+max(0,3-7)+max(0,0-10) = 9. $\max_{j\notin\{1\}} \rho_j\left(u^2\right)$ = 11, $j_2 = 2$, $J^* = \{1, 2\}$, $|J^*| = 2$.

Pasul 4. $z_g = \sum_{i=1}^{4} u_i^1 + \sum_{j=1}^{2} \rho_j = 0 + 24 + 11 = 35$. ■

https://biblioteca-digitala.ro / https://unibuc.ro

7.5. Descompunerea euristică a matricilor de trafic în comunicarea sateliților

Punerea problemei. Fiind dată o matrice $n \times n$ a traficului informațional între n stații radio ce recepționează emisiunile unui satelit și intercomunică între ele, se cere descompunerea matricei de mai sus într-o sumă ponderată a unui număr mic de matrici de permutare astfel încât suma ponderilor să fie minimală. O matrice de permutare $n \times n$ conține n elemente nenule, așezate în linii și coloane distincte. ■

Aplicații posibile: comunicații, televiziune, radio, rețele de calculatoare.

Problema descompunerii matricii traficului (*PDMT*) se enunță formalizat în felul următor.

Fiind dată o matrice $n \times n$, $T = (t_{ij})$ cu intrări nenegative, să se găsească o descompunere a matricei T, adică o secvență de matrici de permutare $p^1, p^2, ..., p^q$ și o secvență de ponderi nenegative $l_1, l_2, ..., l_q$ astfel încât:

$$T \leq \sum_{k=1}^{q} l_k p^k \text{ (cu sumare pe elemente).} \quad (7.6)$$

Durata totală d a descompunerii este egală cu:

$$d = \sum_{k=1}^{q} l_k .$$

Problema constă în efectuarea unei descompuneri în care d să fie minimizată cu condiția ca valoarea lui q să nu fie prea mare.

Algoritmul propus are la bază ideea rotunjirii elementelor matricei T la valori întregi și rezolvarea problemei puse pentru matricea întreagă astfel obținută. Pentru o matrice T nenegativă de dimensiune $n \times n$ structura algoritmului este următoarea.

Algoritmul 7.6. (***Descompunere***)

1. [Căutare] Se caută o *unitate* $F > 0$.
2. [Rotunjire] Se rotunjesc superior intrările matricei la multipli ai lui F: $\bar{t}_{ij} = F \cdot \left\lceil \frac{t_{ij}}{F} \right\rceil$.
3. [Soluție întreagă] Se rezolvă problema descompunerii pentru matricea rezultată $\overline{T}$ (sau echivalent, pentru matricea întreagă U având ca elemente $(u_{ij}) = \left(\left\lceil t_{ij} / F \right\rceil\right)$ obținută prin împărțirea acestor elemente la F).
4. [Rectificare soluție] Se ajustează descrescător descompunerea obținută în vederea compensării rotunjirii de la pasul 2. ■

Baza algoritmului o constituie pasul 2 pentru rezolvarea căruia se consideră două metode pe care le expunem în continuare.

https://biblioteca-digitala.ro / https://unibuc.ro

Prima metodă, simplă şi rapidă, foloseşte *colorarea arcelor*. Fie u^* suma maximă pe linii şi coloane a unei matrici U şi cazul particular al descompunerii de la pasul 3 dacă $l_k = 1$ în (7.6). Această problemă coincide cu problema colorării unui graf bipartit având vârfurile r_i şi c_j, $i, j = 1, 2, ..., n$ şi câte u_{ij} arce paralele între r_i şi c_j. Problema colorării constă în atribuirea de culori tuturor arcelor astfel încât să nu existe două arce de aceeaşi culoare convergente în acelaşi vârf. Arcele de aceeaşi culoare formează o împerechere şi matricea de adiacenţă corespunzătoare, de dimensiune $n \times n$, poate fi completată arbitrar în vederea obţinerii unei permutări complete de tipul celor din formularea (7.6). Metoda descrisă furnizează descompunerea în u^* matrici de permutare, dar nu poate fi adaptată în vederea micşorării numărului acestor matrici.

Problema determinării valorii lui F constă în determinarea unei valori minimale a lui F astfel încât suma maximă pe linii şi coloane a matricii $\lceil t_{ij} / F \rceil$ să nu depăşească o valoare M dată. Pentru aceasta se parcurg succesiv toate liniile şi coloanele matricei T şi se alege cea mai mare valoare a lui F. Fie pentru precizare primul rând al matricei T, pentru care se caută cel mai mic F astfel încât:

$$\sum_{j=1}^{n} \lceil t_{1j} / F \rceil \leq M.$$

Pentru aceasta se consideră tabelul:

$$\begin{array}{llll} t_{11}, & t_{11}/2, & t_{11}/3, & t_{11}/4, \\ t_{12}, & t_{12}/2, & t_{12}/3, & t_{12}/4, \\ \vdots & & & \\ t_{1n}, & t_{1n}/2, & t_{1n}/3, & t_{1n}/4, \end{array}$$

Dacă un element t_{1j} este egal cu 0, el este eliminat deoarece nu poate contribui la identificarea valorii F. Altfel, dacă F este cuprinsă între intrările $(l - 1)$ şi l ale rândului j (sau egală cu intrarea l), atunci $\lceil t_{1j} / F \rceil = l$. Cu alte cuvinte, $\lceil t_{1j} / F \rceil$ este egală cu 1 plus numărul de elemente din rând mai mari decât F. Prin urmare:

$$\sum_{j=1}^{n} \lceil t_{1j} / F \rceil = n' + \text{numărul de elemente din tabel mai mari decât } F,$$

unde n' este numărul de elemente nenule. În continuare se identifică cel mai mic F astfel încât cel mult $M - n'$ elemente să fie mai mari decât F, adică F este al $(M - n' + 1)$-lea mai mare element din tabel.

O margine superioară a lui F este:

$$\overline{F} = \frac{r_1}{M - n + 1}, \text{ unde } r_1 = \sum_{j=1}^{n} t_{1j} / j,$$

deoarece:

$$\sum_{j=1}^{n} \lceil t_{1j} / \overline{F} \rceil < \sum_{j=1}^{n} \left(t_{1j} / \overline{F} + 1 \right) = r_1 / \overline{F} + n = (M - n + 1) + n = M + 1$$

şi prin urmare $\sum_{j=1}^{n} \lceil t_{1j} / F \rceil \leq M.$

https://biblioteca-digitala.ro / https://unibuc.ro

Dacă din fiecare rând j al tabelului se elimină primele $\lceil t_{ij} / \overline{F} \rceil - 1$ elemente, atunci ele coincid cu toate elementele mai mari decât $\overline{F}$, numărul lor fiind $\underline{M} = \sum_{j=1}^{n} \lceil t_{1j} / \overline{F} \rceil - n'$, care este cuprins între M - 2n' + 1 și M - n'. Rezultă că F, care este al (M - n' + 1)-lea mai mare element din tabelul inițial este al k-lea cel mai mare element din tabelul redus, unde k = (M - n' + 1) - $\underline{M}$ este cuprins între 1 și n.

Procedeul descris se reia pentru toate rândurile și coloanele matricei T și furnizează valoarea căutată a lui F.

În a doua metodă, pentru a reduce numărul q al matricelor componente ale matricei U, se poate aplica următoarea strategie *Greedy*: Se selectează prima matrice ponderată a permutării $l_1 p^1$ astel încât suma maximă pe linii și coloane matricii trafic rămase, egală cu max(U - $l_1 p^1$, 0) să fie redusă cât mai mult. Drept rezultat, are loc reducerea marginii u^* a numărului matricilor următoare care participă la descompunere. Modul în care se determină p^1 și l_1 este prezentat în continuare.

Fie r_i și c_j sumele elementelor din rândul i și coloana j ale matricei U, obținută după aplicarea pasului 3 al euristicii descrise mai înainte. Pentru rândurile (coloanele) critice, adică rândurile (coloanele) pentru care $r_i = u^*$ ($c_j = u^*$) trebuie satisfăcută condiția $l_1 \le u_{ij}$, dacă $p_{ij}^1 = 1$. Pentru rândurile și coloanele necritice, valorile r_i, respectiv c_j, satisfac condițiile $r_i < u^*$, respectiv $c_j < u^*$. Pentru aceste rânduri trebuie satisfăcute condițiile:

$$l_1 \le u_{ij} + (u^* - r_i), \text{ respectiv } l_1 \le u_{ij} + (u^* - c_j), \text{ dacă } p_{ij}^1 = 1.$$

Rezultă atunci că, pentru un rând oarecare (critic sau necritic), trebuie satisfăcută condiția:

$$\text{Dacă } p_{ij}^1 = 1, \text{ atunci } l_1 \le u_{ij} + (u^* - \max\{r_i, c_j\}). \tag{7.7}$$

După determinarea lui p^1 și l_1, min($l_1 p^1$, U) se scade din U. În continuare se determină p^2 și l_2 etc.

Exemplul 7.6. Fie matricea:

$$\begin{array}{cc} & \begin{array}{ccc} 16 & 15 & 14 \end{array} \\ U = \begin{array}{c} 14 \\ 15 \\ 16 \end{array} & \begin{bmatrix} 3 & 5 & 6 \\ 8 & 6 & 1 \\ 5 & 4 & 7 \end{bmatrix} \end{array} \text{ cu } u^* = 16,$$

sumele pe linii și coloane fiind notate lângă matrice. Să presupunem că se încearcă descompunerea matricei începând cu $l_1 = 7$, ceea ce ar reduce pe u^* la 9. Parcurgând succesiv elementele matricei U se deduce că singurele elemente ce satisfac condiția (7.7) sunt cele ce urmează:

https://biblioteca-digitala.ro / https://unibuc.ro

$$\begin{bmatrix} \bullet & \bullet & 6 \\ 8 & 6 & \bullet \\ \bullet & \bullet & 7 \end{bmatrix}.$$

Se observă că nici un triplet al elementelor nenule din matricea precedentă nu defineşte o matrice ce descrie o permutare. Dacă se încearcă însă $l_1 = 6$, se obţine matricea:

$$\begin{bmatrix} \bullet & \mathbf{5} & 6 \\ \mathbf{8} & 6 & \bullet \\ \bullet & \bullet & \mathbf{7} \end{bmatrix}$$

în care elementele îngroşate definesc o permutare (p^1). Matricea definită de partea din dreapta a condiţiei (7.7) se obţine adăugând la fiecare intrare în U a minimului diferenţei dintre u^* şi sumele de pe linia şi coloana corespunzătoare intrării considerate. Rezultă:

$$\begin{bmatrix} 3 & 5 & 6 \\ 8 & 6 & 1 \\ 5 & 4 & 7 \end{bmatrix} + \begin{matrix} 2 \\ 1 \\ 0 \end{matrix} \overset{\begin{matrix} 0 & 1 & 2 \end{matrix}}{\begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}} = \begin{bmatrix} 3 & 6 & 8 \\ 8 & 7 & 2 \\ 5 & 4 & 7 \end{bmatrix},$$

unde diferenţele menţionate sunt notate pe lângă cea de a doua matrice.

Ţinând seama că $l_1 = 6$ şi că permutarea p^1 este cea menţionată, rezultă descompunerea:

$$\begin{bmatrix} 3 & 5 & 6 \\ 8 & 6 & 1 \\ 5 & 4 & 7 \end{bmatrix} = \begin{bmatrix} 0 & 5 & 0 \\ 6 & 0 & 0 \\ 0 & 0 & 6 \end{bmatrix} + \begin{matrix} 9 \\ 9 \\ 10 \end{matrix} \overset{\begin{matrix} 10 & 10 & 8 \end{matrix}}{\begin{bmatrix} 3 & 0 & 6 \\ 2 & 6 & 1 \\ 5 & 4 & 1 \end{bmatrix}}.$$

Reluând procedura pentru $l_2 = 5$ şi ţinând seama de (7.7) se obţine:

$$\begin{bmatrix} 3 & 0 & \mathbf{6} \\ 2 & \mathbf{6} & 1 \\ \mathbf{5} & 4 & 1 \end{bmatrix} + \begin{matrix} 1 \\ 1 \\ 0 \end{matrix} \overset{\begin{matrix} 0 & 0 & 2 \end{matrix}}{\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}} = \begin{bmatrix} 3 & 0 & 7 \\ 2 & 6 & 2 \\ 5 & 4 & 1 \end{bmatrix}.$$

şi descompunerea:

$$\begin{bmatrix} 3 & 0 & 6 \\ 2 & 6 & 1 \\ 5 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 5 \\ 0 & 5 & 0 \\ 5 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 3 & 0 & 1 \\ 2 & 1 & 1 \\ 0 & 4 & 1 \end{bmatrix}.$$

Continuând procesul rezultă:

$$\begin{bmatrix} 3 & 0 & 1 \\ 2 & 1 & 1 \\ 0 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} + \begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 2 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Prin urmare, $q = 5$ matrici cu $(l_1, l_2, l_3, l_4, l_5) = (6, 5, 2, 2, 1)$. ■

https://biblioteca-digitala.ro / https://unibuc.ro

Observaţie. După cum se vede din exemplul prezentat, este preferabil ca pentru acelaşi l_k să existe mai multe matrici p^k. Pentru identificarea lor simultană se poate aplica o metodă *Greedy*.

7.6. Problema monedelor

Punerea problemei. Fiind date n monede, pentru care se bănuieşte că ar exista una falsă printre ele, mai grea sau mai uşoară decât celelalte, se cere identificarea acestei monede, dacă există, şi eventual o monedă nefalsă din cât mai puţine cântăriri cu o balanţă, fără greutăţi. Se presupune că abaterea de la greutatea comună a monedei false este suficient de mică astfel încât informaţia furnizată de mulţimi cu număr diferit de monede să fie neesenţială. ■

Problema nu are sens pentru $n \leq 2$. Să considerăm cazul particular $n = 4$. Ţinând seama că fiecare dintre cele patru monede poate fi mai grea sau mai uşoară decât celelalte sau să nu existe o monedă falsă, rezultă că algoritmul de rezolvare trebuie să furnizeze $2 \cdot 4 + 1 = 9$ rezultate posibile. Deoarece fiecare cântărire poate să furnizeze trei decizii (mulţimea din stânga mai grea, mulţimi de greutăţi egale sau mulţimea din dreapta mai grea), rezultă că problema poate fi descrisă, în cel mai bun caz, cu ajutorul unui arbore ternar cu două nivele. Soluţia acestei probleme, în cazul în care există la dispoziţie şi o monedă care nu e falsă, notată în continuare cu 0, este prezentată în figura 7.3. Cu 1, 2, 3 sunt notate cele trei monede şi G, U şi I reprezintă respectiv deciziile: *mai grea*, *mai uşoară* şi monede *identice*.

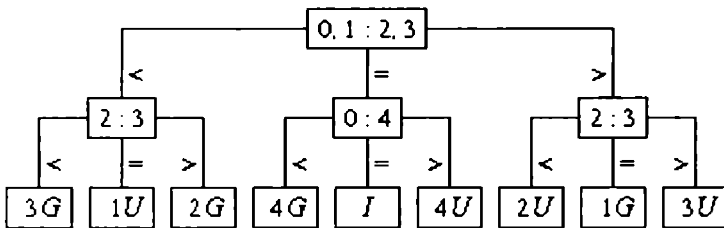


Fig. 7.3.

Se observă că existenţa monedei 0 permite efectuarea primei cântăriri folosind trei dintre cele patru monede date, fapt ce asigură, intuitiv, scăderea frecvenţei participării monedelor în cântăririle ce urmează. Această observaţie este confirmată de faptul că, pentru cazul când moneda 0 *lipseşte* sunt necesare, în cel mai rău caz, trei cântăriri (vezi figura 7.4.).

https://biblioteca-digitala.ro / https://unibuc.ro

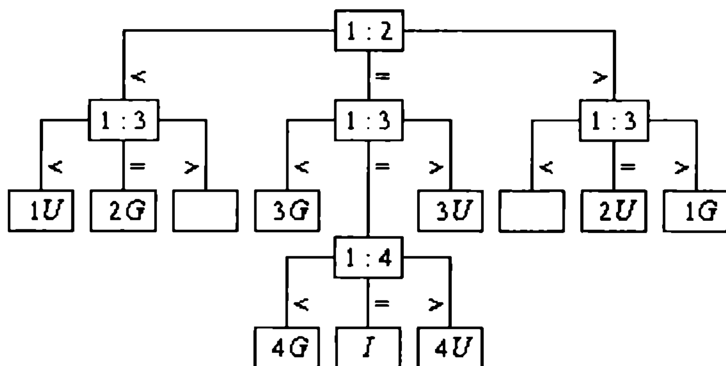


Fig. 7.4.

Fie în continuare cazul $n = 12$. O primă problemă în vederea elaborării algoritmului o constituie alegerea convenabilă a primei cântăriri. Pentru acesta se observă că pentru rezolvarea cazului cu patru monede sunt suficiente două cântăriri, dacă este disponibilă o monedă nefalsă. Rezultă că, pentru a asigura rezolvarea problemei studiate cu trei cântăriri, cele 12 monede trebuiesc împărțite în trei loturi de câte patru monede, două dintre loturi participând la prima cântărire. Dacă aceste două loturi au greutăți egale, atunci toate cele opt monede participante sunt nefalse şi oricare dintre ele poate fi folosită pentru rezolvarea problemei cu monedele din al treilea lot, în conformitate cu figura 7.3.

Rămâne de văzut modul în care poate fi rezolvat cazul în care primele două loturi au greutăți neegale. Să presupunem că aceste loturi conțin monedele numerotate 1, 2, 3, 4 şi respectiv 5, 6, 7, 8 şi că în urma primei cântăriri s-a constatat că cele două loturi au greutăți diferite. Urmează să se precizeze câte monede trebuie să participe în a doua cântărire astfel încât trei cântăriri să fie suficiente pentru rezolvarea problemei. Pentru aceasta se observă că este necesar ca în a doua cântărire să participe 6 monede, câte trei pe fiecare taler al balanței.

Dacă cele două loturi au greutăți egale, atunci rămân două monede dubioase şi pentru identificarea celei false este suficientă o singură cântărire, dacă se ştie că suma greutăților lor este mai mică (mai mare) decât greutatea a două monede nefalse.

Rămâne de precizat modul în care urmează să fie alese cele două loturi de câte trei monede. Dacă notăm cu g_i greutatea monedei de număr i,să presupunem că rezultatul primei cântăriri este $g_1 + g_2 + g_3 + g_4 < g_5 + g_6 + g_7 + g_8$. Pentru a asigura condiția că suma greutăților celor două monede rămase, după alegerea celor două loturi de câte trei monede, este mai mică decât greutatea a două monede nefalse, este suficient ca cele două monede să aparțină uneia dintre mulțimile $\{1, 2, 3, 4\}$,

https://biblioteca-digitala.ro / https://unibuc.ro

respectiv {5, 6, 7, 8}. Să presupunem că s-a decis ca în loturile de câte trei monede să participe monedele 1, 2, 5, 6, 7, 8. Rămâne de văzut cum pot fi grupate aceste monede în două loturi de câte trei monede. Este natural ca această grupare să nu acorde prioritate la oricare dintre monedele ce figurează în acelaşi lot al primei cântăriri. Două loturi ce satisfac această condiție sunt {1, 5, 6} şi respectiv {2, 7, 8}.

Să presupunem că rezultatul celei de a doua cântăriri este $g_1+g_5+g_6<g_2+g_7+g_8$. Se verifică uşor că singurele cazuri în care sunt satisfăcute rezultatele celor două cântăriri sunt $1U$, $7G$ şi $8G$. Prin urmare, ori una dintre monedele 7 sau 8 este mai grea ori moneda 1 este mai uşoară. Rezultă că în a treia cântărire trebuie să participe monedele 7 şi 8. Dacă rezultatul celei de a doua cântăriri este $g_1+g_5+g_6=g_2+g_7+g_8$, atunci rezultă că una dintre monedele 3 sau 4 este mai uşoară, deci ele trebuie să participe la a treia cântărire.

Cazul în care $g_1+g_5+g_6>g_2+g_7+g_8$ se tratează asemămător cu primul caz, rezultatul fiind ilustrat pe figura 7.5.

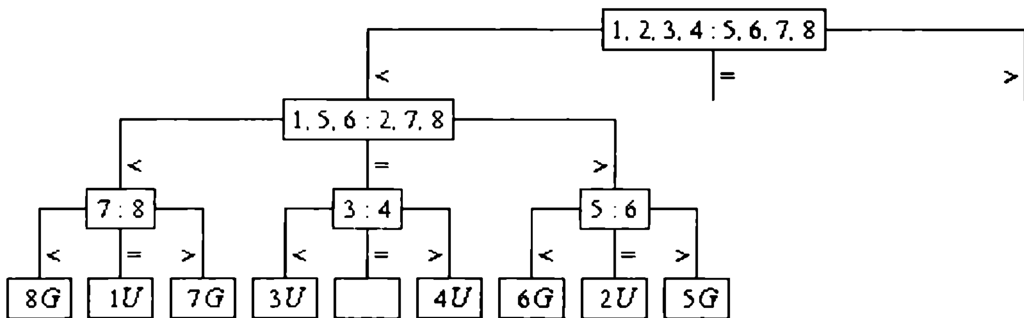


Fig. 7.5.

În cazul când rezultatul primei cântăriri este $g_1+g_2+g_3+g_4=g_5+g_6+g_7+g_8$, rezultă că toate monedele participante sunt nefalse şi problema se reduce la identificarea, conform figurii 7.3. a monedei false dintre monedele 9, 10, 11, 12, folosind una dintre monedele nefalse.

Cazul în care $g_1+g_2+g_3+g_4>g_5+g_6+g_7+g_8$ se rezolvă în mod asemănător cu primul caz şi furnizează rezultate contrare celor menționate în figura 7.5. Numărul total al rezultatelor obținute este egal cu $8+9+8=25=2\cdot12+1$, adică numărul tuturor variantelor privind existența sau nu a unei monede false.

Pentru cazul în care numărul de monede K_l este egal cu $(3^l - 1) / 2$, se poate demonstra (vezi [REI77]) că sunt suficiente l cântăriri pentru identificarea monedei false, utilizând o monedă nefalsă. În cadrul algoritmului se foloseşte strategia prezentată în exemplul cu 12 monede. Acest algoritm furnizează rezultatul după l

https://biblioteca-digitala.ro / https://unibuc.ro

paşi, deoarece $l = \log_3(2K_l + 1)$, unde $2K_l + 1$ este numărul de decizii necesare pentru identificarea faptului dacă există sau nu o monedă falsă.

În cadrul algoritmului se consideră trei stări, descrise mai departe, şi o funcţie de transfer f care decide care este comparaţia următoare, pentru o stare dată. Algoritmul este:

Algoritmul 7.7. (***MonedaFalsă***)

1. [Iniţializări] Se face $i = l$ şi $s = stare_1$.
2. [Evoluţie] Atâta timp cât $i > 1$ se face $s = f(s)$ şi $i = i - 1$.
3. [Terminare] STOP. ■

Funcţia de transfer f are următoarea structură.

Starea 1. Au rămas de făcut i cântăriri, K_i monede dubioase şi $2K_i + 1$ decizii. Se aşează pe talerul din stânga K_{i-1} monede şi moneda nefalsă şi pe talerul din dreapta $K_{i-1} + 1$ monede dubioase. Nu se folosesc K_{i-1} monede. Dacă balanţa este echilibrată se rămâne în *starea 1*. Altfel se trece în *starea 2* dacă monedele de pe talerul stâng cântăresc mai puţin decât cele de pe talerul drept şi în *starea 3* în caz contrar.

Starea 2. Au rămas de făcut i cântăriri şi moneda falsă se află ori în mulţimea U cu K_i monede dubioase ori în mulţimea G cu $K_i + 1$ monede dubioase, numărul de decizii fiind $2K_i + 1$. Se aşează pe fiecare taler K_{i-1} monede din mulţimea U şi $K_{i-1} + 1$ monede din mulţimea G. Nu se folosesc $2K_{i-1} + 1$ monede. Dacă balanţa este neechilibrată, atunci se rămâne în *starea 2*. Altfel se trece în *starea 3*.

Starea 3. Au rămas de făcut i cântăriri şi moneda falsă se află ori în mulţimea G cu K_i monede dubioase ori în mulţimea U cu $K_i + 1$ monede dubioase, numărul de decizii fiind $2K_i + 1$. Se aşează pe fiecare taler K_{i-1} monede din mulţimea G şi $K_{i-1} + 1$ monede din mulţimea U. Nu se folosesc $2K_{i-1} + 1$ monede. Dacă balanţa este neechilibrată, atunci se rămâne în *starea 3*. Altfel se trece în *starea 2*.

În cazul în care numărul n de monede date nu satisface condiţia de a fi egal cu un K_l, pentru un l dat, se determină o valoare l astfel încât $(3^{l-1} - 1) / 2 < n < (3^l - 1) / 2$ şi algoritmul descris poate fi modificat puţin în modul ilustrat pe exemplul cu 12 monede. Dacă este permisă utilizarea unui număr oricât de mare de monede nefalse, atunci este evident că algoritmul poate fi utilizat în forma prezentată.

În cadrul algoritmului descris şi a exemplelor prezentate se foloseşte strategia de reducere succesivă a problemei date într-un număr mai mare de probleme mai simple, adică micşorarea numărului de monede ce participă la cântăriri. În lucrarea [STE61] este prezentată o strategie de identificare a unei monede false dintr-o mulţime de 13 monede. Pentru aceasta se încearcă identificarea monedei false printre 12 din cele 13 monede. Atunci moneda falsă este ori una dintre cele 12, ori, dacă în

https://biblioteca-digitala.ro / https://unibuc.ro

toate cântăririle efectuate balanţa a fost echilibrată, a treisprezecea monedă, care nu a participat la cântăriri. Problema studiată se reduce deci la identificarea, dacă există, a unei monede false dintr-o mulţime de 12 monede folosind trei cântăriri.

În cadrul algoritmului propus, se consideră numai cântăriri cu câte 4 monede pe fiecare taler. Atunci, dacă balanţa este înclinată către stânga (dreapta), ori pe talerul stâng (drept) este o monedă mai grea, ori pe celălalt este o monedă mai uşoară. O monedă poate participa la una, două sau trei cântăriri. În cazul când ea rămâne pe acelaşi taler şi este falsă, balanţa va avea de fiecare dată aceeaşi înclinare. Altfel, dacă o monedă falsă este mutată, într-o cântărire următoare, de pe un taler pe altul, înclinarea balanţei se schimbă. Este evident că schimbarea înclinării balanţei constituie o informaţie utilă în vederea depistării monedei false. Trebuie observat însă că schimbarea de două ori a locului unei monede false de pe un taler pe altul nu creşte informaţia necesară identificării, deoarece are loc revenirea la aceeaşi înclinare a balanţei. Ţinând seamă de observaţiile precedente, se pot deduce rezultatele, în număr de 25, a trei cântăriri succesive, prezentate în figura 7.6.

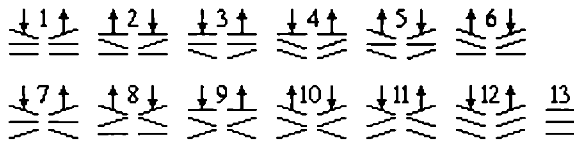


Fig. 7.6.

Să presupunem că rezultatele celor trei cântăriri sunt cele din prima configuraţie din figura 7.6. În acest caz, moneda falsă a participat numai la prima cântărire. Dacă a fost aşezată pe talerul stâng, atunci este mai uşoară şi, dacă a fost aşezată pe talerul drept, atunci este mai grea. În cazul celei de-a 23-a configuraţii, rezultă că ori moneda falsă a fost aşezată pe talerul drept şi este mai grea, ori a fost aşezată pe talerul stâng şi este mai uşoară.

Fie acum cele 13 monede numerotate 1,2,...,13. Moneda notată cu 13 nu participă la cântăriri. Se poate observa pe figura 7.6. că rezultatele primelor 24 de cântăriri sunt grupate în 12 perechi de cântăriri simetrice. Fiecărei asemenea perechi i se poate asocia una dintre cele douăsprezece monede ce participă la ântărire, deoarece dacă aceasta este moneda falsă, ea a contribuit la dezechilibrarea balanţei în cele două moduri ilustrate.

Urmărind figura 7.6. se vede că la prima cântărire pot să participe cele 8 monede numerotate respectiv 1,5,6,7,8,10,11 şi 12. Presupunem că au fost aşezate pe cele două talere monedele 5,6,8,10, respectiv 1,7,11,12. Se constată, urmărind figura 7.6., că monedele 6,8,10,11 şi 12 trebuie să participe şi la a doua cântărire şi

https://biblioteca-digitala.ro / https://unibuc.ro

monedele 8 şi 11 trebuie să treacă pe celălalt taler. Tot din figura 7.6. rezultă că la această cântărire trebuie să mai participe monedele 2,4 şi 9. Toate monedele participante sunt aşezate astfel pe talere: 2,6,10,11, respectiv, 4,8,9,12. În continuare se constată că moneda 3 participă numai la o ultimă cântărire şi trebuie aşezată pe talerul drept, rezultând aşezările 5,7,9,11, respectiv, 3,4,10,12. Urmărina din nou figura 7.6., se deduce că în urma cântăririlor alese, monedele false sunt mai grele sau mai uşoare, după cum săgeţile corespunzătoare sunt orientate în jos sau în sus.

7.7. Organizarea memoriilor liniare

Un factor important în utilizarea memoriilor auxiliare îl constituie *timpul de acces* care are două componente principale: *timpul de localizare* şi *timpul de transfer*. Prin timp de localizare se înţelege timpul necesar poziţionării capătului de citire şi scriere la înregistrarea implicată în transfer de informaţie şi prin timp de transfer se înţelege timpul necesar efectuării operaţiei efective de citire sau scriere a informaţiei.

Deoarece timpul de transfer depinde, în cea mai mare parte, de caracteristicile fizice ale terminalelor şi de modul de acces, nu se pot obţine îmbunătăţiri ale acestui timp la nivel de sistem de operare. De aceea, majoritatea metodelor de organizare şi gestionare a memoriilor auxiliare au în vedere optimizări ale timpului de localizare.

Memoriile liniare sau *memoriile unidimensionale* sunt memoriile în care informaţia este dispusă sub formă de listă liniară secvenţială. Exemplul specific de memorie liniară îl constituie banda magnetică. Uneori şi discul magnetic poate fi considerat ca o memorie liniară, fiecare cilindru fiind considerat un element al acestei memorii liniare. În cazul memoriilor liniare, deplasarea capătului de citire şi scriere se poate măsura folosind o *metrică euclidiană*.

Vom reprezenta memoria liniară ca dreapta reală având înregistrările localizate în punctele cu coordonate numere întregi. Vom presupune că cererile de acces la informaţie sunt plasate într-o *coadă* şi că după executarea fiecărei operaţii capătul de citire/scriere se opreşte la ultima înregistrare la care a avut acces. Vom considera că înregistrările din fişier sunt $R_1, R_2, ..., R_n$ şi probabilităţile cu care sunt accesate sunt respectiv $p_1, p_2, ..., p_n$. Funcţia obiectiv de minimizat este *distanţa medie* de deplasare a capătului de citire/scriere de la o cerere la alta dată de expresia:

$$D(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} p_i p_j d^{(\pi)}(i, j),$$

unde $d^{(\pi)}(i, j)$ reprezintă distanţa între înregistrările R_i şi R_j şi depinde de permutarea π a înregistrărilor şi de metrică. Dacă înregistrările R_i şi R_j ocupă poziţiile s şi t, atunci:

$$d^{(\pi)}(i, j) = d_2(s, t) = |s - t|,$$

unde d_2 este metrica euclidiană.

https://biblioteca-digitala.ro / https://unibuc.ro

Problema care se pune este de a găsi un aranjament π astfel încât $D(\pi)$ să fie minim pentru înregistrările date, având probabilitățile de acces date şi pentru o metrică fixată. Problema a fost rezolvată de Hardy, Littlewood şi Polya. Soluția este dată de următoarea teoremă a cărei demonstrație poate fi găsită în [BÂS89]. Aranjamentul rezultat aplicând procedeul din teoremă se numeşte *aranjament orgă.*

Teorema 7.1. Dacă înregistrările sunt numerotate în ordinea necrescătoare a probabilităților asociate lor, se obține o distanță medie minimă parcursă de capătul de citire/scriere dacă se plasează R_1 într-un punct pe axa reală şi apoi se plasează în ordine R_2, R_3, ..., R_n alternativ la stânga şi la dreapta înregistrărilor deja plasate, pe locurile libere cele mai apropiate de R_1. Deci înregistrările de indice par sunt de o parte şi cele de indice impar sunt de cealaltă parte a înregistrării R_1. ■

O altă problemă este *partajarea înregistrărilor* dată de cazul în care din acelaşi loc se pot citi/scrie mai multe înregistrări. Este cazul discului magnetic la care unui punct îi corespunde un cilindru care poate să conțină mai multe înregistrări. Problema care se pune este de a împărți fişierul în pagini de câte k înregistrări, cu k dat. Accesul la o înregistrare presupune accesul mai întâi la pagina care conține înregistrarea respectivă. Vom presupune că timpul de localizare a unei înregistrări în pagină este neglijabil.

Considerăm înregistrările $R_1, R_2, ..., R_{mk}$ cu probabilitățile de acces $p_1, p_2, ..., p_{mk}$. Problema constă în partiționarea celor mk înregistrări în m pagini de câte k înregistrări şi plasarea celor m pagini într-o memorie liniară care să minimizeze deplasarea capătului de citire/scriere. Mai precis, presupunând că în pagina i sunt plasate înregistrările $R_1^i, R_2^i, \ldots, R_k^i$ şi notând *probabilitatea de acces a paginii i* cu $q_i = \sum_{j=1}^{k} p_j^i$, unde p_j^i este probabilitatea de acces la înregistrarea R_j^i, funcția de cost ce trebuie minimizată este:

$$D(\pi; q_1, q_2, ..., q_m) = \sum_{i=1}^{m} \sum_{j=1}^{m} q_i q_j d^{(\pi)}(i, j),$$

unde $d^{(\pi)}(i, j)$ este distanța între paginile i şi j.

Dacă valorile $q_1, q_2, ..., q_m$ ar fi cunoscute, problema plasării paginilor se reduce la problema precedentă de plasare a unor înregistrări de probabilități date pe memoria liniară, de unde rezultă că un aranjament orgă în raport cu q_1, q_2, ..., q_m minimizează $D(\pi; q_1, q_2, ..., q_m)$. Deci problema se poate pune sub forma următoare: cum trebuie partiționate cele mk înregistrări în m pagini de câte k înregistrări, paginile având probabililtățile q_1, q_2, ..., q_m, astfel încât să se minimizeze $D(\pi^*; q_1, q_2, ..., q_m)$, unde π^* este aranjamentul orgă pentru q_1, q_2, ..., q_m. Adică se caută minimizarea expresiei:

$$D(q) = \sum_{1 \le i < j \le m} q_i q_j d(i, j),$$

https://biblioteca-digitala.ro / https://unibuc.ro

unde $d(i, j)$ este distanţa dintre paginile i şi j în aranjamentul orgă. Dacă paginile sunt numerotate în ordinea necrescătoare a probabilităţilor lor, atunci $d(i, j) = |j - i| / 2$ dacă i şi j au aceeaşi paritate şi $d(i, j) = (j + i - 1) / 2$ dacă au parităţi diferite.

Pentru rezolvarea problemei se foloseşte următorul algoritm de tip Greedy.

Algoritmul 7.8. (***Aranjare pe pagini***)

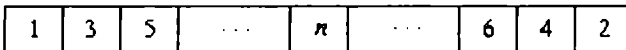
1. [Ordonare] Se numerotează înregistrările astfel încât $p_1 \geq p_2 \geq ... \geq p_{mk}$.
2. [Partiţionare] Se pun înregistrările $R_1, R_2, ..., R_k$ în pagina 1, înregistrările $R_{k+1}, R_{k+2}, ..., R_{2k}$ în pagina 2, înregistrările $R_{2k+1}, R_{2k+2}, ..., R_{3k}$ în pagina 3 ş. aşa mai departe până se complectează toate cele m pagini.
3. [Probabilităţi pagini] Se calculează $q_i = \sum_{j=(i-1)k+1}^{ik} p_j$, pentru $i = 1, 2, ..., m$.
4. [Aranjare pagini] Se aranjează paginile în aranjament orgă. ■

Teorema 7.2. Partiţionarea obţinută aplicând algoritmul 7.8. este optimală.

O demonstraţie a acestei teoreme se poate găsi în [WON83]. ■

Vom presupune în continuare că nu se cunoaşte iniţial mulţimea acelor înregistrări care urmează să fie plasate pe memoria liniară, aceste înregistrări sunt plasate una câte una în ordinea sosirii în sistem şi ele rămân în locul în care au fost plasate iniţial. Acest mod de lucru corespunde tipului de alocare a memoriei pentru utilizarea minicalculatoarelor.

Fie $u_1, u_2, ..., u_n$ un şir de n utilizatori care sosesc în sistem la momente diferite de timp, fiecărui utilizator u_i fiindu-i asociată o frecvenţă de acces la memorie f_i. Fiecărui utilizator îi este alocat un loc liber i în memoria liniară. Vom presupune că locaţiile sunt numerotate ca în figura 7.7.



1	3	5	...	n	...	6	4	2

Fig. 7.7.

Vom nota cu f_i frecvenţa de acces a utilizatorului plasat în locaţia i şi cu f^s frecvenţa utilizatorului sosit la momentul s. Vom considera toate elementele de aceeaşi lungime. După ce au sosit n utilizatori în sistem ocupând toată memoria, este generat un şir de cereri $r_1, r_2, r_3, ...$, fiecare dintre cereri reprezentând un acces la locaţia j cu probabilitatea $p_j = f_j / \sum_{i=1}^{n} f_i$.

Problema constă în a determina acele plasări în memorie care minimizează distanţa medie $E(D)$ de deplasare pentru două cereri consecutive, unde:

$$E(D) = \sum_{1 \leq i < j \leq n} p_i p_j d(i, j)$$

şi $d(i, j) = |j - i| / 2$, pentru i şi j de aceeaşi paritate, şi $d(i, j) = n - (j + i - 1) / 2$, altfel.

https://biblioteca-digitala.ro / https://unibuc.ro

Dacă toate probabilitățile ar fi cunoscute înainte de plasare, s-ar putea obține chiar aranjarea optimă care este *aranjamentul orgă*. Cum ele nu sunt cunoscute, se folosesc diverse strategii care permit aproximarea unei plasări optime. Una dintre ele este algoritmul următor.

Algoritmul 7.9. (***Plasarea euristică***)

1. [Plasare u_1] Se plasează utilizatorul u_1, care soseşte la timpul 1 în poziția i dacă şi numai dacă $(i - 1) / n < f^1 \le i / n$; se face $t = 1$.
2. [Terminare] Dacă $t = n$, atunci STOP.
3. [Plasare element nou] Se face $t = t + 1$. Din cele $n - t + 1$ locații rămase libere, se alocă a i-a în ordinea numerotării inițiale dacă şi numai dacă:
$$(i - 1) / (n - t + 1) < f^t \le i / (n - t + 1).$$
4. [Ciclare] Se trece la pasul 2. ■

Dacă se notează cu D_h distanța medie din plasarea euristică, atunci:

$$E(D_h) \approx 7n / 30 + 7H_n / 60 + 14 / 90,$$

unde $H_n = 1 + 1/2 + 1/3 + ... + 1/n$.

Alte metode care se pot aplica în acest caz pot fi:

- *Plasarea aleatoare* în care se alocă unui utilizator oricare dintre locațiile libere în momentul sosirii în sistem. Dacă se notează cu D_r distanța medie din plasarea aleatoare, atunci:
$$E(D_r) \approx (n^2 - 1) / (3n + 1).$$
- *Plasarea statică optimală* care se poate aplica dacă toate frecvențele sunt cunoscute. Atunci se ordonează crescător utilizatorii în raport de frecvențele de acces asociate lor şi se plasează al i-lea utilizator în locația i. Dacă se notează cu D_o distanța medie din plasarea statică optimală, atunci:
$$E(D_o) \approx 7n / 30 + 5 / 36.$$

După cum se vede, plasarea euristică este asimptotic la fel de bună ca plasarea optimală.

Vom aborda în continuare problema *memorării înregistrărilor de lungimi diferite*. Să presupunem că înregistrările $R_1,R_2,...,R_n$ au probabilitățile de acces $p_1,p_2,...,p_n$ şi respectiv lungimile $l_1,l_2,...,l_n$. Problema care se pune este de a determina o alocare a înregistrărilor care să minimizeze distanța medie parcursă de capătul de citire/scriere. Distanța dintre înregistrările i şi j depinde acum de lungimile înregistrărilor aflate între ele. Problema este NP-completă.

Numim *alocare* sau *aranjament* al celor n înregistrări o permutare $\pi=(\pi_1,\pi_2,...,\pi_n)$ a numerelor $1,2,...,n$, unde π_i este indicele înregistrării alocate celei de-a i-a poziții de la stânga la dreapta. Distanța medie parcursă de capătul de citire/scriere este:

https://biblioteca-digitala.ro / https://unibuc.ro

$$D(\pi) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{\pi_i} p_{\pi_j} d(i, j),$$

unde $d(i, j)$ este distanța parcursă pentru a avea acces la înregistrarea din poziția j după ce a avut loc acces la înregistrarea i. Pentru banda magnetică se obține:

$$d_T(i, j) = \begin{cases} \sum_{k=i+1}^{j-1} l_{\pi_k}, & \text{pentru } i < j, \\ \sum_{k=j}^{i} l_{\pi_k}, & \text{pentru } i \geq j, \end{cases}$$

de unde rezultă:

$$D_T(\pi) = \sum_{i=1}^{n} p_{\pi_i} l_{\pi_i} + 2\sum_{i=2}^{n-1} l_{\pi_i} \left(\sum_{j=1}^{i-1} p_{\pi_j}\right)\left(\sum_{j=i+1}^{n} p_{\pi_j}\right).$$

Pentru discurile magnetice, lungimea înregistrării este dată de numărul de cilindrii ocupați de înregistrarea respectivă. În acest caz, distanța este dată de:

$$d_D(i, j) = \begin{cases} \left(\sum_{k=i+1}^{j-1} l_{\pi_k}\right) + 1, & \text{pentru } i < j, \\ \left(\sum_{k=j}^{i} l_{\pi_k}\right) - 1, & \text{pentru } i \leq j, \end{cases}$$

de unde rezultă:

$$D_D(\pi) = \sum_{i=1}^{n} p_{\pi_i}\left(l_{\pi_i} - p_{\pi_i}\right) + 2\sum_{i=2}^{n-1} l_{\pi_i} \left(\sum_{j=1}^{i-1} p_{\pi_j}\right)\left(\sum_{j=i+1}^{n} p_{\pi_j}\right).$$

Deoarece $D_D(\pi) = D_T(\pi) - \sum_{i=1}^{n} p_{\pi_i}^2$, rezultă că un aranjament optim pentru banda magnetică este optim și pentru discul magnetic. În cele ce urmează ne vom referi de aceea numai la cazul benzilor magnetice.

Dacă renumerotăm înregistrările în ordinea de la stânga la dreapta se obține $\pi_i = i$ și:

$$D(\pi) = \sum_{i=1}^{n} p_i l_i + 2\sum_{i=2}^{n-1} l_i \left(\sum_{j=1}^{i-1} p_j\right)\left(\sum_{j=i+1}^{n} p_j\right),$$

un aranjament optim S_o este un aranjament care satisface inegalitatea $D(S_o) \leq D(S)$, pentru orice aranjament posibil S.

Se poate demonstra (vezi [BÂS89]) că, dacă într-un aranjament se interschimbă două înregistrări vecine t și $t + 1$, se obține o micșorare a distanței medii parcurse de capătul de citire/scriere dacă $p_t/l_t < p_{t+1}/l_{t+1}$ și $\sum_{i=1}^{t-1} p_i > \sum_{i=t+2}^{n} p_i$ sau $p_t/l_t < p_{t+1}/l_{t+1}$ și $\sum_{i=1}^{t-1} p_i < \sum_{i=t+2}^{n} p_i$. Dacă $p_t/l_t = p_{t+1}/l_{t+1}$, pentru $1 \leq i,j \leq n$, atunci orice aranjament dă aceeași valoare medie a distanței parcurse de capătul de citire/scriere.

https://biblioteca-digitala.ro / https://unibuc.ro

Se numeşte *arnjament şa* un aranjament pentru care esistă o poziție h, unde $1 \le h \le n$, astfel încât $p_1 / l_1 \le p_2 / l_2 \le \ldots \le p_h / l_h \ge \ldots \ge p_{n-1} / l_{n-1} \ge p_n / l_n$.

Teorema 7.3. Fie π' şi π'' două aranjamente ale unei mulțimi de înregistrări cu probabilitățile de acces $p_1, p_2, \ldots, p_n$ şi lungimile $l_1, l_2, \ldots, l_n$. Atunci:

$$D(\pi') / D(\pi'') \le (1 + p_{\min}) / (2p_{\min}),$$

unde $p_{\min} = \min_{1 \le i \le n} \{p_i\}$. În plus, pentru orice $n \ge 3$ număr impar, există o mulțime de înregistrări pentru care câtul $D(\pi') / D(\pi'')$ se poate apropia oricât de mult de $(1 + p_{\min}) / (2p_{\min})$.

O demonstrație a acestei teoreme se găseşte în [BÂS89]. ■

Din teorema 7.3. rezultă că un aranjament oarecare poate fi destul de departe de un aranjament optimal. Se poate demonstra o proprietate analoagă pentru aranjamentele şa.

Teorema 7.4. Dacă π' şi π'' sunt două aranjamente şa ale unei mulțimi de n înregistrări cu probabilitățile de acces $p_1, p_2, \ldots, p_n$ şi lungimile $l_1, l_2, \ldots, l_n$. Atunci:

$$D(\pi') / D(\pi'') \le 1 + n / 2.$$

O demonstrație a acestei teoreme se găseşte în [WON83]. ■

S-a demonstrat de fapt că pentru $n \le 4$ are loc inegalitatea $D(\pi') / D(\pi'') \le 2$ şi se presupune că aceasta ar fi adevărată pentru orice n.

Pentru construirea soluției aproximative se folosesc două euristici:

1. Se construieşte aranjamentul orgă față de mulțimea $\{p_i / l_i\}$.
2. Se alege cel mai bun dintre aranjamentele orgă față de $\{p_i\}$ şi respectiv $\{1 / l_i\}$.

O ultimă problemă pe care o prezentăm legată de organizarea memoriilor liniare este *prelucrarea pe loturi a cererilor*. Prin prelucrarea pe loturi a cererilor se înţelege tratarea în acelaşi timp a unei grupe de cereri, fiecare grupă conținând acelaşi număr de cereri. În acest caz se pun două probleme principale:

1. Pentru un aranjament dat, care este ordinea de considerare a cererilor de acces dintr-un lot.
2. Care este cel mai bun aranjament al înregistrărilor.

Răspunsul la a doua problemă este relativ simplu. În majoritatea strategiilor de prelucrare pe loturi a cererilor, aranjamentul optim este aranjamentul orgă. Rămâne de găsit soluții pentru prima problemă.

Să presupunem că locațiile de memorie sunt numerotate cu $1,2,\ldots,n$ de la stânga la dreapta şi că pentru fiecare $i=1,2,\ldots,n$, în locația i a fost memorată înregistrarea R_i având probabilitatea de acces p_i. Să presupunem că la momentul t au fost generate b cereri de acces $1 \le L_1 \le L_2 \le \ldots \le L_b \le n$, unde L_i indică înregistrarea implicată în cerere.

https://biblioteca-digitala.ro / https://unibuc.ro

Dacă în momentul sosirii cererilor capătul de citire/scriere este poziționat la locația x, o strategie specifică ordinea în care se trece plecând din x prin locațiile $L_1, L_2, ..., L_b$. Capătul de citire/scriere se oprește la ultima locație y la care a avut acces, aceasta devenind locația de plecare pentru următorul lot. Notând cu $d(x, y)$ distanța parcursă de capătul de citire/scriere de când pleacă din x până ajunge în y, costul C poate fi evaluat prin valoarea medie a lui $d(x, y)$. În continuare vom nota cu S pe L_1, pe care o numim *marginea stângă*, și cu D pe L_b, pe care o numim *marginea dreaptă*.

Se pot defini următoarele strategii:

- *Strategia stângă* (S): Din locația curentă x se merge la S după care se prelucrează cererile în ordinea crescătoare a locațiilor până la D și se oprește procesul.
- *Strategia dreaptă* (D): Din locația curentă x se merge la D după care se prelucrează cererile în ordinea descrescătoare a locațiilor până la S și se oprește procesul.
- *Strategia alternantă* (A): Se folosește strategia stângă pentru un lot de ordin impar și strategia dreptă pentru un lot de ordin par.

Dacă se notează cu $COST_S$, $COST_D$ și $COST_A$ costurile celor trei strategii,

$$\lambda_i = Pr(S \geq i) = \left(\sum_{j=i}^{n} p_j\right)^b \text{ și } \rho_i = Pr(D \leq i) = \left(\sum_{j=1}^{i} p_j\right)^b,$$

se pot demonstra următoarele proprietăți (vezi [BÂS89]):

Lema 7.1. Notând cu d_b distanța dintre S și D, valoarea medie a lui d_b este:

$$E(d_b) = E(D - S) = \sum_{i=1}^{n-1}\left(1 - \lambda_{i+1} - \rho_i\right). \blacksquare$$

Teorema 7.5. Au loc egalitățile:

$$COST_S = E(d_b) + \sum_{i=1}^{n-1}\left(1 + \lambda_{i+1}\right)\left(1 - \rho_i\right) + \sum_{i=1}^{n-1}\lambda_{i+1}\rho_i = 2E(d_b) + 2\sum_{i=1}^{n-1}\lambda_{i+1}\rho_i =$$

$$= 2\sum_{i=1}^{n-1}\left(1 - \lambda_{i+1}\right)\left(1 - \rho_i\right). \blacksquare$$

Consecință. $COST_S$ este simetric, adică nu se schimbă dacă se pune p_{n-i+1} în loc de p_i, pentru $i = 1, 2, ..., n$. De aici rezultă $COST_S = COST_D$.

Teorema 7.6. Au loc egalitățile:

$$COST_A = E(d_b) + \sum_{i=1}^{n-1}\rho_i\left(1 - \rho_i\right) + \sum_{i=1}^{n-1}\lambda_{i+1}\left(1 - \lambda_{i+1}\right) = \sum_{i=1}^{n-1}\left(1 - \rho_i^2 - \lambda_{i+1}^2\right) = E(d_{2b}). \blacksquare$$

Definim *operația de interschimbare* prin trecerea de la un aranjament $(R_1, R_2, ..., R_n)$ la un alt aranjament ce se obține interschimbând între ele R_i și R_{n-i+1} dacă și numai dacă $p_i > p_{n-i+1}$, pentru $1 \leq i \leq [n / 2]$. În mod similar, definim *operația de interschimbare inversă* cea prin care se obține un nou aranjament interschimbând R_{i+1} și R_{n-i+1} dacă și numai dacă $p_{i+1} < p_{n-i+1}$ (R_1 rămâne pe loc).

https://biblioteca-digitala.ro / https://unibuc.ro

Lema 7.2. Aranjamentul orgă minimizează costul C al unei strategii dacă pentru orice $n \geq 1$ sunt satisfăcute următoarele condiții:

1. C este simetric, adică $C(R_1, R_2, ..., R_n) = C(R_n, R_{n-1}, ..., R_1)$.
2. Dacă $p_n = 0$, atunci $C(R_1, R_2, ..., R_{n-1}, R_n) = C(R_1, R_2, ..., R_{n-1})$.
3. Dacă $(R'_1, R'_2, \ldots, R'_n)$ este aranjamentul obținut din $(R_1, R_2, ..., R_n)$ prin operația de interschimbare, atunci $C(R_1, R_2, ..., R_n) \geq C(R'_1, R'_2, \ldots, R'_n)$. ■

Teorema 7.7. Aranjamentul orgă minimizează $COST_S$. ■

Teorema 7.8. Aranjamentul orgă minimizează $COST_A$. ■

Teorema 7.9. Aranjamentul orgă minimizează $E(d_b)$. ■

Teorema 7.10. $E(d_{2b}) \leq 2\, E(d_b)$. ■

Consecințe: 1. $COST_A \leq COST_S = COST_D$ pentru orice aranjament.

2. $C_{\min} \leq C_A \leq 2C_{\min}$, unde $C_{\min}$ este costul minim pentru orice strategie și orice aranjament și C_A este $COST_A$ pentru aranjamentul orgă.

- *Strategia cea mai apropiată extremitate* (N). Din locația curentă se mută capătul de citire/scriere în cea mai apropiată extremitate după care se parcurg locațiile lotului și se oprește în cealaltă extremitate. Dacă cele două extremități sunt la egală distanță, se alege oricare dintre ele.

Modelul matematic de analiză a acestei strategii utilizează *lanțurile Markov*. Notând cu $COST_N$ costul acestei strategii, în general, $COST_N \leq COST_A$. Există contraexemple pentru care inegalitatea nu este satisfăcută.

- *Strategii B-optimale* (B). Strategii care minimizează distanța medie totală de deplasare a capătului de citire/scriere plecând dintr-o poziție x și parcurgând B loturi de cereri.

Costul unei strategii B-optimale este distanța totală medie împărțită la numărul loturilor B. Pentru B=1 se obține strategia N. La stabilirea unei strategii B-optimale se ține seama de următoarele proprietăți ușor de demonstrat.

Lema 7.3. Într-o strategie B-optimală, imediat după accesul tuturor înregistrărilor unui lot, capătul de citire/scriere se află la una din extremități. ■

Lema 7.4. Într-o strategie B-optimală, un lot se prelucrează mutând capătul de citire/scriere direct la una din extremitățile lotului și prelucrând cererile cu mișcarea capătului de citire/scriere către cealaltă extremitate a lotului. ■

Lema 7.5. Pentru orice strategie ale cărei decizii depind de loturile anterioare, există o strategie de același cost ale cărei decizii depind numai de lotul curent. ■

O strategie care satisface lema 7.5. se poate da sub forma unui șir $d_B(x; s, d)$, $d_{B-1}(x; s, d)$, ..., $d_1(x; s, d)$ de *funcții de decizie*, unde $d_i(x; s, d)$ este probabilitatea cu

https://biblioteca-digitala.ro / https://unibuc.ro

care capătul de citire/scriere se mută la stânga din poziţia curentă x pentru prelucrarea lotului i, ultimul lot având extremele s şi d. O strategie este *deterministă* dacă pentru orice i, x, s şi d valorile $d_i(x; s, d)$ sunt 0 sau 1. În cazul în care $d_i(x; s, d)$ poate lua orice valori din [0, 1] spunem că strategia este *nedeterministă*.

Lema 7.6. Pentru orice strategie nedeterministă există o strategie deterministă de cost cel mult egal cu costul strategiei nedeterministe. ■

Ţinând seama de lema 7.6., se poate deduce că strategiile B-optimale se află printre cele 2^B posibilităţi de a construi funcţii de decizie cu valori 0 sau 1.

Lema 7.7. O strategie optimală este optimală pentru orice poziţie iniţială a capătului de citire/scriere. ■

Pentru a defini clasa strategiilor B-optimale, se construieşte un şir de funcţii $C_B(x)$, pentru $B = 0, 1, \ldots$ date de formulele:

$$C_0(x) = 0, \text{ pentru } 1 \le x \le n,$$

$$C_B(x) = \sum_{s \le d} Pr\left(S = s, D = d\right) \min[|x - s| + C_{B-1}(d), |x - d| + C_{B-1}(s)] + E(d_b),$$

$$\text{pentru } 1 \le x \le n \text{ şi } B = 1, 2, \ldots$$

- *Strategia recursivă.* Fie i numărul loturilor ce mai sunt de prelucrat şi x poziţia curentă a capătului de citire/scriere. Pentru prelucrarea lor se procedează astfel:

1. Se merge în extrema stângă, apoi se prelucrează cererile până la extrema dreaptă dacă $|x - S| + C_{i-1}(D) < |x - D| + C_{i-1}(S)$.
2. Se merge în extrema dreaptă, apoi se prelucrează cererile până la extrema stângă dacă $|x - S| + C_{i-1}(D) > |x - D| + C_{i-1}(S)$.
3. Dacă $|x - S| + C_{i-1}(D) = |x - D| + C_{i-1}(S)$, se poate executa 1. sau 2. la alegere. ■

Teorema 7.11. Cu notaţiile de mai sus, $C_B(x)$ este costul optimal pentru prelucrarea a B loturi cu poziţia iniţială x şi strategia recursivă este o strategie B-optimală având costul $C_B(x) / B$. ■

- *Strategia ∞-optimală.* Dacă $COST_B^r(x)$ este costul prelucrării a B loturi plecând din poziţia x şi folosind strategia r, atunci *costul asimptotic* mediu al strategiei r, notat $COST(r)$, se defineşte prin $\lim_{B \to \infty} COST_B^r(x) / B$. O strategie ∞-optimală are costul asimptotic mediu mai mic sau egal decât oricare altă strategie.

Algoritmul 7.10. (∞-*optimal*)

1. [Iniţializare] Se consideră strategia *cea mai apropiată extremitate* drept strategie *curentă*.
2. [Evaluare] Se calculează $f_B(x)$, costul prelucrării a B loturi plecând din poziţia x, pentru strategia *curentă*, pentru $B = 1, 2, \ldots, f_0(x) = 0$;

https://biblioteca-digitala.ro / https://unibuc.ro

$$f_B(x) = \sum_{s \le d} Pr(S = s, D = d)\delta(x, s, d)$$, unde:

$$\delta(x,s,d) = \begin{cases} |x-s| + (d-s) + f_{B-1}(d), & \text{dacă în strategia } \textit{curentă} \text{ se merge mai întâi în } s \text{ din poziția } x \text{ cu extremele } s \text{ și } d, \\ |x-d| + (d-s) + f_{B-1}(s), & \text{dacă se merge întâi în } d. \end{cases}$$

Calculele continuă până când se obține un B pentru care $f_B(x)$ - $f_{B-1}(x)$ este aproximativ constant în raport cu x.

3. [Rectificare] Se calculează strategia *nouă* astfel: pentru orice x, s și d, dacă $|x - d| - |x - s| + \lim_{B\to\infty}[f_B(s) - f_B(d)] < 0$ (limita poate fi aproximată cu $f_B(s) - f_B(d)$, pentru B suficient de mare), strategia *nouă* mută capătul de citire/scriere mai întâi în extrema dreaptă plecând din x și având extremele s și d, altfel se mută mai întâi la stânga.
4. [Terminare sau ciclare] Dacă strategia *nouă* coincide cu strategia *curentă*, atunci STOP. (Aceasta este stratetgia ∞-optimală.) Altfel strategia *nouă* devine strategia *curentă* și se trece la pasul 2. ■

n	B	Strategia stângă	Strategia alternantă	Strategia cea mai apropiată extremitate	Strategia ∞-optimală
100	5	133.389	81.802	80.753	80.713
	10	163.603	90.443	90.408	90.394
	15	174.950	93.498	93.498	93.496
200	5	266.803	163.628	161.529	161.452
	10	327.256	180.936	180.869	180.839
	15	349.976	187.072	187.070	187.067

Tabelul 7.6.

În tabelul 7.6. sunt date costurile asimptotice în cazul strategiilor stângă, alternantă, cea mai apropiată extremitate, respectiv ∞-optimală pentru distribuția uniformă cu n = 100 și n = 200 de înregistrări, cu un număr de 5, 10 și 15 loturi. După cum se poate observa, diferențele între ultimile trei strategii sunt nesemnificative. Deci strategia alternantă și strategia cea mai apropiată extremitate sunt euristici care aproximează suficient de bine soluția optimală.

7.8. Arbori binari optimi pentru căutare

Arborii binari pentru căutare sunt arborii binari etichetați în care eticheta atribuită unui nod este mai mare decât etichetele atribuite nodurilor din subarborele stâng și mai mică decât etichetele atribuite nodurilor din subarborele drept. Dacă se

https://biblioteca-digitala.ro / https://unibuc.ro

consideră că rădăcina este de *nivelul* 1 şi fiecare alt nod al arborelui are *nivelul* cu 1 mai mare decât *nivelul* tatălui său, numărul de comparaţii pentru a determina un element dintre cele care se găsesc în arborele binar pentru căutare este egal cu *nivelul* nodului asociat.

Eficienţa unui arbore binar pentru căutare este dată de numărul mediu de comparaţii necesare pentru determinarea unui element. Dacă elementele sunt echiprobabile, eficienţa cea mai bună se obţine pentru *arborii binari echilibraţi,* aceştia fiind arbori binari în care nodurile ce nu au câte doi fii se găsesc pe cel mult două niveluri consecutive.

Să considerăm acum cazul în care etichetele sunt căutate cu frecvenţe date. De exemplu, în figura 7.8. este dat arborele optim de căutare pentru cele mai frecvent utilizate 31 de cuvinte din limba engleză şi anume (în paranteze sunt date frecvenţele): *the* (15568), *of* (9767), *and* (7638), *to* (5739), *a* (5074), *in* (4312), *he* (3727), *that* (3017), *is* (2509), *I* (2292), *it* (2255), *for* (1869), *as* (1853), *with* (1849), *was* (1761), *his* (1732), *be* (1535), *not* (1496), *by* (1392), *but* (1379), *have* (1344), *you* (1336), *which* (1291), *are* (1222), *on* (1155), *or* (1101), *her* (1093), *had* (1062), *at* (1053), *from* (1039) şi *this* (1021). Pentru o căutare cu succes sunt necesare în medie 3.437 comparaţii.

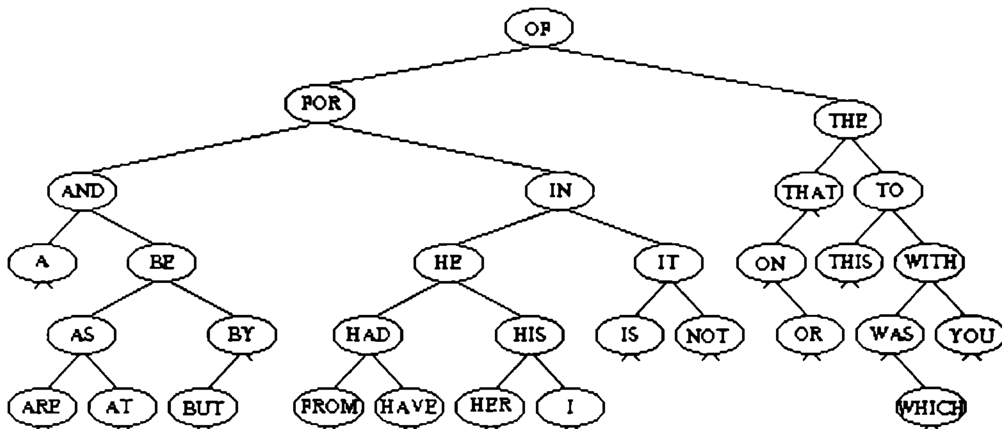


Fig. 7.8.

De multe ori trebuie considerat şi cazul căutării fără succes. De exemplu, cele 31 de cuvinte din figura 7.8. reprezintă circa 36% dintr-un text obişnuit în limba engleză. Evident, restul de 64% pot influenţa stuctura arborelui optim de căutare.

Pentru înregistrările R_1, R_2, ..., R_n având cheile K_1, K_2, ..., K_n vom considera probabilităţile p_1, p_2, ..., p_n şi q_0, q_1, q_2, ..., q_n, unde p_i este probabilitatea cu care este

https://biblioteca-digitala.ro / https://unibuc.ro

căutată înregistrarea cu cheia K_i şi q_i este probabilitatea de a fi căutată o valoare cuprinsă între K_i şi K_{i+1}. Se consideră că q_0 este probabilitatea de a căuta o valoare mai mică decât K_1 şi q_n este probabilitatea de a căuta o valoare mai mare decât K_n. În locul probabilităților se pot considera şi frecvențele de apariție ale valorilor. Obiectivul este de a determina un arbore binar de căutare care să minimizeze numărul sperat de comparații în timpul căutării, numit *costul căutării*, adică valoarea dată de expresia:

$$\sum_{i=1}^{n} p_i \; nivel(i) + \sum_{k=0}^{n} q_k \; (nivel(k) - 1).$$

Construirea arborelui binar optim de căutare se bazează pe proprietatea că orice subarbore al unui arbore binar optim de căutare este un arbore binar optim pentru căutare. Acest principiu sugerează o procedură de calcul care va găsi în mod sistematic subarbori optimi din ce în ce mai mari până când se determină un arbore care conține toate elementele.

Fie $c(i,j)$ costul unui subarbore optim cu ponderile $(p_{i+1},\dots,p_j;q_i,\dots,q_j)$ şi suma ponderilor $w(i,j)=p_{i+1}+\dots+p_j+q_i+\dots+q_j$. Acestea sunt definite pentru $0\le i\le j\le n$. Rezultă:

$$c(i, i) = 0, \text{ pentru } i = 0, 1, \dots, n,$$

$$c(i, j) = w(i, j) + \min \sum_{i\le k<j} \big(c(i, k-1) + c(k, j)\big), \text{ pentru } 0 \le i < j \le n,$$

deoarece costul minim posibil al unui arbore cu rădăcina k este $w(i,j)+c(i,k-1)+c(k,j)$. Vom nota cu $r(i, j)$ acea valoare a lui k pentru care se atinge minimul în formula anterioară; dacă sunt mai multe se alege una dintre ele. Acestea sunt posibilele rădăcini din arborele optimal. Evaluarea valorilor $c(i,j)$ se face pentru j-. $1,2,\dots,n$. Sunt aproximativ $n^2 / 2$ astfel de valori şi pentru determinarea lor sunt necesare circa $n^3 / 6$ valori ale lui k. Deci complexitatea algoritmului este $O(n^3)$ în timp şi $O(n^2)$ în spațiu. Cum se poate demonstra că:

$$r(i, j - 1) \le r(i, j) \le r(i + 1, j),$$

se poate reduce mulțimea valorilor posibile pentru k la intervalul $[r(i, j-1), r(i+1, j)]$. Algoritmul de construire a arborelui binar optimal este următorul.

Algoritmul 7.11. (Arbore optim de căutare)

1. [Inițializări] Se face $c(i, i) = 0$, $w(i, i) = q_i$ şi $w(i, j) = w(i, j - 1) + p_j + q_j$, pentru $i = 0, 1, \dots, n$ şi $j = i + 1, \dots, n$. Apoi se face $c(j - 1, j) = w(j - 1, j)$ şi $r(j - 1, j) = j$, pentru $j = 1, 2, \dots, n$.
2. [Ciclare după d] Se execută pasul 3 pentru $d = 2, 3, \dots, n$, apoi STOP.
3. [Ciclare după j] Se execută pasul 4 pentru $j = d, d + 1, \dots, n$.
4. [Calcul c şi r] Se face $i=j-d$, $c(i,j)=w(i,j)+\min_{r(i,j-1)\le k\le r(i+1,j)}(c(i,k-1)+c(k,j))$ şi se stabileşte $r(i, j)$ egal cu un k pentru care se realizează minimul precedent. ■

https://biblioteca-digitala.ro / https://unibuc.ro

În urma aplicării algoritmului precedent, se poate reconstitui arborele optimal. Să notăm cu $t(i,j)$ subarborele care conține ponderile $(p_{i+1},...,p_j;q_i,...,q_j)$. Arborele optim este $t(0,n)$. Dacă $i=j$, subarborele $t(i,j)$ conține numai un nod extern (subarbore vid) de pondere q_i. Altfel subarborele $t(i,j)$ are rădăcina $r(i,j)$, subarborele din stânga este $t(i,r(i,j)-1)$ și subarborele din dreapta este $t(r(i,j),j)$.

Deoarece algoritmul 7.11. necesită timp de execuție și spațiu de memorare proporționale cu n^2, devine nepractică utilizarea lui când n este foarte mare. În acest caz se pot aplica diferite euristici cum sunt și cele care urmează.

- Inserarea elementelor în arborele binar în ordinea descrescătoare a fr. vențeloı asociate elementelor. De foarte multe ori această strategie produce un arbore destul de avantajos. Totuși el poate fi diferit de arborele optimal în primul rând pentru că nu ține seama de probabililtățile nodurilor externe.
- Alegerea rădăcinii în așa mod încât subarborele din stânga ei să aibă o pondere cât mai apropiată de cea a subarborelui din dreapta. Acest procedeu poate să nu fie eficient dacă probabilitatea asociată rădăcinii este destul de mică.
- Se combină cele două metode anterioare. Se încearcă să se egalizeze ponderile din subarborele din stânga și cel din dreapta însă se acceptă posibilitatea de a deplasa rădăcina cu câteva poziții la stânga sau la dreapta pentru a găsi un element cu probabilitatea suficient de mare. Când se ajunge la un subarbore cu un număr suficient de mic de noduri, se aplică algoritmul 7.11.

7.9. Euristici în jocul de șah

Necesitatea utilizării unor euristici este exemplificată sugestiv de jocul de șah. În acest joc, un algoritm exact poate să dea răspuns la următoarele întrebări:

- *Problema 1*: Poate să câștige jucătorul care joacă cu piesele albe indiferent de strategia utilizată de adversar?
- *Problema 2*: Poate să câștige jucătorul care joacă cu piesele negre indiferent de strategia utilizată de adversar?
- *Problema 3*: Poate să câștige sau cel puțin să remizeze un jucător indiferent de strategia adversarului?

Evident că, dacă s-ar cunoaște răspunsul la aceste întrebări, jocul de șah ar pierde mult din interesul fanilor lui. Algoritmi care să rezolve problemele anterioare există. Vom prezenta și noi aici unele variante. Acești algoritmi se bazează pe existența unui arbore finit care să dea toate posibilitățile de desfășurare ale unei partide de șah. Nodurile arborelui sunt etichetate cu poziții posibile de pe tabla de șah.

https://biblioteca-digitala.ro / https://unibuc.ro

Rădăcina arborelui corespunde poziției de începere a partidei. Se consideră că rădăcina are nivelul 1. Pentru fiecare nod a de pe un nivel de ordin impar, se consideră succesoare directe toate nodurile ce corespund la poziții de pe tabla de şah ce se obțin din poziția asociată lui a prin efecutarea unei mutări de către jucătorul care joacă cu piesele albe. Analog, pentru fiecare nod a de pe un nivel de ordin par, se consideră succesoare directe toate nodurile ce corespund la poziții de pe tabla de şah ce se obțin din poziția asociată lui a prin efecutarea unei mutări de către jucătorul care joacă cu piesele negre.

Arborele construit după modelul anterior este finit după cum vom justifica în continuare. Numărul configurațiilor posibile pe tabla de şah este finit, o limitare superioară a numărului lor fiind de exemplu 13^{64}, deoarece pe oricare din cele 64 de pătrățele ale tablei de şah există cel mult 13 posibilități, să fie liber sau să se afle pe el una din cele 12 tipuri de piese (câte 6 pentru fiecare culoare). Numărul de nivele este finit deoarece într-o partidă de şah, o poziție nu se poate repeta decât de un număr finit de ori (la a treia repetare a unei poziții partida se termină remiză). Fiecare nod are un număr finit de succesori direcți deoarece numărul de mutări posibile ale fiecărui jucător pentru o poziție dată este limitat (de exemplu cele mai multe posibilități de mutare le poate avea dama care poate fi mutată în cel mult 27 moduri).

Răspunsul la prima problemă poate fi dat de următorul algoritm.

Algoritmul 7.12. (***Câştigă albul***)

1. [Terminare] Dacă nu mai există noduri terminale de nivel par şi care să nu corespundă la poziții de remiză, atunci STOP. Dacă toate nodurile au fost eliminate, atunci albul poate să câştige întotdeauna.
2. [Alegere] Se alege un nod terminal a care nu corespunde la o poziție de remiză şi este de nivel par.
3. [Eliminare] Se elimină toate nodurile ce aparțin subarborelui cu rădăcina tatăl nodului a.
4. [Ciclare] Se trece la pasul 1. ■

Deoarece arborele este finit şi la fiecare ciclare sunt eliminate cel puțin două noduri (nodul ales şi tatăl său) algoritmul se termină după un număr finit de paşi.

Un algoritm analog poate da răspunsul la cea de-a doua problemă.

Algoritmul 7.13. (***Câştigă negrul***)

1. [Terminare] Dacă nu mai există noduri terminale de nivel impar şi care să nu corespundă la poziții de remiză, atunci STOP. Dacă toate nodurile în afară de rădăcina arborelui au fost eliminate, atunci negrul poate să câştige întotdeauna.

https://biblioteca-digitala.ro / https://unibuc.ro

2. [Alegere] Se alege un nod terminal a care nu corespunde la o poziţie de remiză şi este de nivel impar.
3. [Eliminare] Se elimină toate nodurile ce aparţin subarborelui cu rădăcina tatăl nodului a.
4. [Ciclare] Se trece la pasul 1. ■

Deoarece arborele este finit şi la fiecare ciclare sunt eliminate cel puţin două noduri (nodul ales şi tatăl său) algoritmul se termină după un număr finit de paşi.

Pentru rezolvarea celei de-a treia probleme se poate aplica algoritmul:

Algoritmul 7.14. (Strategia optimă comună)

1. [Iniţializare] Se marchează cu 1 poziţiile în care câştigă albul (negrul este mat), cu 2 poziţiile în care câştigă negrul (albul este mat) şi cu 3 poziţiile de remiză; restul nodurilor se consideră iniţial nemarcate.
2. [Terminare] Dacă rădăcina a fost marcată, atunci STOP. Dacă marcajul rădăcinii este 1, atunci poate câştiga mereu albul, dacă este 2, atunci poate câştiga tot timpul negru şi dacă este 3, cea mai bună strategie este jocul la remiză pentru ambii parteneri.
3. [Alegere] Se alege un nod a nemarcat şi care are toţi succesorii direcţi marcaţi.
4. [Marcare] Dacă a este de nivel impar, atunci se marchează cu 1 dacă există cel puţin un succesor direct marcat cu 1, se marchează cu 2 dacă toţi succesorii direcţi sunt marcaţi cu 2, respectiv se marchează cu 3 dacă nu sunt îndeplinite condiţiile anterioare. Analog, dacă a este de nivel par, atunci se marchează cu 2 dacă există cel puţin un succesor direct marcat cu 2, se marchează cu 1 dacă toţi succesorii direcţi sunt marcaţi cu 1, respectiv se marchează cu 3 dacă nu sunt îndeplinite condiţiile anterioare.
5. [Ciclare] Se trece la pasul 1. ■

Deoarece arborele este finit şi la fiecare ciclare se marchează cel puţin un nod, algoritmul se termină după un număr finit de paşi.

Deşi nu sunt complicaţi, aceşti algoritmi nu au putut să fie executaţi din cauza numărului extrordinar de mare de noduri din arborele descris. De aceea se joacă în continuare şah, aplicându-se diferitele euristici care dau posibilitatea obţinerii unor poziţii mai avantajoase, cu şanse de câştig. Au fost dezvoltate teorii ale deschiderilor în şah şi ale finalurilor de partide dar şi tactici sau strategii ale jocului de mijloc de partidă. Euristicile folosite se bazează pe evaluarea pieselor, pe evaluarea posibilităţilor de mişcare şi pe alte funcţii care pot să evalueze situaţiile create în timpul unei partide. Dar de cele mai multe ori soarta unei partide este decisă de inspiraţia de moment şi de condiţii psihologice.

https://biblioteca-digitala.ro / https://unibuc.ro

BIBLIOGRAFIE

[AHO74] Aho A.V., Hopcroft J.E., Ullman J.D. - *The Design and Analysis of Computer Algorithms* - Addison-Wesley, Reading, Mass., 1974.

[ALE76] Alexandrescu I., Drăguţ C., Iambor I.P., Popescu T., Preoteasa P., Rogai E., Sfichi R., Tomescu I. - *Culegere de probleme de matematici aplicate* - Societatea de Ştiinţe Matematice din RSR, 1976.

[ASH71] Ashour S., Parker R.G. - *A Precedence Graph Algorithm for the Shop Scheduling Problem* - Operational Research Quarterly, Vol. 22, Nr. 2, June 1971, 165-176.

[BAK69] Bakshi M.K., Arora S.R. - *The segmendary problem* - Mgmt Sci, Vol. 16, Nr. 4, 1969, B247-263.

[BÂS87] Bâscă O.C., Popescu I. - *Sisteme de operare - Volumul I* - Tipografia Universităţii din Bucureşti, 1987.

[BÂS89] Bâscă O.C., Popescu I. - *Sisteme de programe pentru minicalculatoare (Sisteme de operare - Volumul II)* - Tipografia Universităţii Bucureşti, 1989.

[BEA90] Beasley J.E. - *A Lagrangian Heuristic for Set-Covering Problems* - Naval Research Logistics, Vol. 37, Nr. 1, February 1990, 151-164.

[BER69] Berge C. - *Teoria grafurilor şi aplicaţiile ei* - Editura Tehnică, 1969.

[BOR75] Borodin A., Munro I. - *The Computational Complexity of Algebraic and Numeric Problems* - American Elsevier Publishing Company, Inc., 1975.

[BRA96] Brasard G., Bratley P. - *Fundamentals of Algorithmics* - Prentice Hall, Inc., 1996, 480-482.

[BUC73] Bucur C.M. - *Metode numerice* - Editura Facla, Timişoara, 1973.

[BUR84] Burstein M.C., Nevison C.H., Carlson R.C. - *Dynamic Lot-Sizing When Demand Timing is Uncertain* - Oper. Res., Vol. 32, 1984, 362-379.

[CAR88] Carlier J., Chrétienne P. - *Problèmes d'ordonnancement - Modélisation, Complexité, Algorithmes* - Masson, Paris, Milan, Barcelone, Mexico, 1988.

[CAR80] Carpaneto G., Toth P. - *Branching and Bounding Criteria for the Asymmetric Travelling Salesman Problem* - Mgmt. Sci., Vol. 26, 1980, 736-743.

https://biblioteca-digitala.ro / https://unibuc.ro

[CAR84] Carpaneto G., Martello S., Toth P. - *An Algorithm for the Bottleneck Traveling Salesman Problem* - Oper. Res., Vol. 32, Nr. 2, March-April 1984, 380-389.

[CHA90] Chang Shiwei, Matsuo Hirofumi, Tang Guochun - *Worst-Case Analysis of Local Search Heuristics for the One-Machine Total Tardiness Problem* - Naval Research Logistics, Vol. 37, Nr. 1, february 1990, 111-122.

[CHO78] Choukhame E.A. - *Une heuristique pour le problème de l'arbre de Steiner* - RAIRO, Recherche Operationelle, Vol. 12, Nr. 2, mai 1978, 207-212.

[CHR71] Christofides N., Viola P. - *The Optimum Location of Multi-centres on a Graph* - Operational Research Quarterly, Vol. 22, Nr. 2, June 1971, 145-154.

[CHV79] Chvátal V. - *A Greedy Heuristic for the Set-Covering Problem* - Mathematics of Operations Research, Vol. 4, Nr. 3, 1979, 233-235.

[COR77] Cornuejols G., Fisher M.L., Nehmauser G.L. - *Location of Bank Accounts to Oprimize Float: An Analytic Study of Exact and Approximate Algorithms* - Management Science, Vol. 23, Nr. 8, April 1977, 789-810.

[COU69] Courant R., Robbins H. - *Ce este matematica?* - , Editura Ştiinţifică, Bucureşti, 1969.

[CRA82] Craiu V., Bâscă O. - *Elemente de calcul numeric şi programare* - Ediţia a II-a revizuită şi îmbunătăţită, Tipografia Universităţii din Bucureşti, 1982.

[DAN90] Dantzig G.B., Dempster M.A.H., Kallio M. (Ed.) - *Programarea liniară a sistemelor mari* - Editura Tehnică, 1990.

[DAV81] Davis E., Jaffe J.M. - *Algorithm for Scheduling Tasks on Unrelated Processors* - J. Ass. C. Mach., Vol. 28, Nr. 4. Oct. 1981, 721-736.

[DEM81] Demidovich B.P., Maron I.A. - *Computational Mathemati* - MIR Publishers, Moscow, 1981.

[DEM83] Dempster M.A.H., Fisher M.L., Jansen L., Lageweg B.J., Lenstra J.K., Rinnoy Kan A.H.G. - *Analysis of Heuristics for Stochastic Programming: Results for Hierarchical Scheduling Problems* - Math. Oper. Res., Vol. 8, Nr. 4, Nov. 1983, 525-537.

[DOR76] Dorn W.S., McCracken D.D. - *Metode numerice cu programe în FORTRAN IV* - Editura Tehnică, 1976.

[DRA72] Dragomirescu M., Maliţa M. - *Programare neliniară* - Editura Ştiinţifică, 1972.

[DUD64] Dudek R.A., Teuton O.F. - *Development of M-stage decision rule for scheduling n jobs through m machines* - Operational Research, Vol. 12, Nr. 3, 1964, 471-497.

https://biblioteca-digitala.ro / https://unibuc.ro

[EAD82] Eades P., Foulds L.R., Giffin J.W. - *An Efficient Heuristic for Identifing a Maximum Weight Planar Subgraph* - in Lectures Notes in Mathematics, Nr. 952, *Combinatorial Mathematics IX*, Springer-Verlag, Berlin, 1982.

[FIS80] Fisher M.L. - *Worst Case Analysis of Heuristic Algorithms* - Management Sci., Vol. 26, Nr. 1, Jan. 1980, 1-17.

[FLO62] Floyd R.W. - *Algorithm 97: Shortest Path* - Comm.ACM, Vol. 5, Nr. 6, 1962, 345.

[FOR62] Ford L.R.Jr., Fulkerson D.R. - *Flows in Networks* - Princeton Univ. Press, Princeton, NJ, 1962.

[FOU85] Foulds L.R., Gibbons P.B., Giffin J.W. - *Facilities Layout Adjacency Determination: An Experimental Comparison of Three Graph Theoretic Heuristics* - Operations Research, Vol. 33, Nr. 5, Sept.-Oct. 1985, 1091-1106.

[FRI83] Frieze A. M. - *An Extension of Christofides Heuristic to the k-Person Travelling Salesman Problem* - Discrete Applied Mathematics, Vol. 6, Nr. 1, May 1983, 79-84.

[GAR79] Garey M., Johnson D. - *Computers and Intractability* - W.H.Freeman, San Francisco, 1979.

[GAV65] Gavett W.J. - *Three Heuristic Rules for Sequencing Jobs to a Single Production Facility* - Management Science, Vol. 11, Nr. 8, June 1965, B166-176.

[GEO74] Geoffrion A.M. - *Lagrangian Relaxation for Integer Programming* - Math. Prog. Study, Vol. 2, 1974, 82-114.

[GIL85] Gilmore P.C., Lawler E.L., Shmoys D.B. - *Well-Solved Special Case of the Traveling Salesman Problem* - Chap.4 în *The Traveling Salesman Problem*, Lawler E.L., Lenstra J.K., Rinnoy Kan A.M.G., Shmoys D.B. (editori), John Wiley & Sons, New York, 1985.

[GLO75] Glover F., Klingman D. - *Finding Minimum Spanning Trees with a Fixed Number of Links at a Node* - în *Combinatorial Programming Methods and Applications*, D. Reidel, Dordrecht, 1975, 191-201.

[GOL87] Golden B.C., Levy L., Vohra R. - *The Orienteering Problem* - Naval Research Logistics, Vol. 34, Nr. 3, June 1987, 307-318.

[GOL93] Goldberg A.V., Radzik T. - *A Heuristic Improvement of the Bellman-Ford Algorithm* - Computer Science Department, Stanford Univ., California, Report, 1993,1-5.

[GOO77] Goodman S.E., Hedetniemi S.T. - *Introduction to the Design and Analysis of Algorithms* - McGraw-Hill Book Company, 1977.

https://biblioteca-digitala.ro / https://unibuc.ro

[GRA69] Graham R.L. - *Bounds on Multiprocessing Timing Anomalies* - SIAM Jr. on Appl. Math., Vol. 17, Nr. 2, March 1969, 416-429.

[GRA90] Grassberger P., Freund H. - *An Efficient Heuristic Algorithm for Minimum Matching* - Zeitscrift für Operations Research, Vol. 34, Issue 4, 1990, 239-254.

[GUP71] Gupta Jatinder N.D. - *A Functional Heuristic Algorithm for the Flowshop Scheduling Problem* - Operational Research Quarterly, Vol. 22, Nr. 1, March 1971, 39-47.

[GUP76] Gupta Jatinder N.D. - *A Heuristic Algorithm for the Flowshop Scheduling Problem* - RAIRO, Vol. 10, Nr. 6, Juin 1976, 63-73.

[HAC89] Hackman S.T., Magazine M.J., Wee T.S. - *Fast, Effective Algorithms for Simple Assembly Line Balancing Problems* - Operations Research, Vol. 37, Nr. 6, Nov.-Dec. 1989, 916-924.

[HAD62] Hadley G. - *Linear Programming* - Addison-Wesley, Reading, M/ 1962.

[HAI85] Haimovich M., Rinnoy Kan A.M.G. - *Bounds and Heuristics for Capacited Routing Problem* - Math. Oper. Res., Vol. 10, Nr. 4, 1985, 527-542.

[HAN66] Hanan M. - *On Steiner's Problem with Rectilinear Distance* - SIAM Journal on Applied Mathematics, Vol. 14, Nr. 2, 1966, 255-265.

[HIL69] Hillier F.S. - *Efficient Heuristic Procedures for Integer Linear Programing with an Interior* - Operations Research, Vol. 17, Nr. 4, July-August 1969, 600-637.

[HOC82] Hochbaum D.S. - *Heuristics for the Fixed Cost Median Problem* - Mathematical Programming, Vol. 22, Nr. 2, Febr. 1982, 148-162.

[HOP74] Hopcroft J., Tarjan R. - *Efficient Planarity Testing* - J. Assoc. Comput. Mach., Vol. 21, Nr. 4, 1974, 549-568.

[HOR78] Horowitz E., Sahni S. - *Fundamentals of Computer Algorithms* - Computer Science Press, Potomac, MD, 1978.

[HSU84] Hsu When-Lian - *Approximation Algorithms for the Assembly ˏ ıe Crew Scheduling Problem* - Math. Of Oper. Res., Vol. 9, Nr. 3, Aug. 1984, 376-383.

[HUY80] Huyn N., Dechter R. Pearl J. - *Propabilistic Analysis of the Complexity of A** - Artificial Intelligence, Vol. 15, 1980, 241-254.

[IBA75] Ibarra O.H., Kim C.E. - *Fast Approximation Algorithms for the Knapsack and Sum of Subsets Problems* - J. ACM, Vol. 22, 1975, 463-468.

[IBA77] Ibarra O.H., Kim C.E. - *Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors* - J. ACM, Vol. 24, Nr. 2, April 1977, 280-289.

https://biblioteca-digitala.ro / https://unibuc.ro

[IBA83] Ibaraki T., Muro S., Murakami T., Hasegawa T. - *Using Branch-and-Bound Algorithms to obtain Suboptimal Solutions* - Zeitscrift für Operations Research, Band 27, Heft 5, 1983, A177-202.

[JOH54] Johnson S.M. - *Optimal two-and three-stage production schedules with setup times included* - Naval Research Logistics Quarterly, 1, 1954, 61-68.

[JOH74] Johnson D.S. - *Approximation Algorithms for Combinatorial Problems* - Journal of Computer and System Sciences, Vol. 9, Nr. 3, Dec. 1974, 256-278.

[JOH74a] Johnson D.S., Demers A., Ullman J.D., Garey M.R., Graham R.L. - *Worst-case Performance Bounds for simple one-dimensional Packing Algorithms* - SIAM J. Comput., Nr. 3, 1974, 299-325.

[JON90] Joneja Dev - *The Joint Replenishment Problem: New Heuristics and Worst Case Performance Bounds* - Operations Research, Vol. 38, Nr. 4, July-August 1990, 711-723.

[KAR64] Karg R.R., Thompson G.L. - *A Heuristic Approach to Solving Travelling Salesman Problems* - Management Science, Vol. 10, Nr. 2, January 1964, 225-248.

[KAR77] Karp R. - *Probabilistic Analysis of Partitioning Algorithms for the Travelling Salesman Problems in the Plane* - Math. of Oper. Res., Vol. 2, Nr. 3, 1977, 209-224.

[KHA79] Khachian L.G. - *A Polinomial Time Algorithm in Linear Programming* - Soviet. Math. Dokl., Vol. 20, 1979, 191-194.

[KHU72] Khumawala Basheer M. - *An Efficient Branch and Bound Algorithm Warehouse Location* - Mgm. Sci., Vol. 18, Nr. 12, Aug. 1972, 718-731.

[KHU73] Khumawala Basheer M. - *An Efficient Heuristic Procedure for the Uncapacitated Warehouse Location Problem* - NRLQ, Vol. 20, Nr. 1, March 1973, 109-122.

[KNU76] Knuth D.E. - *Tratat de programarea calculatoarelor - Sortare şi căutare* - Editura Tehnică, Bucureşti, 1976.

[LAN70] Lange O. - *Decizii Optimale: Bazele Programării* - Editura Ştiinţifică, 1970.

[LAW76] Lawler E.L. - *Combinatorial Optimisation: Networks and Matroids* - Holt, Rinehart&Winston, New York, 1976.

[LAZ68] Lazăr S. - *Analiza drumului critic* - Editura Ştiinţifică, 1968.

[LIN73] Lin S., Kernighan B.W. - *An Effective Heuristic Algorithm for Traveling Salesman Problem* - Operations Research, Vol. 21, Nr. 2, March-April 1973, 498-516.

https://biblioteca-digitala.ro / https://unibuc.ro

[LIV85] Livovschi L., Georgescu H. - *Analiza şi Sinteza Algoritmilor* - Editura Ştiinţifică şi Enciclopedică, 1985.

[LUS86] Luss Hanan - *A Heuristic for Capacity Expansion Planning with Multiple Facility Types* - Naval Research Logistic Quarterly, Vol. 33, Nr. 4, Nov. 1986, 685-701.

[MAH75] McMahon G., Florian M. - *On Scheduling with Ready Times and Due Dates to Minimize Maximum Lateness* - Op. Res., Vol. 23, 1975, 475-482.

[MAR87] Marinescu Gh., Rizzoli I., Popescu I., Ştefan C. - *Probleme de analiză numerică rezolvate cu calculatorul* - Editura Academiei RSR, 1987.

[MAR83] Martello S. - *An Enumeration Algorithm for Finding Hamiltonian Circuits in a Directed Graph* - ACM Trans. on Math Software, Nr. 9, 1983, 131-138.

[MĂC72] Măcriş A., Dumitru V. - *Aplicaţii ale cercetării operaţionale în probleme de conducere, organizare şi planificare a lucrărilor de investiţii, construcţii şi montaj* - Editura Academiei RSR, 1972.

[MIC94] Michalewicz Z. - *Genetic Algorithms + Data Structures = Evolution Programs* - Springer-Verlag, 1994.

[MIH82] Mihu C., Dăneţ T. - *Probleme pentru aplicarea matematicii în practică* - Editura Didactică şi Pedagogică, 1982.

[MIL77] Milgram M., Dubuisson B. - *Un Algorithme Heuristique de Décomposition d'un Graphe* - RAIRO, Recherche Opérationelle, Vol.11, Nr. 2, Mai 1977, 175-199.

[MOR81] Moran S. - *General Approximation Algorithms for Some Arithmetical Combinatorial Problems* - Theoretical Computer Science, Vol. 14, Nr. 3, June 1981, 289-303.

[MÜL76] Müller-Merbach H. - *Darmstadt-Morphologie heuristicher Verfahren* - Zeitscrift für Operations Research, Band 20, Heft 3, Juni 1976, 69-87.

[MUN69] Muntz R.R., Coffman E.G.Jr. - *Optimal Preemptive Scheduling on Two-Processor Systems* - IEEE Trans. Comp. C-18, 11, 1969.

[MUN70] Muntz R.R., Coffman E.G.Jr. - *Preemptive Scheduling of Real Time Tasks on Multiprocessor Systems* - J. Assoc. Comput. Mach., 17, 2, Apr. 1970, 324-338.

[NEV85] Nevison C.H. - *A Cost Adjustment Heuristic for Dynamic Lot-Sizing with Uncertain Demand Timing* - Operations Research, Vol. 33, Nr. 6, Nov.-Dec. 1985, 1342-1352.

[NIC71] Nicholson T.A.J., Pullen R.D. - *Dynamic Programming Applied to Ship Fleet Management* - Operational Research Quarterly, Vol. 22, Nr. 3, 1971, 211-220.

https://biblioteca-digitala.ro / https://unibuc.ro

[NIL82] Nilsson N.J. - *Principles of Artificial Intelligence* - Springer-Verlag, Berlin Heidelberg New York, 1982.

[NIS83] Nishizeki T., Asano T., Watanabe T. - *An Approximation Algorithm for the Hamiltonian Walk Problem on Maximal Planar Graphs* - Discrete Applied Mathematics, Vol. 5, Nr. 2, February 1983, 211-222.

[PAG61] Page E.S. - *An approach to the scheduling of jobs on machines* - J. R. Statist. Soc., 23, Seria B, 1961, 484-492.

[PAI88] Pair C., Mohr R., Schott R. - *Construire les algorithmes - les améliorer, les connaître, les évaluer* - Dunod Informatique, Paris, 1988.

[PAL65] Palmer D.S. - *Sequencing Jobs through a Multi-Stage Process in the Minimum Total Time - A Quick Method of Obtaining a Near Optimum* - Operational Research Quarterly, Vol. 16, Nr. 1, March 1965, 101-107.

[PEA84] Pearl J. - *Heuristics, Intelligent Search Strategies for Computer Problem Solving* - Addison-Wesley Publishing Company, 1984.

[PET88] Peters J.G. - *Lower Bounds on Time-Accuracy Trade-offs for the 0-1 Knapsack Problem* - Math. Of Oper. Res., Vol. 13, Nr. 3, Aug. 1988, 497-507.

[PLA87] Plante R.D., Lowe T.J., Chandrasekaran R. - *The product matrix traveling salesman problem: an application and solution heuristic* - Operations Research, Vol. 35, Nr. 5, Sept-Oct 1987, 772-783.

[PLA90] Plaisted D.A. - *A Heuristic Algorithm for Small Separators in Arbitrary Graphs* - SIAM Journal on Computing, Vol. 19, Nr. 2, April 1990, 267-280.

[POS76] Posa L. - *Hamiltonian Circuits in Random Graphs* - Discrete Mathematics, Vol. 14, 1976, 359-364.

[POS88] Posner M.E. - *The Deadline Constrained Weighted Completion Time Problem: Analysis of a Heuristic* - Operations Research, Vol. 36, Nr. 5, Sept.-Oct. 1988, 742-746.

[POT80] Potts C.N. - *Analysis of a Heuristics for One Machine Sequencing with Release Dates and Delivery Times* - Oper. Res., Vol. 28, Nr. 6, 1980, 1436-1441.

[POT85] Potts C.N. - *Analysis of a Linear Programming Heuristic for Scheduling Unrelated Parallel Machines* - Discrete Applied Mathematics, Vol. 10, Nr. 2, 1985, 155-164.

[POT85a] Potts C.N. - *Analysis of Heuristics for Two-Machine Flow-Shop Sequencing Subject to Release Dates* - Math. Oper. Res., Vol. 10, nr. 4, 1985, 576-584.

https://biblioteca-digitala.ro / https://unibuc.ro

[PUR82] Purcaru I. - *Elemente de algebră şi programare liniară* - Editura Ştiinţifică şi Enciclopedică, 1982.

[PUR97] Purcaru I. - *Matematici generale & elemente de optimizare. Teorie şi aplicaţii* - Editura Economică, 1997.

[RAL76] Ralston A. (Ed.), Meek C.L. (Ass.Ed.) - *Encyclopedia of Computer Science* - Petrocelli/Charter, New York, 1976.

[REI77] Reingold E.M., Nievergelt J., Deo N. - *Combinatorial Algorithms. Theory and Practice* - Prentice-Hall, Inc., 1977.

[RIC89] Richards D. - *Fast Heuristic Algorithms for Rectilinear Steiner Trees* - Algorithmica, Vol. 4, Nr. 2, 1989, 191-208.

[ROM71] Roman R.J. - *Mine-mill Production Scheduling by Dynamic Programming* - Operational Research Quarterly, Vol. 22, Nr. 4, December 1971, 319-328.

[ROS92] Rosenberg O., Ziegler H. - *A Comparison of Heuristic Algorithms for Cost-Oriented Assembly Line Balancing* - ZOR, Nr. 6, 1992, 477-495.

[ROS77] Rosenkrantz D.J., Stearns R.E., Lewis P.M. - *An Analysis of several Heuristics for the Travelling Salesman Problem* - SIAM J. Comput. 6, 1977, 568-581.

[ROT93] Rote G., Vogel A. - *A Heuristic for Decomposing Traffic Matrices in TDMA Satellite Communication* - ZOR, Vol. 38, Nr. 3, 1993, 281-307.

[SAH75] Sahni S. - *Approximate Algorithms for the 0/1 Knapsack Problem* Journal of the Assoc. For Computing Machinery, Vol. 22, Nr. 1, January 1975, 115-124.

[SAH77] Sahni S. - *General Techniques for Combinatorial Approximation* - Oper. Res., Vol. 25, Nr. 6, 1977, 920-936.

[SCH71] Schrage L. -*Obtaining Optimal Solutions to Resource Constrained Network Scheduling Problems* - manuscris nepublicat, martie 1971.

[SEN68] Senju Shizuo, Toyoda Yoshiaki - *An Approach to Linear Programming with 0-1 Variables* - Management Science, Vol. 15, Nr. 4, December 1968, B196-207.

[SET90] Sethi S.P., Sriskandarajah C., Tayi Giri Kumar, Rao M.R. - *Heuristic Methods for Selection and Ordering of Part-Orienting Devices* - Oper. Res., Vol. 38, Nr. 1, Ian.-Febr. 1990, 84-98.

[SIM88] Simchi-Levi D., Berman O. - *A Heuristic for the Traveling Salesman Location Problems on Networks* - Operations Research, Vol. [illegible], Nr. 3, 1988, 478-484.

https://biblioteca-digitala.ro / https://unibuc.ro

[SPÄ85] Späth H. - *Heuristically Determining Cliques of Given Cardinality and with Minimal Cost within Weighted Complete Graphs* - Zeitschrift für Operations Research, Vol. 29, Issue 3, June 1985, 125-131.

[STA81] Stanciu P., Criveanu D., David Gh., Fuchs W. - *Matematici aplicate în economie* - Editura Facla, 1981.

[STE61] Steinhaus H. - *Caleidoscop matematic* - Editura Tehnică, Bucureşti, 1961.

[SUB93] Subhlok J. - *Solving Integer Programs from Dependence and Synchronization Problems* - Carnegie Mellon University F·:tsburgh. CMU-CS-93-130, March 1993.

[SZW87] Szwarc W., Gupta J.N.D. - *A flow-shop problem with sequence dependent additive setup times* - Naval Research Logistics Quarterly, Vol. 34, 1987, 619-627.

[SZW89] Szwarc W., Liu J.J. - *An Approximate Solution of the Flow-Shop Problem with Sequence Dependent Setup Times* - Zeitschrift für Operations Research, Vol. 33, Issue 6, 1989, 439-451.

[TEO72] Teodorescu N., Boldur Gh., Stoica M., Stancu-Minasian I., Băncilă I. - *Metode ale cercetării operaţionale în gestiunea întreprinderilor* - Editura Tehnică, 1972.

[TOM78] Tomescu I. - *Combinatorică şi teoria grafurilor* - Tipografia Universităţii din Bucureşti, 1978.

[TOY75] Toyoda Yoshiaki - *A Simplified Algorithm for Obtaining Approximate Solutions to Zero-One Programming Problems* - Managemen Science, Vol. 21, Nr. 12, Aug. 1975, 1417-1427.

[TSI84] Tsiligirides T. - *Heuristic Methodes Applied to Orienteering* - J. of Op. Res. Soc., Vol. 35, Nr. 9, 1984, 797-809.

[TUR89] Turner J. S. - *Approximation Algorithms for the Shortest Common Superstring Problem* - Information and Control, Vol. 83, Nr. 1, October 1989, 1-20.

[VĂD74] Văduva I., Dinescu C., Săvulescu B. - *Modele matematice de organizarea şi conducerea producţiei* - Editura Didactică şi Pedagogică, 1974.

[VEI69] Veinott A.F. - *Minimum Concave Cost Solutions of Leontief Substitution Models of Multi-Facility Inventory Systems* - Oper. Res., Vol. 17, Nr. 2, 1969, 262-291.

[WEB71] Webb M.H.J. - *Some Methods of Producing Approximate Solutions to Travelling Salesman Problems with Hundreds or Thousands of Cities* - Operational Research Quarterly, Vol. 22, Nr. 1, March 1971, 49-6

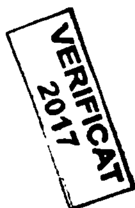
https://biblioteca-digitala.ro / https://unibuc.ro

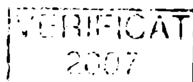
[WIN92] Winter P., MacGregor Smith J. - *Path-Distance Heuristics for the Steiner Problem in Undirected Networks* - Algorithmica, Nr. 7, 1992, 309-327.

[WRE72] Wren A., Holliday A. - *Computer Scheduling of Vehicles from One or More Depots to a Number of Delivery Points* - Op. Res. Q., Vol. 23, Nr. 3, 1972, 333-344.

[ZAN69] Zangwill W.I. - *A Backlogging Model and a Multi-echelon Model of a Dynamic Economic Lot Size Production System-A Network Approach* - Mgmt. Sci., Vol. 15, Nr. 9, 1969, 506-527.

[***80] *** - *Mică enciclopedie matematică* - Editura Tehnică, 1980.





https://biblioteca-digitala.ro / https://unibuc.ro

Tiparul s-a executat sub c-da nr. 1080/2003 la
tipografia Editurii Universității din București

https://biblioteca-digitala.ro / https://unibuc.ro

https://biblioteca-digitala.ro / https://unibuc.ro

DATA RESTITUIRII

13 MAR. 2004		
17 MAR. 2004		
17 MAR. 2004		
15 IUN. 2004		
4 FEB. 2005		
27 MAI 2005		
27 MAI 2005		
12 DEC. 2005		
10 IUN. 2014		
	30 IUN. 2009	

170.000

https://biblioteca-digitala.ro / https://unibuc.ro

OCTAVIAN BÂSCĂ – *S-a născut la 26 aprilie 1947 la Craiova. A absolvit în 1970 Facultatea de Matematică-Mecanică, secția Mașini de calcul, a Universității din București. A fost asistent universitar (1970-1977), lector universitar (1977-1991) și conferențiar universitar (din 1991) la Facultatea de Matematică a Universității București. A fost prodecan (1992-1996). În 1977 a obținut titlul de doctor în matematici în urma susținerii tezei cu titlul: Sisteme de stocare și regăsire a informației. A ținut cursuri de Mașini de calcul, Structuri de date, Sisteme de operare, Baze de date, Calcul distribuit și altele. A obținut premii la Olimpiada Internațională de Matematică (II la Moscova în 1964 și III la Berlin în 1965). Cercetări în domeniile: structuri de date, sisteme de operare, sisteme distribuite, baze de date, teoria algoritmilor. A elaborat, singur sau în colaborare, un număr de 19 monografii și manuale publicate printre altele la Editura Academiei, Editura Albatros, Editura Științifică și Enciclopedică, Editura Economică, Editura All și altele.*

LEON LIVOVSCHI – *S-a născut la 26 mai 1921, în comuna Vărzărești, județul Lăpușna. A absolvit Facultatea de electromecanică, secția Aviație, Școala Politehnică din București în 1945 și Facultatea de Fizică-Matematică a Universității din București în 1962. A fost asistent universitar (1951-1954), lector universitar (1954-1967), conferențiar universitar (1967-1971) și profesor universitar (din 1971), actualmente fiind pensionar (din 1986). În 1962 a obținut titlul de doctor în științe fizico-matematice în urma susținerii tezei cu titlul: Sinteza mecanismelor automate cu aplicații la mecanismele pneumatice și hidraulice. În 1970 a obținut titlul de doctor inginer în urma susținerii tezei cu titlul: Circuite cu contacte de relee. A ținut cursuri la I.P.G.G. și la Universitatea din București de Mecanică teoretică, Teoria elasticității, Bazele informaticii, Matematici speciale și altele. A primit premii ale M.E.Î. în anii 1956, 1960 și 1968. Cercetări în domeniile: teoria automatelor finite, metode și algoritmi de sinteză ale circuitelor combinaționale și secvențiale, teoria programării calculatoarelor, teoria algoritmilor. A elaborat, singur sau în colaborare, un număr de 17 monografii și manuale publicate printre altele la Editura Academiei, Editura Tehnică, Editura Albatros, Editura Didactică și Pedagogică, Editura Științifică și Enciclopedică și altele.*

ISBN 973-575-785-0

Lei 170000

https://biblioteca-digitala.ro / https://unibuc.ro