ACADEMIA DE STUDII ECONOMICE DIN BUCUREŞTI

Vasile Teodor NICA

CERCETĂRI OPERAȚIONALE II

Introducere în programarea în numere întregi
Introducere în optimizarea combinatorială

Note de curs pentru învățământul la distanță

Editura ASE
Bucureşti
2011

Editura ASE
Piața Romană nr. 6, sector 1, Bucureşti, România
cod 010374
www.ase.ro
www.editura.ase.ro
editura@ase.ro

Referenți:

Prof. univ. dr. Virginia MĂRĂCINE
Prof. univ. dr. Dobre ION

ISBN 978-606-505-500-1
978-606-505-501-8 Vol. II

CUPRINS

INTRODUCERE ÎN PROGRAMAREA ÎN NUMERE ÎNTREGI

Unitatea de învățare 1
Utilizarea variabilelor întregi în modelarea proceselor economice 6

1.1 Exemple de modelare cu variabile întregi 7
1.2 Terminologie 19
Probleme propuse 20
Bibliografie. 23

Unitatea de învățare 2
Particularitățile programării în numere întregi 24

2.1 Reprezentări geometrice 25
2.2 Programarea liniară vs programarea în numere întregi 27
2.3 Dificultăți în studiul şi rezolvarea programelor întregi 30
Probleme propuse 33

Unitatea de învățare 3
Metode de rezolvare exactă a programelor întregi 34

3.1 Generalități privind metodele de rezolvare a programelor întregi 35
3.2 Metoda Branch and Bound – descriere de principiu 36
3.3 Aplicarea metodei B&B. Ilustrări numerice. 40
Probleme propuse 46

INTRODUCERE ÎN OPTIMIZAREA COMBINATORIALĂ

Unitatea de învățare 4
Obiectul optimizării combinatoriale 49

4.1 Elemente de teoria grafurilor 50
4.2 Ce este optimizarea combinatorială 55
4.3 Alte probleme reprezentative ale optimizării combinatoriale... 69
Probleme propuse..... 71
Bibliografie... 73

Unitatea de învățare 5
Drumuri şi arbori.. 74

5.1 Problema drumului de valoare minimă (Shortest Path Problem) ... .75
5.1.1 Definiții şi notații.... 75
5.1.2 Algoritmul lui Dijkstra.... 76
5.1.3 Algoritmul lui Dijkstra. Ilustrare numerică.... 77
5.2 Problema arborelui de cost minim (Minimum Spanning Tree Problem).... 80
5.2.1 Algoritmul lui Kruskal. Algoritmul lui Prim.... 82
5.2.2 Versiunea „operațională" a algoritmului lui Kruskal.... 84
Probleme propuse.... 87

Unitatea de învățare 6
Problema comisvoiajorului (Traveling Salesman Problem ≡ TSP).... 91

6.1 Formularea problemei. Clasificare. Aplicații.... 92
6.2 Problema firmei de curierat rapid.... .. 93
6.3 Un algoritm de tip B&B pentru TSP în cazul asimetric.... .. 95
6.3.1 Algoritmul lui Eastman.... 96
6.3.2 Aplicație la problema firmei de curierat rapid.... 98
6.4 Euristici de rezolvare suboptimală a TSP euclidiene.... 101
6.4.1 Euristica: mergi la cel mai apropiat vecin.... 102
6.4.2 Euristica: ajustare locală.... 103
6.4.3 Euristica: inserează punctul cel mai depărtat.... 105
6.4.4 Euristica: dublează muchiile unui arbore minimal.... 106
6.4.5 Euristica lui Cristofides.... 108
Probleme propuse.... 109

Unitatea de învățare 7
Problema stabilirii traseelor (Vehicle Routing Problem ≡ VRP).... 113

7.1 Formularea problemei. Aplicații.... 114
7.2 Metode de rezolvare suboptimală a VRP.... 115
7.2.1 Euristica Cluster First – Route Second.... 115
7.2.2 Euristica Route First – Cluster Second.... 115
7.2.3 Euristica Clarke – Wright.... 116
7.3 Ilustrări numerice.... 118
7.3.1 Aplicarea euristicii Cluster First – Route Second.... 120
7.3.2 Aplicarea euristicii Route First – Cluster Second.... 122
7.3.3 Aplicarea euristicii Clarke – Wright.... 123
Probleme propuse.... 126

Unitatea de învățare 8
Problema poştaşului chinez (Chinese Postman Problem ≡ CPP). 129

8.1 Formularea problemei. Aplicații....130
8.2 Grafuri euleriene....130
8.3 Modelarea problemei poştaşului chinez....131
8.4 Soluția CPP în grafuri neorientate....133
8.4.1 Graful este eulerian....134
8.4.2 Graful nu este eulerian....135
Probleme propuse....139

Unitatea de învățare 9
Problema echilibrării liniilor de asamblare (Assembly Line Balancing Problem ≡ ALBP)....142

9.1 Problema echilibrării liniilor de asamblare....143
9.2 Reprezentarea mulțimii operațiilor....144
9.3 Modelarea ALBP....145
9.4 Reducerea ALBP la o problemă de drum minim....146
9.5 Algoritmi de rezolvare suboptimală a ALBP....150
9.6 Ilustrări numerice....156
Probleme propuse....163

Unitatea de învățare 10
Introducere în studiul problemei ordonanțării (Scheduling Problem)....164

10.1 O clasificare a problemelor de ordonanțare....165
10.2 Probleme de ordonanțare pe un utilaj....166
10.3 Ordonanțarea în flux (Flow Shop)....169
10.3.1 Ordonanțarea în flux pe două utilaje....172
10.3.2 Ordonanțarea în flux pe m ≥ 3 utilaje....174
10.4 Ordonanțarea pe mai multe utilaje, cazul general (Job Shop)....177
10.4.1 Modelarea problemei....177
10.4.2 Cazul general – o rezolvare suboptimală....182
10.4 Ilustrări numerice....186
Probleme propuse....190

Unitatea de învățare 1

PROGRAMARE ÎN NUMERE ÎNTREGI
Utilizarea variabilelor întregi în modelarea proceselor economice

---

Cuprins

1.1 Exemple de modelare cu variabile întregi
1.2 Terminologie

Probleme propuse
Bibliografie

Obiectivul unității de învățare 1

Este binecunoscut faptul că **programarea liniară** reprezintă un instrument puternic şi eficace pentru studiul proceselor economice în vederea îmbunătățirii performanțelor acestora.

O proprietate foarte importantă a variabilelor de decizie dintr-un program liniar era aceea că, o dată cu două valori **permise**, puteau lua **orice altă valoare intermediară**. Proprietatea „**de a putea varia continuu**" era esențială în fundamentarea metodelor de determinare a soluțiilor optime.

Există însă o mare varietate de situații practice, modelate cu ajutorul programării liniare, în care unele variabile de decizie nu au sens economic decât dacă au numai valori **întregi**.

Obiectivul urmărit în această secțiune este acela de a sublinia, prin exemple judicios alese, **necesitatea** şi **importanța** utilizării variabilelor întregi în modelarea economico matematică.

1.1 Exemple de modelare cu variabile întregi

Nu puține sunt situațiile practice în care apar „activități" al căror output este măsurat în unități **indivizibile**; este clar că valorile permise ale variabilei care indică nivelul unei asemenea activități trebuie să fie numere întregi. La fel, repartizarea personalului muncitor, al utilajelor sau al mijloacelor de transport pe diverse lucrări trebuie exprimată tot prin valori întregi.

Exemplul 1. Un fermier are nevoie de 107 funți de îngrăşământ pe care îl poate procura fie în saci de 35 funți la prețul de 14 $ sacul, fie în saci de 24 funți a 12 $ fiecare. Obiectivul său este de a cumpăra cantitatea necesară de îngrăşământ la cel mai mic cost.

Modelarea acestei situații este simplă. Dacă notăm cu x_1, x_2 **numărul** sacilor de 35 funți, respectiv 24 funți cumpărați de fermier,obținem imediat următorul model:

Să se determine x_1^*, x_2^* care minimizează **funcția obiectiv**:

$$f = 14x_1 + 12x_2 \quad \leftarrow \quad \text{costul sacilor cumpărați}$$

cu satisfacerea **restricției**:

$$35x_1 + 24x_2 \geq 107 \quad \leftarrow \quad \text{asigurarea cantității dorite de îngrăşământ}$$

şi a **condițiilor explicite** impuse variabilelor:

$$x_1 \geq 0\,,\; x_2 \geq 0$$
$$x_1, x_2 \textbf{ întregi}$$

În modelul rezultat, condiția „x_1, x_2 **întregi**" înseamnă pur şi simplu că fermierul nu poate cumpăra o jumătate sau o treime de sac; ori cumpără un sac întreg ori nu. Fără această condiție avem de a face cu o problemă uzuală de programare liniară!

Exemplul 2 Un atelier de prelucrare a maselor plastice produce două tipuri de ambalaje pentru protejarea obiectelor din sticlă: tipul A pentru pahare de suc de şase uncii de şi tipul B pentru pahare de coctail de zece uncii.Ambalajele sunt fabricate cu ajutorul a două matrițe alimentate prin injecție de la o maşină care produce polistiren expandat. Cu prima matriță se pot realiza 100 ambalaje A în 6 ore în timp ce cu a doua matriță se pot face 100 ambalaje B în 5 ore.Procesul tehnologic nu permite folosirea simultană a matrițelor! Utilajul care produce materia primă poate lucra până la 60 ore pe săptămână. Ambalajele fabricate sunt stocate în depozitul atelierului care are o capacitate de 15000 u.v. (unități de volum ≡ picioare cubice).Un ambalaj A are nevoie de un spațiu de 10 u.v. iar un ambalaj B ocupă un spațiu de 20 u.v. Singurul client al ambalajelor A nu acceptă mai mult de 800 bucăți pe săptămână în timp ce ambalajele B pot fi realizate în orice cantitate, cererea fiind mare. Cele două tipuri de ambalaje aduc un profit de 5 $ respectiv 4.50 $ per bucată. Câte ambalaje din cele două tipuri ar trebui să producă atelierul în vederea maximizării profitului total?

I) Identificarea variabilelor modelului

La prima vedere (şi judecând prin analogie cu exemplul 1) ar trebui să luăm ca variabile de decizie cantităţile de ambalaje A şi B ce urmează a fi produse într-o săptămână; lucrurile sunt însă ceva mai complicate! Din enunţ rezultă că producţia de ambalaje este **discretizată** în **cicluri de fabricaţie**, un ciclu însemnând utilizarea uneia sau alteia dintre cele două matriţe pentru o perioadă determinată de timp.Astfel, un ciclu de fabricaţie pentru ambalajul A durează 6 ore şi are ca rezultat fabricarea a 100 bucăţi de produs finit; un ciclu de fabricaţie pentru ambalajul B dureză 5 ore şi are ca rezultat fabricarea a 100 ambalaje B.

În consecinţă, vom lua ca variabile de decizie:

$x_1 \equiv$ **numărul** ciclurilor de fabricaţie în care se produce ambalajul A

$x_2 \equiv$ **numărul** ciclurilor de fabricaţie în care se produce ambalajul B într-o săptămână.

Ca urmare, cantităţile de produse finite vor avea expresiile:

$100x_1$ bucăţi ambalaje A $\equiv x_1$ **sute** de bucăţi ambalaje A;
$100x_2$ bucăţi ambalaje B $\equiv x_2$ **sute** de bucăţi ambalaje B;

II) Idetificarea criteriului de performanţă şi formalizarea lui în funcţia obiectiv.

Obiectivul urmărit este **maximizarea profitului**; pe suta de bucăţi profiturile sunt:

$5 \times 100 = 500\,\$$ pentru ambalajul A şi $4.50 \times 100 = 450\,\$$ pentru ambalajul B.

Profitul săptămânal are expresia:

$$f = 500x_1 + 450x_2 \quad \rightarrow \quad \max \qquad (1)$$

II) Identificarea condiţiilor limitative şi formalizarea lor în restricţii.

- Există un plafon al timpului disponibil pentru producerea materiei prime care nu trebuie să depăşească 60 ore săptămânal:

$$6x_1 + 5x_2 \le 60 \qquad (2)$$

Membrul stâng reprezintă timpul necesar pentru realizarea a x_1 cicluri pentru ambalajul A şi a x_2 cicluri pentru ambalajul B (aici a intervenit ipoteza că cele două matriţe nu pot fi folosite simultan!)

- Spaţiul de depozitare al producţiei săptămânale este limitat la 15000 u.v.; 100 bucăţi A necesită $100 \times 10 = 1000$ u.v. iar 100 bucăţi B necesită $100 \times 20 = 2000$ u.v. astfel că producţia săptămânală va avea nevoie de un spaţiu pentru depozitare de $1000x_1 + 2000x_2$ u.v. Rezultă inegalitatea:

$$1000x_1 + 2000x_2 \le 15000 \quad \Leftrightarrow \quad x_1 + 2x_2 \le 15 \qquad (3)$$

- Producția ambalajului ‚A este limitată la (cel mult) 8 sute bucăți săptămânal:

$$x_1 \le 8 \qquad (4)$$

III) Condiții explicite impuse variabilelor.

Evident:

$$x_1 \ge 0\,,\; x_2 \ge 0 \qquad (5)$$

Deoarece un ciclu de fabricație este un proces care nu poate fi întrerupt, urmează că x_1, x_2 **nu au sens economic** dacă nu sunt exprimate prin numere **întregi**:

$$x_1\,,\; x_2 \text{ (numere) întregi} \qquad (6)$$

Reunind (1) – (6) obținem **modelul matematic**:

$$(P)\begin{cases} \max\; f = 500x_1 + 450x_2 \\ 6x_1 + 5x_2 \le 60 \\ x_1 + 2x_2 \le 15 \\ x_1 \le 8 \\ x_1, x_2 \ge 0 \\ x_1, x_2 \text{ întregi} \end{cases}$$

Rezolvarea modelului va fi dată în următoarea unitate de învățare. Soluția optimă este $x_1 = 8\,,\; x_2 = 2$ cu următoarea interpretare:

Profitul maxim posibil de realizat în condițiile date este de $8 \times 500 + 2 \times 450 = 4900\,\$$; el s-ar obține din producerea şi vânzarea a 800 ambalaje A şi a 200 ambalaje B. S-ar utiliza 58 ore din fondul de timp al utilajului care produce material primă şi s-ar folosi 12000 u.v. din spațiul disponibil pentru depozitare.

O altă sursă de modele cu variabile întregi o constituie situațiile în care trebuie luate decizii de tip „**da sau nu**” ca de exemplu:

- se poate aproba realizarea unui anumit proiect de dezvoltare ?
- se poate iniția o activitate implicând un anumit cost **fix** de pregătire ?
- se poate amplasa o facilitate (fabrică, depozit, magazin) într-un anumit loc ?

Existând doar două posibilități - de a aproba sau de a respinge - asemenea decizii pot fi reprezentate prin variabile cu numai două valori permise, 0 sau 1 (**variabile bivalente**):

1 - dacă decizia este **afirmativă**;
0 - dacă decizia este **negativă**.

Exemplul 3. Conducerea unei mari companii a luat în considerare şapte proiecte de investiţii. În tabelul 1.1, pentru fiecare proiect se precizează costul investiţiei, valoarea netă prezentă a profitului în caz de realizare precum şi condiţiile de care trebuie să se ţină seama în decizia de aprobare sau de respingere a proiectului. Bugetul alocat este de 50 milioane dolari. Conducerea doreşte să afle care proiecte ar trebui aprobate – cu respectarea tuturor condiţiilor - astfel încât costul total să se încadreze în bugetul alocat iar valoarea netă prezentă a profitului total să fie maximă.

Tabelul 1.1

Proiect	Valoarea netă prezentă – în caz de aprobare - (milioane$)	Costul investiţiei – în caz de aprobare – (milioane $)	Condiţii de aprobare sau respingere
P_1	1	5	-
P_2	2	8	**se poate** aproba **numai dacă** se aprobă proiectul P_1
P_3	3	12	**se poate** aproba **numai dacă** se respinge proiectul P_2
P_4	4	18	**trebuie** (neapărat) aprobat dacă proiectele P_1 şi P_2 au fost (ambele)aprobate
P_5	5	24	**se va respinge** în caz că unul dintre proiectele P_1 ,P_2 a fost aprobat
P_6	6	27	**se va respinge** dacă proiectele P_2 şi P_3 au fost (ambele) aprobate
P_7	7	30	**se poate** aproba **numai dacă** se aprobă proiectul P_2 şi se respinge proiectul P_3
			dintre proiectele P_1, P_2, P_3, P_4 **cel mult** două **pot fi** aprobate

Pentru modelare asociem proiectului P_j j = 1,..,7 variabila **bivalentă** x_j cu valorile:

$x_j = 1$ dacă proiectul P_j este aprobat;

şi

$x_j = 0$ dacă proiectul P_j este respins.

Aceste variabile ne vor permite să operăm cu valoarea netă prezentă a unui proiect sau cu costul său fără să ne mai gândim dacă proiectul a fost sau nu a fost aprobat!

Astfel, valoarea netă prezentă a proiectului P_6 va fi dată de expresia $6x_6$ cu numai două valori permise:

$$6 \quad \text{dacă } x_6 = 1 \quad \Leftrightarrow \quad \text{proiectul } P_6 \text{ este aprobat}$$
$$0 \quad \text{dacă } x_6 = 0 \quad \Leftrightarrow \quad \text{proiectul } P_6 \text{ este respins.}$$

Analog, costul proiectului P_6 va fi reprezentat de expresia $27x_6$.

Funcţia obiectiv formalizează valoarea netă prezentă a profitului total şi este dată de expresia:

$$f = 1 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 + 4 \cdot x_4 + 5 \cdot x_5 + 6 \cdot x_6 + 7 \cdot x_7 \quad \rightarrow \quad \max \qquad (1)$$

Restricțiile modelului formalizează:

- încadrarea cheltuielilor în bugetul alocat:

$$5x_1 + 8x_2 + 12x_3 + 18x_4 + 24x_5 + 27x_6 + 30x_7 \le 50 \qquad (2)$$

- condițiile speciale de aprobare / respingere. Vom ține seama de următoarele observații simple dar esențiale:

dacă x este o variabilă bivalentă atunci şi 1-x este o variabilă bivalentă;
dacă y şi z sunt variabile nenegative atunci $(y = 0 \text{ şi } z = 0) \Leftrightarrow y + z = 0$;
dacă y şi z sunt variabile nenegative atunci $(y = 0 \Rightarrow z = 0) \Leftrightarrow z \le y$.

- pentru proiectul P_2: prin negație, condiția se reformulează astfel:

dacă proiectul P_1 este respins atunci proiectul P_2 trebuie respins $\Leftrightarrow (x_1 = 0 \Rightarrow x_2 = 0) \Leftrightarrow x_2 \le x_1$ (3)

- pentru proiectul P_3: $(x_3 = 1 \Rightarrow x_2 = 0) \Leftrightarrow (1 - x_3 = 0 \Rightarrow x_2 = 0) \Leftrightarrow x_2 \le 1 - x_3 \Leftrightarrow x_2 + x_3 \le 1$ (4)

- pentru proiectul P_4: $(x_1 = 1 \text{ şi } x_2 = 1 \Rightarrow x_4 = 1) \Leftrightarrow (1 - x_1 = 0 \text{ şi } 1 - x_2 = 0 \Rightarrow 1 - x_4 = 0) \Leftrightarrow$
$(2 - x_1 - x_2 = 0 \Rightarrow 1 - x_4 = 0) \Leftrightarrow 1 - x_4 \le 2 - x_1 - x_2 \Leftrightarrow x_1 + x_2 - x_4 \le 1$ (5)

- pentru proiectul P_5: $(x_1 = 1 \text{ sau } x_2 = 1 \Rightarrow x_5 = 0)$ prin negare! $\Leftrightarrow (x_5 = 1 \Rightarrow x_1 = 0 \text{ şi } x_2 = 0) \Leftrightarrow$

$$(1 - x_5 = 0 \Rightarrow x_1 = 0 \text{ şi } x_2 = 0) \Leftrightarrow \begin{cases} 1 - x_5 = 0 \Rightarrow x_1 = 0 \\ 1 - x_5 = 0 \Rightarrow x_2 = 0 \end{cases} \Leftrightarrow \begin{cases} x_1 \le 1 - x_5 \\ x_2 \le 1 - x_5 \end{cases} \Leftrightarrow \begin{cases} x_1 + x_5 \le 1 \\ x_2 + x_5 \le 1 \end{cases} \qquad (6)$$

- pentru proiectul P_6: $(x_2 = 1 \text{ şi } x_3 = 1 \Rightarrow x_6 = 0) \Leftrightarrow (1 - x_2 = 0 \text{ şi } 1 - x_3 = 0 \Rightarrow x_6 = 0) \Leftrightarrow$
$(2 - x_2 - x_3 = 0 \Rightarrow x_6 = 0) \Leftrightarrow x_6 \le 2 - x_2 - x_3 \Leftrightarrow x_2 + x_3 + x_6 \le 2$ (7)

- pentru proiectul P_7: Multă atenție din cauza lui „se poate”; raționăm astfel: dacă proiectul P_7 a fost până la urmă aprobat atunci neapărat P_2 a fost aprobat iar P_3 a fost respins. Formal:

$$(x_7 = 1 \Rightarrow x_2 = 1 \text{ şi } x_3 = 0) \Leftrightarrow (1 - x_7 = 0 \Rightarrow 1 - x_2 = 0 \text{ şi } x_3 = 0) \Leftrightarrow \begin{cases} 1 - x_2 \le 1 - x_7 \\ x_2 \le 1 - x_7 \end{cases} \Leftrightarrow$$

$$\begin{cases} x_7 \le x_2 \\ x_3 + x_7 \le 1 \end{cases} \qquad (8)$$

- din primele patru proiecte se aprobă cel mult două: $x_1 + x_2 + x_3 + x_4 + \le 2$ (9)

Obținem în final următorul model matematic:

$$(P)\begin{cases} \max f = x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 + 6x_6 + 7x_7 \\ 5x_1 + 8x_2 + 12x_3 + 18x_4 + 24x_5 + 27x_6 + 30x_7 \le 50 \\ x_1 + x_2 \le 0 \\ x_2 + x_3 \le 1 \\ x_1 + x_2 - x_4 \le 1 \\ x_1 + x_5 \le 1 \\ x_2 + x_5 \le 1 \\ x_2 + x_3 + x_6 \le 2 \\ -x_2 + x_7 \le 0 \\ x_3 + x_7 \le 1 \\ x_1 + x_2 + x_3 + x_4 \le 2 \\ x_j \in \{0,1\} \quad j = 1,...,7 \end{cases}$$

Merită să zăbovim puțin asupra rezolvării acestui model: cel mai simplu mod de găsire a soluției optime constă în cercetarea – pe rând – a celor 2^7 =128 de combinații de valori 0, 1 date celor șapte variabile bivalente după schema:

- se generează o combinație (procedura de generare trebuie să satisfacă două cerințe:să nu producă de două ori aceeaşi combinație şi să fie capabilă să producă toate combinațiile posibile)
- se verifică dacă combinația generată satisface restricțiile modelului; dacă există măcar o restricție neverificată se trece la generarea altei combinații.Altminteri:
- se evaluează funcția obiectiv în soluția admisibilă generată. Această soluție se va reține cât timp nu este descoperită o altă combinație admisibilă capabilă să ofere funcției obiectivo valoare mai mare. **Enumerarea completă** încetează a fi o metodă eficientă de rezolvare dacă numărul variabilelor bivalente este mare. De exemplu, dacă numărul variabilelor este 30, numărul combinațiilor de valori 0, 1 este de 2^{30} =1.073.741.824 !

Soluția optimă a problemei date este : $x_1^* = 1$, $x_4^* = 1$, $x_6^* = 1$, $x_j^* = 0$ în rest ; (max) f =11milioane $

Aceasta înseamnă că aprobarea proiectelor 1, 4 şi 6 satisface condițiile enunțului, ar conduce la maximizarea valorii nete prezente a profitului total şi ar utiliza în întregime bugetul alocat.

Exemplul 4. Problema rucsacului Să modelăm următoarea situație ipotetică. Se dau *n* tipuri de obiecte. Unele obiecte vor fi alese pentru a umple un rucsac cu capacitatea (greutate, volum) *b*. Obiectele de tipul *j* sunt identice, sunt disponibile într-un număr limitat u_j şi fiecare dintre ele are o utilitate c_j şi o pondere (greutate, volum) a_j. Câte obiecte din fiecare tip vor fi introduse în rucsac astfel încât valoarea încărcăturii să fie maximă?

Notând:

$x_j \equiv$ numărul obiectelor de tipul *j* introduse în rucsac

obținem imediat următorul program întreg:

$$\begin{cases} (\max) f = \sum_{j=1}^{n} c_j x_j \\ \sum_{j=1}^{n} a_j x_j \le b & (1) \\ 0 \le x_j \le u_j & (2) \\ x_j \text{ întregi} \end{cases}$$

Chiar şi în absența mărginirilor (2) variabilele x_j nu pot lua decât un număr finit de valori pentru că:

$$x_j \le \left\lfloor b / a_j \right\rfloor \qquad (3)$$

Dacă se ignoră relațiile de mărginire (2) - care, de altfel, nu contează mai mult decât mărginirile „intrinseci" (3) – modelul problemei rucsacului are o singură restricție „veritabilă",inegalitatea (1), şi astfel este cel mai simplu program întreg. El prezintă interes atât în sine cât mai ales prin faptul că apare ca subprogram în contexte de optimizare mai generale.

Un interes aparte îl are cazul particular în care din fiecare tip de obiecte există un singur exemplar. Atunci $u_j = 1 \quad j = 1,...,n$ şi în consecință variabilele x_j nu mai pot lua decât valorile 0 sau 1. Rezultă modelul problemei bivalente a rucsacului:

$$\begin{cases} (\max)\ f = \sum_{j=1}^{n} c_j x_j \\ \sum_{j=1}^{n} a_j x_j \le b \\ x_j \in \{0, 1\} \end{cases}$$

cu următoarea interpretare alternativă,deja întâlnită în exemplul precedent: există n proiecte de investiții pentru care este alocată suma b. Proiectul j necesită investiția a_j şi are o valoare netă prezentă c_j. Ce proiecte pot fi realizate cu capitalul disponibil dat astfel încât valoarea netă prezentă totală să fie maximă?

De obicei, proiectele se întind pe mai multe perioade (ani, luni etc) iar investițiile necesare ca şi capitalul disponibil sunt defalcate pe perioade.Modelul corespunzător este:

$$\begin{cases} (\max)\ f = \sum_{j=1}^{n} c_j x_j \\ \sum_{j=1}^{n} a_{ij} x_j \le b_i \quad i = 1,...,m \\ x_j \in \{0, 1\} \end{cases}$$

în care:

$m \equiv$ numărul de perioade;
$b_i \equiv$ suma disponibilă pentru perioada i ;
$a_{ij} \equiv$ investiția necesară proiectului j în perioada i.

Exemplul 5. Elaborarea programelor de producție cu costuri de pregătire

Să considerăm programul liniar:

$$\begin{cases} (\min) f = \sum_{j=1}^{n} c_j x_j \\ \sum_{j=1}^{n} a_{ij} x_j = b_i \quad i = 1,...,m \\ x_j \geq 0 \quad j = 1,...,n \end{cases} \qquad (1)$$

în care $x_1, x_2, ..., x_n$ reprezintă nivelul unor activități productive iar $c_1, c_2, ..., c_n$ sunt costurile unitare aferente. Ipoteza uzuală de liniaritate presupune **proporționalitatea directă** între costul unei activități şi nivelul la care este operată activitatea respectivă. Există însă situații în care demararea unei activități necesită un **cost de pregătire**, independent de nivelul la care activitatea va fi operată. Costul activității j va avea deci forma:

$$c_j(x_j) = \begin{cases} 0 & \text{dacă } x_j = 0 \\ q_j + c_j x_j & \text{dacă } x_j > 0 \end{cases}$$

unde q_j este costul fix al pregătirii începerii activității j.
Pentru a încorpora aceste elemente în modelul (1) introducem variabilele bivalente:

$$y_j = \begin{cases} 1 & \text{dacă activitatea } j \text{ va fi operată la un nivel } x_j > 0 \\ 0 & \text{în caz contrar} \end{cases}$$

Funcția obiectiv $f = \sum_{j=1}^{n} c_j x_j$ din (1) se va extinde cu termenul $\sum_{j=1}^{n} q_j x_j$ reprezentând costul total de pregătire aferent activităților ce vor fi efectiv operate.Pentru fiecare j variabilele x_j şi y_j nu sunt independente: este clar că nu putem avea simultan $x_j > 0$ şi $y_j = 0$ sau $x_j = 0$ şi $y_j = 1$. Prima situație va fi exclusă prin introducerea inegalităților:

$$x_j \leq m_j y_j \quad j = 1,...,n \qquad (2)$$

în care m_j este o constantă pozitivă foarte mare ce depăşeşte, de exemplu, nivelul maxim posibil la care ar putea fi operată activitatea *j*. Într-adevăr, $y_j = 0$ şi ipoteza de nenegativitate $x_j \geq 0$ transformă (2) în $x_j = 0$. Dacă $y_j = 1$, (2) devine $x_j \leq m_j$ şi este banal verificată având în vedere semnificația constantei m_j. A doua situație se exclude prin faptul că în model se urmăreşte minimizarea costului producției şi al celui de pregătire (nu are rost să „pregătim" o activitate dacă nu se intenționează operarea ei; pregătirea ar implica o cheltuială inutilă!).
Astfel, programul liniar (1) se extinde la programul bivalent mixt:

$$\begin{cases} (\min) f = \sum_{j=1}^{n} c_j x_j + \sum_{j=1}^{n} q_j x_j \\ \sum_{j=1}^{n} a_{ij} x_j = b_i \quad i = 1,...,m \\ x_j \leq m_j x_j \quad j = 1,...,n \\ x_j \geq 0 \,;\, y_j \in \{0,1\} \end{cases}$$

Exemplul 6. Problema amplasării facilităților. Conducerea unei firme de distribuție studiază problema deschiderii unor noi depozite pe nişte amplasamente 1,2,...,*m* (anterior alese şi fixate) în vederea unei mai bune satisfaceri a cererilor clienților 1,2,...,*n*. În principiu, fiecare depozit are o capacitate proiectată în stare să satisfacă orice cerere – de la unul sau mai mulți clienți. Deschiderea unui nou depozit pe amplasamentul *i* necesită un cost fix q_i iar livrarea în întregime a cererii clientului *j* din depozitul *i* implică un cost de transport c_{ij}. Chestiunea este de a decide ce depozite vor fi efectiv deschise astfel încât suma costurilor fixe de deschidere şi a celor de transport să fie minimă.

Este evidentă asemănarea dintre această problemă şi problema clasică de transport: şi aici există furnizori şi consumatori şi se urmăreşte aducerea cheltuielilor la un nivel cât mai mic. Există totuşi o deosebire notabilă. În problema de față, fiecare furnizor (depozit) este deocamdată un furnizor potențial. El devine un furnizor real printr-un act prealabil de decizie, act ce implică un anumit cost, independent de modul în care respectivul furnizor – o dată admis – va contribui la satisfacerea cererilor clienților.

Introducem variabilele:

$x_{ij} \equiv$ fracția din cererea clientului *j* livrată din depozitul *i*;

$$y_i = \begin{cases} 1 & \text{daca pe amplasamentul } i \text{ se deschide un depozit} \\ 0 & \text{in caz contrar} \end{cases}$$

Cerința satisfacerii cererii fiecărui client implică egalitățile:

$$\sum_{i=1}^{m} x_{ij} = 1 \quad j = 1,...,n \qquad (1)$$

Evident, dacă pe amplasamentul *i* nu se deschide un depozit nu vor exista livrări din acest loc. Formal:

$$y_i = 0 \quad \Rightarrow \quad x_{ij} = 0\,,\ j = 1,...,n \qquad (2)$$

Deoarece $x_{ij} \le 1$ implicațiile (2) sunt echivalente cu inegalitățile:

$$x_{ij} \le y_i \quad i = 1,...,m\,;\ j = 1,...,n \qquad (3)$$

Se poate vedea uşor că implicațiile (2) pot fi formalizate alternativ şi prin setul de inecuații:

$$\sum_{j=1}^{n} x_{ij} \le n \cdot y_i \quad i = 1,...,m \qquad (3')$$

Într-adevăr, dacă $y_i = 0$, din (3') rezultă $\sum_{j=1}^{n} x_{ij} \le 0$, adică $x_{ij} = 0 \quad j = 1,...,n$, deoarece x_{ij} sunt variabile nenegative. Dacă $y_i = 1$, (3') devine: $\sum_{j=1}^{n} x_{ij} \le n$ şi este banal satisfăcută pentru că membrul stâng este o sumă de *n* termeni subunitari.
Funcția obiectiv are două componente:

- costul deschiderii depozitelor pe amplasamentele date, cost reprezentat prin suma $\sum_{i=1}^{m} q_i y_i$;

deoarece y_i este fie 0 fie 1 este clar că această sumă cuprinde numai costurile depozitelor efectiv deschise;

- costul transportului de la depozite la clienți, reprezentat de expresia $\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij}$. Relațiile (3) sau (3') fac ca în această sumă să apară numai costurile de transport de la depozitele efectiv deschise, la clienți.

Costul total de minimizat va avea deci expresia:

$$f = \sum_{i=1}^{m} q_i y_i + \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij} \qquad (4)$$

Reunind (1), (3) şi (4) sau (1), (3') şi (4) obținem în final două formulări alternative pentru problema dată:

$$a)\begin{cases} (\min) f = \sum_{i=1}^{m} q_i y_i + \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij} \\ \sum_{i=1}^{m} x_{ij} = 1 \quad j = 1,...,n \\ x_{ij} \le y_i \quad i = 1,..,m\,;\ j = 1,..,n \\ x_{ij} \ge 0\,,\ y_i \in \{0\,,1\} \end{cases} \qquad b)\begin{cases} (\min) f = \sum_{i=1}^{m} q_i y_i + \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij} \\ \sum_{i=1}^{m} x_{ij} = 1 \quad j = 1,...,n \\ \sum_{j=1}^{n} x_{ij} \le n \cdot y_i \quad i = 1,..,m \\ x_{ij} \ge 0\,,\ y_i \in \{0\,,1\} \end{cases}$$

Observaţie: Modelul a) are mai multe restricţii decât b) deoarece setul (3) are *mn* inegalităţi în timp ce setul alternativ (3') are numai *m*. Am putea spune că modelul b) oferă o formulare mai „compactă" a problemei date decât modelul a). Paradoxal, deşi este mai „stufos" modelul a) este preferabil la rezolvare!!

Exemplul 7. Formalizarea restricţiilor disjunctive Există situaţii în care restricţiile unei probleme de optimizare nu trebuie satisfăcute simultan. Ca exemplu, să luăm o problemă de ordonanţare în care două joburi pot fi procesate pe un acelaşi utilaj dar nu în acelaşi timp. Dacă p_1, p_2 sunt duratele de procesare ale celor două joburi iar variabilele t_1, t_2 desemnează momentele de începere ale operaţiilor atunci sau jobul 1 precede jobul 2 în care caz $t_2 \geq t_1 + p_1$ sau jobul 2 intră primul în lucru şi atunci $t_1 \geq t_2 + p_2$.

Să considerăm o problemă de optimizare al cărei vector de variabile $x \in R^n$ trebuie să satisfacă printre alte cerinţe şi condiţia „disjunctivă":

$$a \cdot x = a_1 x_1 + \ldots + a_n x_n \leq b \quad \textbf{sau} \quad a' \cdot x = a'_1 x_1 + \ldots + a'_n x_n \leq b'$$

Se pune problema de a exprima această condiţie în maniera uzuală, „conjunctivă", adică printr-un set de relaţii care să fie verificate simultan. Vom presupune că există o constantă M care mărgineşte superior valorile expresiilor $a \cdot x - b$ şi $a' \cdot x - b'$.Această condiţie este îndeplinită dacă, de exemplu, valorile variabilelor problemei sunt mărginite, aşa cum se întâmplă în mai toate contextele practice.Pentru a răspunde la chestiunea pusă introducem variabilele bivalente y_1, y_2 şi considerăm setul „conjunctiv" de relaţii:

$$\begin{cases} a \cdot x - b \leq M(1 - y_1) \\ a' \cdot x - b' \leq M(1 - y_2) \\ y_1 + y_2 = 1 \end{cases} \qquad (1)$$

Deoarece y_1, y_2 sunt variabile bivalente, două situaţii sunt posibile:

- $y_1 = 1, y_2 = 0$ Setul (1) devine: $\begin{cases} a \cdot x \leq b \\ a' \cdot x - b' \leq M \end{cases} \Leftrightarrow a \cdot x \leq b$,deoarece relaţia $a' \cdot x - b' \leq M$ este superfluă având în vedere definiţia lui M.
- $y_1 = 0, y_2 = 1$ Raţionând ca mai sus, (1) se reduce acum la inegalitatea $a' \cdot x \leq b'$.

Observaţie: Cititorul atent a observat desigur că în situaţia descrisă este nevoie de fapt de o singură variabilă bivalentă *y* (se ia $y_1 = y$ şi $y_2 = 1 - y$!)
Cazul a *p* restricţii disjunctive:

$$a^1 \cdot x \leq b^1 \text{ sau } a^2 \cdot x \leq b^2 \text{ sau } \ldots \text{ sau } a^p \cdot x \leq b^p \qquad (2)$$

se tratează absolut analog cu ajutorul a p variabile bivalente $y_1, y_2, .., y_p$. Se vede uşor că sistemul „conjunctiv”:

$$\begin{cases} a^i \cdot x - b^i \le M(1 - y_i) \quad i = 1,.., p \\ \sum_{i=1}^{p} y_i = 1 \end{cases}$$

este echivalent cu sistemul „disjunctiv” (2).

Exemplul 8. Problema monezilor Cum poate fi plătită o sumă de bani astfel încât:

- numărul tipurilor valorice de monezi utilizate la plată să nu depăşească o limită dată;
- numărul total al monezilor necesare plății să fie minim.

Introducem notațiile:

S = suma de plată,
n = numărul total al tipurilor valorice de monezi disponibile pentru plată;
p = numărul maxim de tipuri valorice de monezi ce pot fi utilizate la plata sumei S;
v_j = valoarea monezii de tip j;
m_j = numărul monezilor de tip j disponibile în casă.

Introducem variabilele întregi:

x_j = numărul monezilor de tip j utilizate la plata sumei S

şi variabilele bivalente:

$$y_j = \begin{cases} 1 & \text{daca moneda de tip } j \text{ va fi efectiv utilizata la plata sumei } S \\ 0 & \text{in caz contrar} \end{cases}$$

Rezultă următorul program întreg:

$$\begin{cases} (\min) f = \sum_{j=1}^{n} x_j & \textbf{(1)} \\ \sum_{j=1}^{n} v_j x_j = S & \textbf{(2)} \\ \sum_{j=1}^{n} y_j \le p & \textbf{(3)} \\ x_j \le m_j y_j \quad j = 1,..., n & \textbf{(4)} \\ x_j \ge 0 \text{ , } \textbf{intregi} \text{ ; } y_j \in \{0, 1\} \end{cases}$$

Funcţia obiectiv (1) reprezintă numărul total de monezi folosite la plata sumei *S*.Egalitatea (2) arată că suma valorilor monezilor utilizate trebuie să fie egală cu suma de plată în timp ce inegalitatea (3) formalizează cerinţa de a nu se folosi la plată mai mult de *p* tipuri valorice de monezi.Seturile de variabile (x_j) şi (y_j) sunt legate prin inegalităţile (4):

- dacă $y_j = 0$ adică moneda de tip *j* nu este utilizată, (4) implică $x_j = 0$ ceea ce este normal;
- dacă $y_j = 1$ adică moneda de tip *j* este efectiv utilizată la plată, (4) devine $x_j \le m_j$ şi exprimă

cerinţa naturală ca numărul monezilor de tip *j* incluse în plata sumei *S* să nu depăşească disponibilul m_j.

1.2 Terminologie

Pentru cele ce urmează este necesar să precizăm câţiva termeni.

- variabilă **continuă** ≡ variabilă care, o dată cu două valori permise, poate lua orice altă valoare **intermediară**;
- **variabilă întreagă** ≡ variabilă care nu poate lua decât valori numere **întregi**;
- **variabilă bivalentă (binară sau booleană)**≡variabilă **întreagă** care nu poate lua decât valorile 0 sau 1;

În continuare vom avea în vedere numai probleme de optimizare în care funcţia obiectiv şi restricţiile sunt **liniare**.Motivaţia acestei restrângeri voluntare o constituie faptul că în cvasitotalitatea situaţiilor practice suntem conduşi la modele alcătuite numai din relaţii liniare.

• problemă de **programare în numere întregi** (pe scurt **program întreg**) ≡ problemă de programare **liniară** care utilizează una sau mai multe variabile întregi. Problema se zice **totală** dacă toate variabilele sale sunt întregi şi **mixtă** dacă utilizează simultan şi variabile continue şi variabile întregi;

• problemă de **programare bivalentă** (sau **program bivalent**) – totală sau mixtă ≡ problemă care utilizează variabile bivalente.

• programul **relaxat** ≡ programul liniar uzual rezultat dintr-un program întreg prin eliminarea condiţiei ca variabilele să ia numai valori întregi.Orice soluţie admisibilă a programului întreg este soluţie admisibilă şi pentru programul relaxat nu şi reciproc, deoarece programul relaxat are de regulă soluţii admisibile în care unele variabile întregi au valori fracţionare (neîntregi) Soluţia optimă a programului relaxat se va numi soluţia optimă **fracţionară** pentru a o deosebi de soluţia optimă **întreagă** a programului original. Evident, un program întreg şi relaxatul său au aceeaşi funcţie obiectiv; valorile acestei funcţii în soluţia optimă întreagă şi în cea fracţionară se vor numi **optim întreg** respectiv **optim fracţionar**. Pentru exemplificare rescriem mai jos programul întreg din exemplul 2 împreună cu relaxatul său:

$$(P)\begin{cases}(\max) f = 500x_1 + 450x_2\\ 6x_1 + 5x_2 \le 60\\ x_1 + 2x_2 \le 15\\ x_1 \le 8\\ x_1, x_2 \ge 0\\ x_1, x_2 \text{ întregi}\end{cases} \qquad (PL)\begin{cases}(\max) f = 500x_1 + 450x_2\\ 6x_1 + 5x_2 \le 60\\ x_1 + 2x_2 \le 15\\ x_1 \le 8\\ x_1, x_2 \ge 0\end{cases}$$

Soluția optimă fracționară $x^* = (x_1^*, x_2^*)$ - adică soluția optimă a programului relaxat (PL) –are componentele:

$$x_1^* = 6\frac{3}{7} \qquad x_2^* = 4\frac{2}{7}$$

Optimul fracționar are valoarea: $f(x^*) = 500 \times 6\frac{3}{7} + 450 \times 4\frac{2}{7} = 5142\frac{6}{7}$

Soluția optimă întreagă $x^0 = (x_1^0, x_2^0)$ – adică soluția optimă a programului original (P) – are componentele:

$$x_1^0 = 8 \qquad x_2^0 = 2$$

iar optimul întreg are valoarea: $f(x^0) = 500 \times 8 + 450 \times 2 = 4900$ (vezi unitatea de învățare 2)

Probleme propuse

1. Se consideră un program întreg (P) în care funcția obiectiv se **maximizează,** împreună cu relaxatul său (PL). Dacă max P, max PL desemnează optimul întreg respectiv optimul fracționar din cele două probleme, care din următoarele afirmații este **întotdeauna** adevărată?

a) $\max P < \max PL$; b) $\max P > \max PL$; c) $\max P \ge \max PL$; d) $\max P \le \max PL$; e) relația dintre max *P* şi max *PL* depinde de specificul problemei.

2. Arătați că dacă soluția optimă a relaxatei unei probleme de programare în numere întregi (P) este şi o soluție admisibilă pentru (P) atunci ea este chiar o soluție optimă a problemei (P).

3. a) Recapitulați procedura de rezolvare a unui program întreg prin enumerarea completă a tuturor combinațiilor de valori întregi date variabilelor (vezi finalul exemplului 3)

b) Printre restricțiile unui program întreg total cu trei variabile nenegative x_1, x_2, x_3 există şi inegalitățile $x_1 \le 1$, $x_2 \le 2$, $x_3 \le 3$.Cât ar dura rezolvarea problemei dacă examinarea fiecărei combinații de valori permise date variabilelor necesită cinci minute?

a) 30 min.; b) o oră; c) 1,5 ore; d) 2 ore; e) 2,5 ore.

4. O mică firmă produce bunurile alimentare G_1, G_2, G_3 folosind trei materii prime agricole de bază R_1, R_2, R_3. Necesarul de resurse, în kg, pentru realizarea unui kilogram din fiecare din bunurile G_1, G_2, G_3 sunt indicate în tabelul 1.2

Pentru ziua următoare există în stoc 45 kg din R_1, 30 kg din R_2 şi 20 kg din R_3.Profiturile sunt de 7, 9, 12 unităţi monetare pe kilogramul din G_1, G_2, respectiv G_3. Firma livrează bunul G_1 în cutii de ½ kg, bunul G_2 în cutii de 1 kg şi bunul G_3 în cutii de 2 kg iar desfacerea este asigurată pentru tot ceeace se produce.

Tabelul 1.2

Bunuri / Resurse	G_1	G_2	G_3
R_1	2	4	5
R_2	1	3	2
R_3	2	1	1

1) Scrieţi un program liniar în numere întregi pentru determinarea programului de activitate al firmei din ziua următoare având ca obiectiv maximizarea profitului.
2) Cum se modifică programul iniţial dacă din bunul G_1 nu trebuie să se facă mai mult de zece cutii?
3) Cum se modifică programul iniţial dacă din bunul G_1 se pot face cel mult zece cutii iar din G_3 cel puţin trei?
4) Deoarece resursele R_1, R_2, R_3 sunt foarte perisabile firma este interesată în a le valorifica cât mai bine. Cum se poate modela acest deziderat?

5. O fabrică de ciment produce două mărci de ciment C_1 şi C_2. Pentru următoarea săptămână sunt programate realizarea a 1500t ciment C_1 şi a 2400t ciment C_2. Fabrica are două instalaţii I_1 si I_2 cu următoarele caracteristici:

- Pe fiecare ciclu de fabricaţie instalaţia I_1 poate produce 80t ciment C_1 sau 120t ciment C_2, în timp ce instalaţia I_2 poate produce 160t ciment C_1 sau 180t ciment C_2;
- Un ciclu de fabricaţie durează 8 ore pentru cimentul C_1 şi 6 ore pentru cimentul C_2 indiferent de instalaţia producătoare;
- În următoarea săptămână fondurile de timp productiv ale celor două instalaţii sunt de 120 ore pentru I_1 şi de 100 ore pentru I_2;
- Costurile de producţie (u.m. pe tona de ciment) variază în funcţie de marca cimentului şi instalaţia producătoare aşa cum rezultă din tabelul 1.3.

Cum vor fi folosite cele două instalaţii pentru ca programul săptămânii următoare să fie realizat cu costuri minime?

Tabelul 1.3

	I_1	I_2
C_1	30	25
C_2	20	15

6. Se consideră problema programării producţiei a 1900 unităţi dintr-un produs ce poate fi realizat pe trei linii de fabricaţie diferite. Începerea activităţii de producţie pe o linie de fabricaţie necesită un “cost de pregătire”, indiferent de volumul producţiei ce va fi realizat pe linia respectivă. Costul fabricării unei unităţi de produs pe o linie sau alta ca şi capacităţile maxime de producţie ale celor trei linii sunt date în tabelul 1.4

Tabelul 1.4

Linia de fabricație	Cost de pregătire	Cost de producție (u.m./unitate)	Cap. maximă de producție (unități)
1	100	10	600
2	300	2	800
3	200	5	1200

Cum trebuie organizată activitatea pe cele trei linii de fabricație astfel încât costul total de pregătire şi de producție să fie minim?

7. Pentru asigurarea activității curente de producție, o firmă are nevoie de anumite cantități din patru repere. În principiu, firma are posibilitatea de a fabrica aceste repere cu mijloace proprii dar conducerea este de părere că resursele disponibile nu sunt suficiente pentru producerea cantităților necesare astfel că se pune problema achiziționării unora de pe piață, cel puțin în parte. În procesul de fabricație al reperelor la firmă sunt implicate şase utilaje,fiecare cu un număr limitat de ore de fiuncționare.Datele concrete ale situației sunt indicate în tabelul 1.5

Tabelul 1.5

Repere / Utilaje	Consumuri unitare de timp de prelucrare						Cost de prod. ($/buc.)	Cost de achiziție ($/buc.)	Cantitate necesară
	1	2	3	4	5	6			
1	0.04	0.02	0.02	-	0.03	0.06	2.55	3.10	150
2	-	0.01	0.05	0.15	0.09	0.06	2.47	2.60	150
3	0.02	0.06	-	0.06	0.20	0.20	4.40	4.50	150
4	0.06	0.04	0.15	-	-	0.05	1.90	2.25	150
Nr. ore de funcționare	40	40	30	40	30	35			

Conducerea firmei este interesată în a stabili cât să producă şi cât să cumpere de piață astfel încât totalul cheltuielilor să fie minim.

1) Scrieți un program liniar care să răspundă acestui obiectiv;
2) Analizând soluția obținută, conducerea firmei consideră că nu este rentabil ca reperele să fie produse în parte la firmă şi restul să fie cumpărat de pe piață deoarece aceasta ar produce complicații în organizarea producției proprii ca şi în relațiile cu partenerii externi. Pentru asigurarea necesarului de repere cu mijloace proprii ar exista posibilitatea creşterii numărului orelor de funcționare al utilajelor prin folosirea acestora "peste program" dar o asemenea politică nu pare a avea asentimentul personalului muncitor. Din aceste motive, conducerea a decis, ca pentru fiecare reper, cantitatea necesară sau să fie produsă în întregime în cadrul firmei sau să fie achiziționată în totalitate de pe piață. Ce modificări apar în modelul construit anterior?

8. Conducerea firmei ProTZ a luat în discuție şase proiecte de investiții. Alternativele investiționale, valoarea netă prezentă a profiturilor viitoare, cererea de capital şi fondurile disponibile pe următorii trei ani sunt date în tabelul 1.6

Tabelul 1.6

Proiecte	Val.netă prezentă a profitului	Cerere de capital (mii$)		
		Anul I	Anul II	Anul III
1. Dezvoltare limitată a capacităților de depozitare	4	3	1	4
2. Dezvoltare masivă a capacităților de depozitare	6	2.5	3.5	3.5
3. Testarea pe piață a unui nou produs	10.5	6	4	5
4. Campanie publicitară	4	2	1.5	1.8
5. Cercetare fundamentală	8	5	1	4
6. Achiziționarea de echipamente noi	3	1	0.5	0.9
Fonduri disponibile (mii$)		10.5	7	8.75

1) Formulați un program bivalent pentru alegerea acelor proiecte care maximizează valoarea netă prezentă totală.
2) Ce modificare intervine în programul inițial dacă trebuie realizat măcar unul din proiectele de extindere a capacităților de depozitare?
3) Ce modificare intervine în programul inițial dacă alegerea proiectului 6 implică neapărat realizarea proiectului 5?

Bibliografie

Nica,V. T.,Mustață, Fl., Mărăcine, V.,Ciobanu, Gh., Cercetări Operaționale, Editura Matrix Rom, Bucureşti, 1998

Hillier,F. S., Lieberman, G. J., Introduction to Operations Research, Mc Graw Hill Publishing Company, New York, ...,2001

Taha, H. A., Operations Research: an introduction, 8th ed., Pearson Prentice Hall, New Jersey, 2007

Bronson, R., Naadimuthu, G., Theory and Problems of Operations Research, Tata Mc Graw Hill Publishing Company Limited, New Delhi, 2008

Unitatea de învățare 2

PROGRAMAREA ÎN NUMERE ÎNTREGI
Particularitățile programării în numere întregi

Cuprins

2.1 Reprezentări geometrice
2.2 Programarea liniară vs programarea în numere întregi
2.3 Dificultăți în studiul şi rezolvarea programelor întregi

Probleme propuse

Obiectivele unității de învățare 2

Exemplele prezentate în unitatea de învățare 1 au arătat că utilizarea variabilelor întregi aduce un plus de **flexibilitate** în modelarea unor situații practice. Această flexibilitate costă însă destul de mult, deoarece programele întregi sunt mult mai greu de rezolvat decât programele liniare uzuale.Diferențele structurale dintre programarea liniară şi programarea în numere întregi vor fi treptat relevate şi explicate prin parcurgerea următoarelor etape:

- rezolvarea grafică a relaxatului unui program întreg în două variabile (recapitulare din cursul de Cercetări Operaționale I !);
- recapitularea trăsăturilor fundamentale ale programării liniare;
- evidențierea soluțiilor întregi şi a anvelopei convexe a acestora;
- dificultăți structurale cauzate de prezența variabilelor întregi;
- conceptul de tăietură; utilizarea tăieturilor în rezolvarea programelor întregi;
- rezolvarea aproximativă a programelor întregi prin rotunjire; dezavantaje;
- analiza sensitivității în programarea întreagă.

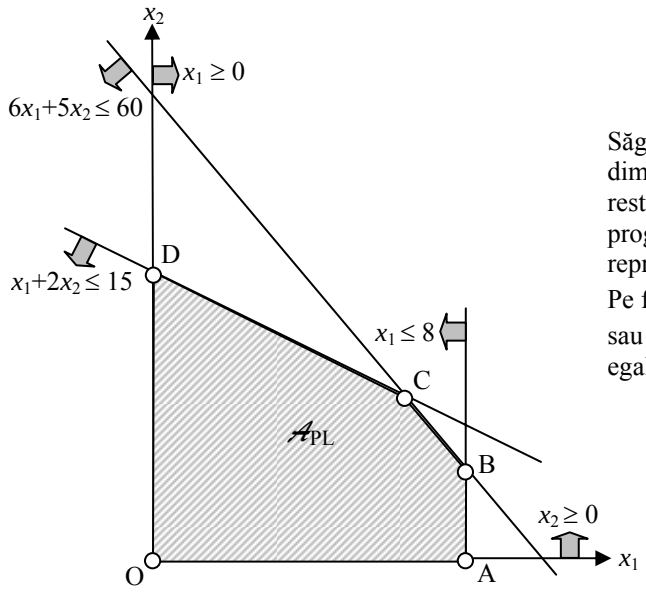
2.1 Reprezentări geometrice

Ca şi în alte domenii de cercetare, reprezentările şi interpretările geometrice au jucat un rol decisiv în stabilirea rezultatelor fundamentale ale programării matematice în general şi ale programării liniare în special.

Vom considera programul întreg (P) în două variabile rezultat din modelarea situației practice descris în unitatea de învățare 1, exemplul 2; alăturat am notat programul relaxat (PL):

$$(P)\begin{cases}\max\ f = 500x_1 + 450x_2\\ 6x_1 + 5x_2 \le 60\\ x_1 + 2x_2 \le 15\\ x_1 \le 8\\ x_1, x_2 \ge 0\\ x_1, x_2 \text{ intregi}\end{cases} \qquad (PL)\begin{cases}\max\ f = 500x_1 + 450x_2\\ 6x_1 + 5x_2 \le 60\\ x_1 + 2x_2 \le 15\\ x_1 \le 8\\ x_1, x_2 \ge 0\end{cases}$$

Ne propunem să rezolvăm grafic cele două programe, punând în evidență principalele proprietăți ale mulțimilor lor de soluții admisibile. Analiza comparativă a acestor proprietăți ne va permite degajarea – cel puțin la nivel de principiu – a unor metode de rezolvare a programelor întregi. Ideea de bază este aceea de a identifica orice cuplu de valori numerice x_1, x_2 cu un punct într-un plan raportat la un sistem de axe de coordonate.



Săgețile indică semiplanele (sau semispațiile în dimensiuni mai mari...) în care sunt verificate restricțiile şi condițiile de nenegativitate din programul (PL) Punctele mulțimii haşurate OABCD reprezintă soluțiile admisibile ale programului (PL) Pe frontiera mulțimii $\mathcal{A}_{PL}$ una sau alta dintre restricții sau condiții de nenegativitate este îndeplinită cu egalitate!

Figura 2.1

Fie $\mathcal{A}_P$ şi $\mathcal{A}_{PL}$ mulțimile de soluții admisibile ale programelor (P) respectiv (PL). Vizualizăm mai întâi $\mathcal{A}_{PL}$ – vezi figura 2.1 Reamintim că toate perechile de valori numerice care satisfac prima

restricție $6x_1 + 5x_2 \le 60$ se identifică cu punctele unuia dintre cele două semiplane determinate de dreapta $6x_1 + 5x_2 = 60$; aici este vorba de semiplanul care conține origina sistemului de axe deoarece coordonatele $x_1 = 0, x_2 = 0$ ale acesteia satisfac inegalitatea.
Vizualizarea punctelor ale căror coordonate satisfac celelalte două restricții se tratează asemănător.

Condiția de nenegativitate $x_1 \ge 0$ are loc în semiplanul "din dreapta" axei verticale $x_1 = 0$ în timp ce condiția $x_2 \ge 0$ este îndeplinită în semiplanul "de deasupra" axei orizontale $x_2 = 0$.

În figura 2.1 sunt evidențiate cele cinci semiplane în care sunt verificate cele trei restricții şi cele două condiții de nenegativitate din programul (PL). Partea lor comună, adică mulțimea haşurată OABCD, este formată din punctele ale căror coordonate satisfac simultan aceste condiții şi ca urmare reprezintă mulțimea de soluții admisibile ale programului relaxat (PL).

Relația de definiție a funcției obiectiv $500x_1 + 450x_2 = f$ poate fi interpretată şi ca ecuația unui fascicol de drepte paralele numite **drepte de nivel** ale funcției *f*. Dacă o dreaptă de nivel particulară $500x_1 + 450x_2 = \bar{f}$ intersectează mulțimea $\mathcal{A}_{PL}$, aceasta va însemna că programul (PL) are soluții admisibile care oferă funcției obiectiv valoarea prestabilită $\bar{f}$.

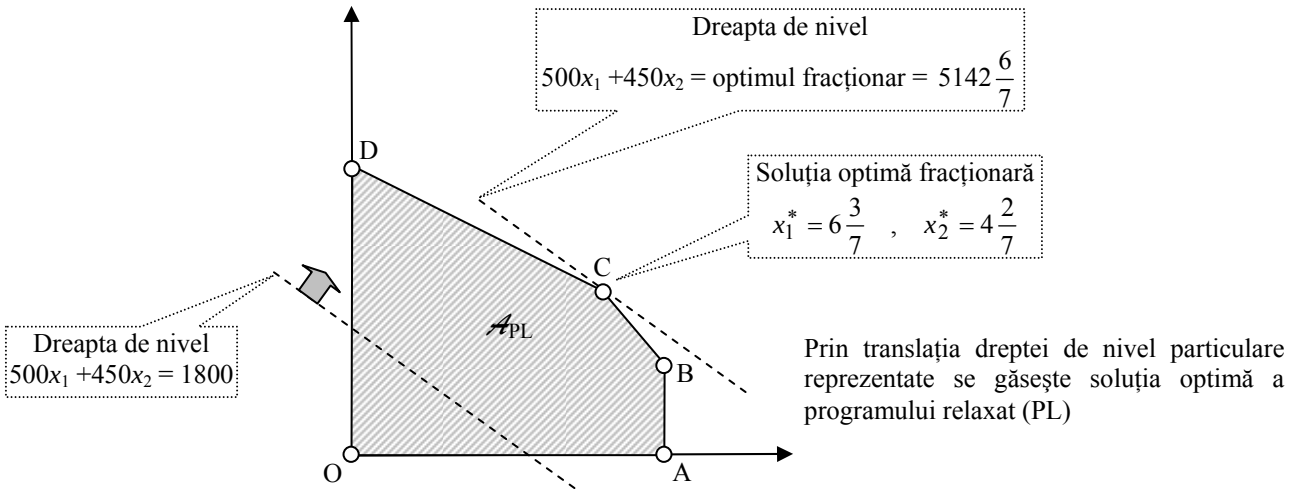


Figura 2.2

În figura 2.2 a fost reprezentată dreapta de nivel $500x_1 + 450x_2 = 1800$. Coordonatele oricărui punct al intersecției acestei drepte cu $\mathcal{A}_{PL}$ reprezintă o soluție admisibilă în care funcția obiectiv are valoarea 1800. Pentru valori numerice superioare, dreptele de nivel corespunzătoare sunt **translații** ale dreptei originale în sensul indicat de săgeată. Rezultă imediat că maximul funcției *f* se atinge în „vârful" C al frontierei mulțimii $\mathcal{A}_{PL}$. Punctul C este intersecția dreptelor de ecuații $6x_1 + 5x_2 = 60$ şi $x_1 + 2x_2 = 15$; rezolvând sistemul lor găsim că C are coordonatele $x_1 = \frac{45}{7}, x_2 = \frac{30}{7}$.

În concluzie, programul relaxat (PL) are soluția optimă „fracționară” $x^* = (6\frac{3}{7}, 4\frac{2}{7})$. Optimul „fracționar” are valoarea $f(x^*) = 5142\frac{6}{7}$.

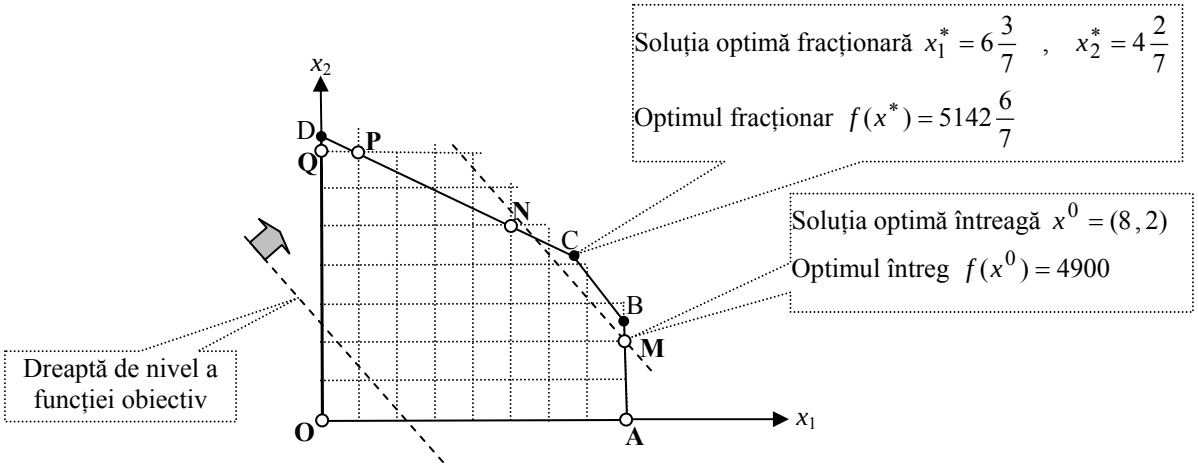


Figura 2.3

Prin simplă inspecție, se constată că mulțimea soluțiilor admisibile întregi ale programului (P) se identifică cu mulțimea celor 54 de puncte cu coordonate întregi situate în interiorul sau pe frontiera mulțimii $\mathcal{A}_{PL}$ – vezi figura 2.3. Repetând operația de translație a unei drepte de nivel a funcției obiectiv în sensul maximizării se găsește că soluția optimă întreagă este reprezentată de punctul M cu coordonatele $x_1^0 = 8$, $x_2^0 = 2$; optimul întreg are valoarea $f(x^0) = 4900$.

2.2 Programarea liniară vs programarea în numere întregi

Reluăm exemplul din secțiunea precedentă şi comparăm mulțimile $\mathcal{A}_P$ şi $\mathcal{A}_{PL}$ reprezentate simultan în figura 2.3. Putem trage următoarele concluzii, valabile şi în cazul general.

1) $\mathcal{A}_{PL}$ are proprietatea că fiecare punct al său este „înconjurat” de alte puncte din mulțime, **oricât de apropiate**. Ca urmare, teoria clasică a optimizării, bazată, după cum se ştie, pe posibilitatea efectuării unor deplasări infinitesimale în jurul unui punct, este **direct** aplicabilă.
Prin contrast, soluțiile admisibile întregi formează o mulțime **rară:** orice punct din $\mathcal{A}_P$ posedă o vecinătate în care nu se mai află alte puncte din mulțime. Aplicarea teoriei amintite este în acest caz lipsită de sens.

2) $\mathcal{A}_{PL}$ este o mulțime **convexă** şi mai mult **poliedrală** având un număr **finit** de **vârfuri.** Aceste proprietăți, plus liniaritatea funcției obiectiv, ne asigură că soluția optimă fracționară se găseşte într-unul din **vârfuri** (conform teoremei centrale a programării liniare). Cercetarea sistematică a

mulţimii vârfurilor lui $\mathcal{A}_{PL}$ cu ajutorul **algoritmului simplex** conduce la optimul fracţionar într-un număr finit de paşi.

Cu rare excepţii, soluţia optimă întreagă – care este la urma urmei o soluţie admisibilă a problemei relaxate – se găseşte **în interiorul** mulţimii $\mathcal{A}_{PL}$ şi ca urmare nu este nici măcar generată de către algoritmul simplex!

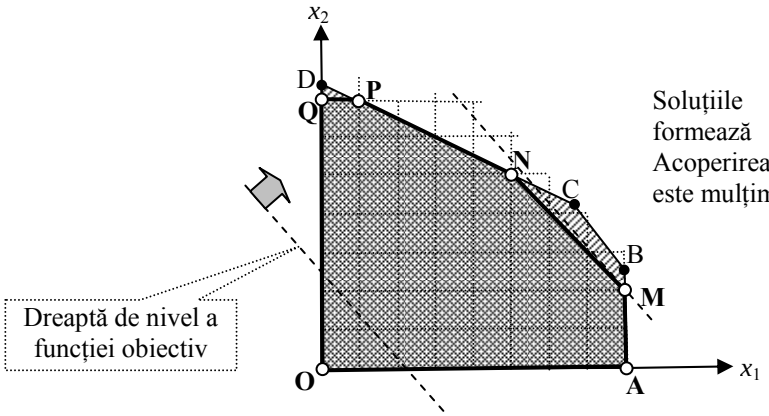


Figura 2.4

3) Să considerăm **anvelopa convexă** Conv($\mathcal{A}_P$) a mulţimii soluţiilor întregi ale programului (P), adică cea mai mică mulţime convexă care conţine $\mathcal{A}_P$. În figura 2.4 această anvelopă este reprezentată de mulţimea dublu haşurată OAMNPQ. Prin construcţie vârfurile anvelopei Conv($\mathcal{A}_P$) sunt soluţii admisibile întregi astfel că, dacă vom maximiza funcţia obiectiv *f* pe această mulţime, vom regăsi soluţia optimă întreagă x^0.

În concluzie, putem rezolva o problemă de programare în numere întregi ca o problemă de programare liniară uzuală cu condiţia să ştim să descriem în limbaj de inecuaţii liniare anvelopa convexă a soluţiilor admisibile întregi!

În cazul studiat, anvelopa Conv($\mathcal{A}_P$) se compune din soluţiile sistemului de inecuaţii:

$$\begin{cases} x_1 + 2x_2 \le 15 \\ x_1 + x_2 \le 10 \\ x_1 \le 8 \\ x_2 \le 7 \\ x_1, x_2 \ge 0 \end{cases}$$

(atenţie: noile inegalităţi $x_1 + x_2 \le 10$ şi $x_2 \le 7$ sunt derivate din dreptele MN şi PQ din figura 2.4!)

În general, descrierea ecuaţională a mulţimii Conv($\mathcal{A}_P$) este posibilă, chiar fără generarea explicită a soluţiilor întregi sau măcar a unora dintre ele. Rezultatul are o valoare pur teoretică întrucât

realizarea efectivă a descrierii este, de cele mai multe ori, imposibil de făcut din cauza volumului imens de calcule.

Totuşi am putea recupera ceva din această idee: adăugând la problema relaxată (PL) un număr (finit) de restricţii suplimentare, astfel alese încât să fie verificate de toate soluţiile întregi, se pot îndepărta din $\mathcal{A}_{PL}$ o serie de porţiuni aşa încât soluţia optimă întreagă să devină vârf în mulţimea rămasă putând fi astfel recunoscută de algoritmul simplex – vezi figura 2.5.

Din acest motiv, noile restricţii se numesc **tăieturi** sau **plane de secţiune**.

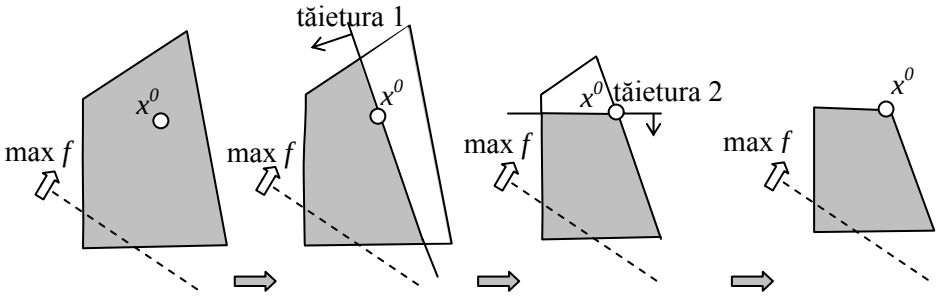


Figura 2.5

Concret, soluţia optimă întreagă va putea fi detectată de către algoritmul simplex după un număr de reoptimizări cu adăugare de restricţii (tăieturi).

În principiu, **orice tăietură se obţine printr-un proces de „rotunjire întreagă" a unei combinaţii liniare a restricţiilor programului original şi eventual a tăieturilor anterior generate**.

Pentru ilustrare, să notăm cu E_1,E_2,E_3 restricţiile programului studiat mai sus.Combinaţia $E_1 + E_2$ produce inegalitatea $7x_1 + 7x_2 \le 75 \Leftrightarrow x_1 + x_2 \le 10\frac{5}{7}$ verificată de orice soluţie admisibilă a programului relaxat(PL) şi în particular de toate soluţiile întregi ale lui (P). Este clar că inegalitatea $x_1 + x_2 \le 10 = \left\lfloor 10\frac{5}{7} \right\rfloor$ va fi verificată de toate soluţiile întregi ale lui (P) dar nu şi de toate soluţiile admisibile ale programului relaxat (de exemplu soluţia optimă fracţionară $x^* = (6\frac{3}{7}, 4\frac{2}{7})$ nu verifică). Prin urmare inegalitatea $x_1 + x_2 \le 10$ este o tăietură. Analog combinaţia redusă la restricţia E_2 şi rescrisă echivalent $\frac{1}{2}x_1 + x_2 \le 7\frac{1}{2}$ conduce prin rotunjire la tăietura $x_2 \le 7$.

De remarcat că cele două tăieturi construite, împreună cu restricţiile originale definesc complet acoperirea convexă Conv($\mathcal{A}_P$) = OAMNPQ şi reduc rezolvarea programului întreg (P) la rezolvarea programului liniar uzual:

$$\begin{cases} (\max) f = 500x_1 + 450x_2 \\ 6x_1 + 5x_2 \le 60 \\ x_1 + 2x_2 \le 15 \\ x_1 \le 8 \\ x_1 + x_2 \le 10 \\ x_2 \le 7 \\ x_1, x_2 \ge 0 \end{cases} \Leftrightarrow \begin{cases} (\max) f = 500x_1 + 450x_2 \\ \\ x_1 + 2x_2 \le 15 \\ x_1 \le 8 \\ x_1 + x_2 \le 10 \\ x_2 \le 7 \\ x_1, x_2 \ge 0 \end{cases}$$

(inegalitatea $6x_1 + 5x_2 \le 60$ a fost eliminată fiind, în mod clar, o consecință a inegalității $x_1 + x_2 \le 10$!) În general procesul de obținere a tăieturilor descris mai sus este „stufos" şi conține multe „redundanțe". De exemplu, combinația $E_1 + E_2 + E_3 \Leftrightarrow 8x_1 + 7x_2 \le 83 \Leftrightarrow \frac{8}{7}x_1 + x_2 \le \frac{83}{7}$ implică tăietura $\left\lfloor \frac{8}{7} \right\rfloor x_1 + x_2 \le \left\lfloor \frac{83}{7} \right\rfloor \Leftrightarrow x_1 + x_2 \le 11$, care decurge din inegalitatea $x_1 + x_2 \le 10$ obținută anterior; în fapt noua relație, deşi este verificată de toate soluțiile întregi nu este o tăietură „veritabilă" fiind satisfăcută şi de toate soluțiile programului relaxat (PL)! Tot aşa, inegalitatea $x_2 \le 7$, deşi este o tăietură „efectivă" nu este de nici un folos în optimizare aşa cum rezultă din figurile 2.3 şi 2.4.

4) În cazul studiat $\mathcal{A}_P$ era o mulțime **finită**. Acest lucru se întâmplă şi în situații mai generale; de exemplu o problemă de programare cu *n* variabile bivalente nu poate avea mai mult de 2^n soluții admisibile întregi. Fără a influența generalitatea concluziilor, se poate presupune că orice program întreg are un număr finit de soluții întregi.

Aceste constatări ne vor permite să înțelegem mai bune fundamentele, performanțele şi limitele metodelor de rezolvare a programelor întregi ce vor fi prezentate în următoarea unitate de învățare.

2.3 Dificultăți în studiul şi rezolvarea programelor întregi

În general, programele întregi sunt mult mai greu de rezolvat decât programele liniare uzuale. Astfel, dacă în prezent a devenit o obişnuință rezolvarea unor programe liniare cu mii de restricții şi zeci de mii de variabile (continui) soluționarea unui program cu mai puțin de 100 de variabile întregi poate cauza mari dificultăți.

Mai precis, principalul neajuns al tuturor metodelor de rezolvare **exactă** a programelor întregi îl constituie **comportamentul impredictibil** pe probleme de dimensiuni apropiate: este posibil ca o problemă să fie rezolvată „repede" în timp ce alta, de aceleaşi dimensiuni sau chiar mai „mică" să nu cedeze într-un interval de timp rezonabil! Experiența de calcul acumulată până în prezent arată că în programarea liniară uzuală acest fenomen al impredictibilității este extrem de rar: algoritmul simplex „depune" cam acelaşi efort de calcul pe probleme asemănătoare ca dimensiune.

În multe situații practice se poate recurge la următoarea **schemă de rezolvare aproximativă a unui program întreg prin rotunjire**.

Ignorăm cerința de integritate şi rezolvăm programul relaxat (care este un program liniar uzual). Două situații sunt posibile:

1) variabilele întregi originale au valori optime întregi; dacă este aşa atunci soluția optimă a programului relaxat este şi soluția optimă a programului întreg dat.

2) unele variabile întregi au valori fracționare în soluția optimă a programului relaxat. În acest caz aceste componente vor fi rotunjite fie superior fie inferior la valori întregi în ideea că soluția optimă întreagă este situată în „apropierea" soluției optime „fracționare". Evident, operația de rotunjire trebuie astfel făcută încât rezultatul să fie o soluție **admisibilă**, adică să verifice restricțiile problemei! În final se acceptă soluția obținută prin rotunjire ca **soluție suboptimală** a programului întreg dat. Adeseori în practică acest lucru este justificat prin faptul că valorile permise variabilelor sunt suficient de mari astfel încât rezultatul rotunjirii să fie **neglijabil**.

Dezavantajele schemei sunt aproape evidente:

- Frecvent numărul posibilităților de rotunjire este foarte mare, implicând un volum apreciabil de calcule pentru verificarea şi sortarea lor.
- Impedimentul major apare la programele în care plaja de valori permise variabilelor întregi este mică cum este cazul variabilelor bivalente. De multe ori este posibil ca „distanța" dintre soluția optimă întreagă şi cea fracționară să fie aşa de mare încât simpla rotunjire să nu conducă nici măcar la soluții acceptabile.

Exemplul 1 Reluăm programul întreg (P) rezolvat grafic în secțiunea 2.1. Soluția optimă fracționară $x_1^* = 6\frac{3}{7}, x_2^* = 4\frac{2}{7} \quad f(x^*) = 5142\frac{6}{7}$ nu are un sens economic coerent deoarece x_1 şi x_2 reprezintă cicluri de fabricație care odată începute nu pot fi întrerupte până la sfârşit!

Există patru posibilități de rotunjire:

$$x_1 = 7, x_2 = 5; \quad x_1 = 7, x_2 = 4; \quad x_1 = 6, x_2 = 5; \quad x_1 = 6, x_2 = 4$$

dintre care numai ultima este admisibilă aducând un profit de $6 \cdot 500 + 4 \cdot 450 = 4800\,\$$.O cale mai bună ar consta în rotunjirea valorii uneia dintre variabile şi găsirea „celei mai bune" valori pentru cea de a doua; ar rezulta combinațiile admisibile $x_1 = 7, x_2 = 3$ cu profitul 4850$ şi $x_1 = 5, x_2 = 5$ cu profitul 4750$.

Toate aceste combinații rezultate din rotunjire sunt inferioare totuşi veritabilei soluții optime întregi: $x_1^0 = 8, x_2^0 = 2 \quad f(x^0) = 4900\,\$$.

Exemplul 2 Considerăm următoarea problemă de alegere a unor proiecte de investiții:

$(\max) f = 4x_1 + 6x_2 + 7x_3 + 16x_4 \quad \leftarrow$ valoarea netă prezentă a profitului(milioane lei);

$16x_1 + 35x_2 + 45x_3 + 85x_4 \le 100$ ← restricția de încadrare în capitalul disponibil(mil.lei);
$x_j \in \{0,1\}$

Relaxatul acestui program bivalent este programul liniar uzual:

$$\begin{cases} (\max) f = 4x_1 + 6x_2 + 7x_3 + 16x_4 \\ 16x_1 + 35x_2 + 45x_3 + 85x_4 \le 100 \\ 0 \le x_j \le 1 \quad j = 1,...,4 \end{cases}$$

a cărui soluție optimă este:

$$x_1^* = 1 \quad x_2^* = 0 \quad x_3^* = 0 \quad x_4^* = 0.988 \qquad f(x^*) = 19.8118$$

Dacă $x_1 = 1$ înseamnă acceptarea proiectului 1 iar $x_2 = 0$ înseamnă respingerea proiectului 2, $x_4 = 0.988$ nu are absolut nici o semnificație, spre deosebire de $x_1 = 6\frac{3}{7}$ din exemplul precedent care tot înseamnă „ şase cicluri şi ceva”!!
Rotunjirea superioară a lui x_4 nu conduce la o soluție admisibilă în timp ce rotunjirea inferioară duce la soluția:

$$x_1 = 1 \quad x_2 = x_3 = x_4 = 0 \qquad f(x) = 4$$

„departe” de soluția optimă întreagă:

$$x_1^0 = 1 \quad x_2^0 = 1 \quad x_3^0 = 1 \quad x_4^0 = 0 \qquad f(x^0) = 17$$

Exemplul 2 este interesant şi din alt punct de vedere: dacă se măreşte capitalul disponibil cu numai 1%, adică de la 100 la 101 milioane lei soluția optimă întreagă se modifică radical:

$$x_1^{00} = 1 \quad x_2^{00} = 0 \quad x_3^{00} = 0 \quad x_4^{00} = 1 \qquad f(x^{00}) = 20$$

Valoarea funcției obiectiv creşte cu 3 milioane lei reprezentând 17.6% din câştigul anterior!!(orice manager cu scaun la cap ar mări capitalul disponibil cu un milion pentru a obține de trei ori mai mult!)

Dacă ne referim la programul relaxat, soluția optimă a problemei modificate coincide cu soluția optimă întreagă amintită, astfel că creşterea funcției obiectiv este de numai 20 – 19.8118 = 0.1882 mil. reprezentând 0.95% din optimul inițial. Se confirmă concluzia că în programarea liniară uzuală la variații **mici** ale constantelor modelului corespund variații **mici** ale optimului!

Acest exemplu a arătat că optimul unui program întreg poate fi **extrem de sensibil** la variații mici ale constantelor programului aşa încât practicienii recomandă rezolvarea lui de mai multe ori, cu mici modificări în mărimile constante, în încercarea de a obține „cea mai bună” soluție optimă ce poate fi acceptată pentru implementarea în practică.

Probleme propuse

1. Se consideră programele întregi:

$$a)\quad (P)\begin{cases} 2x_1 + 2x_2 \le 9 \\ 3x_1 + x_2 \le 11 \\ x_1, x_2 \ge 0 \text{ intregi} \\ (\max) \mathrm{f} = 5\mathrm{x}_1 + 2x_2 \end{cases} \qquad b)\quad (P)\begin{cases} -x_1 + 2x_2 \le 2 \\ x_1 - x_2 \le 2 \\ 2x_1 + 2x_2 \le 7 \\ x_1, x_2 \ge 0 \text{ intregi} \\ (\max) f = 3x_1 + 5x_2 \end{cases} \qquad c)\quad (P)\begin{cases} -3x_1 + 8x_2 \le 19 \\ 6x_1 + 3x_2 \le 17 \\ x_1, x_2 \ge 0 \text{ intregi} \\ (\max) f = 3x_1 + 2x_2 \end{cases}$$

Pentru fiecare din ele:

i) Scrieți programul relaxat (PL) şi reprezentați grafic mulțimea $\mathcal{A}_{PL}$ formată din soluțiile admisibile ale acestuia. Recapitulați principalele proprietăți ale mulțimii $\mathcal{A}_{PL}$. Determinați soluția optimă fracționară x^*.

ii) În același desen, puneți în evidență soluțiile admisibile întregi ale programului (P). Ce deosebiri observați între mulțimea $\mathcal{A}_P$ a soluțiilor întregi şi mulțimea $\mathcal{A}_{PL}$ a soluțiilor admisibile ale programului relaxat? Determinați soluția optimă întreagă x^0.

iii) Puneți în evidență acoperirea convexă Conv($\mathcal{A}_P$) a mulțimii $\mathcal{A}_P$. Încercați să scrieți Conv($\mathcal{A}_P$) ca mulțime de soluții ale unui sistem de inecuații liniare.Generați tăieturi după modelul dat în text. Reduceți rezolvarea programului întreg (P) la rezolvarea unui program liniar uzual.

Unitatea de învățare 3

PROGRAMAREA ÎN NUMERE ÎNTREGI
Metode de rezolvare exactă a programelor întregi

Cuprins

3.1 Generalități privind metodele de rezolvare a programelor întregi
3.2 Metoda Branch and Bound – descriere de principiu
3.3 Aplicarea metodei B&B. Exemple numerice

Probleme propuse

Obiectivele unității de învățare 3

Rezolvarea unui program întreg este o chestiune dificilă mai cu seamă în cazul în care dimensiunile acestuia – număr de restricții, număr de variabile – sunt mari ca şi atunci când programul este "puternic structurat". În unele contexte concrete este suficientă rezolvarea programului relaxat, operație care de regulă este mult mai uşor de făcut, şi rotunjirea rezultatului la o soluție întreagă acceptabilă (suboptimală). Există totuşi destule situații în care cunoaşterea soluției optime întregi este imperios necesară. Iată de ce în această unitate de învățare se propun următoarele obiective:

- prezentarea la nivel de principiu a unor clase de metode de rezolvare exactă a programelor întregi;
- descrierea efectivă a metodei Branch and Bound, des utilizată în programele de calculator;
- ilustrarea numerică a aplicării metodei B&B.

3.1 Generalităţi privind metodele de rezolvare a programelor întregi

Din punctul de vedere al teoriei complexităţii, programele întregi fac parte din clasa problemelor „grele".Aceasta nu înseamnă că ele sunt cu totul intractabile. În dimensiune „relativ moderată", aceste probleme pot fi rezolvate exact şi în timp rezonabil. Folosind metode de căutare din ce în ce mai sofisticate au putut fi rezolvate cu succes şi probleme de dimensiuni apreciabile. Totuşi nu se poate spune că metodele exacte – oricât de sofisticate ar fi ele – sunt în stare să facă faţă oricărei probleme şi mai mult pot avea comportamente radical diferite pe probleme asemănătoare şi de aceeaşi talie!

Faptul că un program întreg are (sau poate fi făcut să aibe) un număr finit de soluţii sugerează şi cel mai simplu mod de rezolvare bazat pe **enumerarea** totală sau parţială a acestor soluţii. Enumerarea totală a fost evocată doar ca posibilitate pentru că, deşi este finit, numărul soluţiilor întregi ce trebuie generate şi verificate poate fi (mai cu seamă în aplicaţiile practice) „excesiv" de mare.

Schemele de **enumerare parţială** determină soluţia optimă întreagă generând efectiv doar o parte a mulţimii soluţiilor întregi – de dorit cât mai mică – soluţiile negenerate fiind recunoscute implicit ca neoptimale. Domeniul predilect al metodelor de enumerare îl constituie programarea **bivalentă** şi un exemplu reprezentativ (în fapt, primul) este algoritmul **aditiv** al lui BALAŞ (1965).

Exceptând enumerarea, **metodele exacte „clasice" reduc rezolvarea unui program întreg (P) la rezolvarea mai multor programe liniare uzuale cu ajutorul procedurii simplex primală sau duală.**

Astfel, metodele de tip **„plane de secţiune"** generează o secvenţă finită de programe liniare $PL \equiv PL_0, PL_1, PL_2, \ldots, PL_t$ primul fiind relaxatul programului (P) şi ultimul având drept soluţie optimă chiar soluţia optimă întreagă căutată. Pentru fiecare $k = 1,2,...,t$ programul (PL_k) se obţine din programul anterior (PL_{k-1}) prin adăugarea unei **tăieturi** a cărei construcţie diferă de la metodă la metodă. Elementul comun al acestor proceduri este observaţia potrivit căreia soluţia optimă întreagă va fi recunoscută de algoritmul simplex dacă şi numai dacă ea va fi – într-o problemă extinsă – o soluţie de bază cu trei proprietăţi: să fie primal admisibilă (≡ să aibe componentele nenegative), să fie dual admisibilă (≡ să verifice criteriul de optim al algoritmului simplex) şi să aibe toate componentele întregi. Toate metodele bazate pe plane de secţiune construiesc tăieturile de aşa manieră încât soluţiile optime ale programelor (PL_k) să menţină două din cele trei proprietăţi, „forţând" îndeplinirea proprietăţii restante.

În algoritmul **ciclic** al lui GOMORY (1958) tăieturile conservă primal şi dual admisibilitatea în timp ce algoritmul **discret** (GOMORY, 1963) menţine dual admisibilitatea şi integritatea soluţiilor intermediare. Ambele proceduri conservă deci dual admisibilitatea şi ca urmare rezolvarea fiecărui program (PL_k) $k = 1,2,...,t$ se face prin **reoptimizarea** programului precedent – extins cu tăietura ataşată – cu algoritmul simplex **dual**.
Algoritmul lui YOUNG şi GLOVER (1972) conservă primal admisibilitatea şi integritatea soluţiilor intermediare folosind pentru reoptimizare algoritmul simplex **primal**.

Deşi în teorie metodele amintite determină soluţia optimă întreagă într-un număr finit de paşi, în practică ele s-au dovedit lente, instabile şi mai cu seamă impredictibile în comportament pe

probleme de aceleaşi dimensiuni. Meritul lor major este acela de a fi impulsionat cercetarea în domeniul **teoriei poliedrale** şi al căutării de tăieturi mai restrictive care să exploateze particularitățile structurii problemei de rezolvat.Fie separat, fie în combinație cu alte metode, planele de secțiune sunt utilizate astăzi la soluționarea unei largi varietăți de probleme de optimizare „grele".

Ar fi de amintit în final că tăieturile au fost utilizate pentru prima dată – desigur într-o formă incipientă – de DANTZIG, FULKERSON şi JOHNSON (1954) la rezolvarea unei **probleme a comisvoiajorului** în care se cerea determinarea celui mai scurt traseu de vizitare a 42 de oraşe din SUA.

Branch and Bound, abreviat B&B (LAND, DOIG, 1960; DAKIN, DRIEBECK, 1964; BALAŞ, 1965) este o metodă alternativă de rezolvare a programelor întregi totale sau mixte bazată pe un principiu ce aminteşte vechiul adagiu latin **„divide et impera".** Deoarece problema originală este în general greu de rezolvat „direct", ea este **ramificată**, adică divizată în subprobleme mai mici, cu mai puține soluții întregi. Fiecare subproblemă rezultată din ramificare este **mărginită**, aceasta însemnând calcularea unei valori (margini) care arată cât de „bune" sunt soluțiile subproblemei respective pentru procesul de optimizare. În principiu, fiecare subproblemă este la rândul ei ramificată în subprobleme şi mai mici, afară de cazul în care marginea ataşată arată că, printre soluțiile ei întregi, nu se găseşte cu siguranță soluția optimă întreagă căutată.

3.2 Metoda Branch and Bound – descriere de principiu

Pentru prezentarea metodei vom fixa un program liniar întreg (P) în care funcția obiectiv se **maximizează**. Putem presupune că toți coeficienții funcției obiectiv sunt numere întregi astfel că evaluarea ei în orice soluție întreagă a programului (P) va fi un număr întreg. Admitem că mulțimea soluțiilor admisibile ale programului relaxat (PL) este **mărginită**; de aici va rezulta că (P) are un număr **finit** de soluții întregi. Această ipoteză nu este deloc restrictivă putând fi asigurată în toate aplicațiile practice.

Metoda utilizează o **„listă"** $\mathcal{L}$ în care vor fi înscrise toate programele întregi rezultate din procesul de ramificare. După cum se va vedea, toate programele din lista $\mathcal{L}$ sunt extinderi ale programului original (P) cu limitări impuse unora dintre variabile. Lista este **dinamică** în sensul că, pe parcurs, în ea se vor opera ştergeri şi/sau adăugiri de programe. La start $\mathcal{L} = \{P\}$.

Într-o **„locație"** notată x_{CMB} se va păstra **C**ea **M**ai **B**ună soluție admisibilă întregă găsită în timpul derulării procedurii. Într-o altă locație, z_{CMB} , se va înscrie valoarea funcției obiectiv în soluția din x_{CMB}. La start:

$x_{\mathbf{CMB}} = \{\varnothing\}$, $z_{\mathbf{CMB}} = -\infty$ în problemele de maximizare sau $z_{\mathbf{CMB}} = +\infty$ în cele de minimzare.

Locația $x_{\mathbf{CMB}}$ se poate inițializa şi cu o soluție întreagă particulară, în caz că o asemenea soluție se cunoaşte! Evident $z_{\mathbf{CMB}}$ se va inițializa cu valoarea corespunzătoare a funcției obiectv.

Vom arăta acum cum lucrează metoda.

Pasul 1 Dacă lista $\mathcal{L}$ este vidă procedura se opreşte. Dacă locaţia x_{CMB} este ocupată acolo se va găsi soluţia optimă întreagă a programului (P). Valoarea optimă a funcţiei obiectiv se va lua din z_{CMB}. În caz că locaţia x_{CMB} este vidă, programul (P) nu are soluţii admisibile întregi.

Dacă lista $\mathcal{L}$ este nevidă se trece la:

Pasul 2 Se selectează unul dintre programele înscrise în $\mathcal{L}$, de obicei primul. Programul selectat, notat P_α , **se şterge** din listă (atenţie, lista $\mathcal{L}$ este **ordonată** aşa că de la bun început trebuie convenit locul în care vor fi operate ştergerile şi adăugirile de programe, de exemplu în „capul" listei).

Notă: la prima iteraţie, $P_\alpha \equiv P$ însă pe parcurs indicele α va fi – după cum se va vedea – o succesiune de 1 şi 2.

Pasul 3 Se trece la rezolvarea programului relaxat (PL_α) folosind algoritmul simplex. Sunt posibile următoarele situaţii:

I) (PL_α) este un program **incompatibil**, adică nu are soluţii admisibile; evident nici P_α nu va avea soluţii întregi. Se revine la pasul 1.

II) (PL_α) este un program **compatibil** şi cu siguranţă va avea un **optim finit** notat z_α. Este uşor de văzut că numărul întreg $\lfloor z_\alpha \rfloor$ este o **margine superioară** a funcţiei obiectiv din (P) pe soluţiile întregi ale programului P_α . Două cazuri sunt posibile în continuare:

II_1) $\lfloor z_\alpha \rfloor \leq z_{\mathbf{CMB}}$

În această situaţie nici o soluţie întreagă a lui P_α nu va fi mai „bună" decât soluţia întreagă depozitată în x_{CMB} ! Ca urmare, este inutil să mai cercetăm soluţiile întregi ale programului P_α (vom spune că programul (P_α) este „abandonat"). Se revine la pasul 1.

II_2) $\lfloor z_\alpha \rfloor > z_{\mathbf{CMB}}$

(în acest caz **s-ar putea** ca (P_α) să aibe soluţii întregi mai „bune" decât soluţia din locaţia x_{CMB} !) Fie x^* soluţia optimă a programului (PL_α).Sunt posibile două continuări:

II_{21}) Toate variabilele întregi din programul original au valori întregi în x^* ; evident, x^* va fi o soluţie întreagă **mai bună** decât cea existentă în x_{CMB}, drept care se fac **actualizările**:

$$x_{\mathbf{CMB}} \leftarrow x^* \; ; \; z_{\mathbf{CMB}} \leftarrow f(x^*)$$

după care din nou se revine la pasul 1.

II_{22}) Una sau mai multe variabile **întregi** din programul original au în x^* valori **fracţionare**. Fie x_k una din aceste variabile. Este clar că orice soluţie întreagă a programului P_α va verifica una din inegalităţile **mutual exclusive**:

$$x_k \leq \lfloor x_k^* \rfloor \text{ sau } x_k \geq \lfloor x_k^* \rfloor = \lfloor x_k^* \rfloor + 1$$

(s-a notat cu $\lfloor x \rfloor$ şi $\lceil x \rceil$ **rotunjirea întreagă inferioară** respectiv **superioară** a numărului real x)

Extindem programul întreg (P_α) cu fiecare dintre cele două inegalități, obținând două noi programe întregi:

$$P_{\alpha 1} \equiv \begin{cases} P_\alpha \\ x_k \le \lfloor x_k^* \rfloor \end{cases} \quad \text{şi} \qquad P_{\alpha 2} \equiv \begin{cases} P_\alpha \\ x_k \ge \lceil x_k^* \rceil \end{cases}$$

Noile programe au următoarele proprietăți evidente:

- nu au soluții întregi comune;
- reuniunea soluțiilor lor întregi este mulțimea soluțiilor întregi ale programului (P_α) – vezi figura 3.1

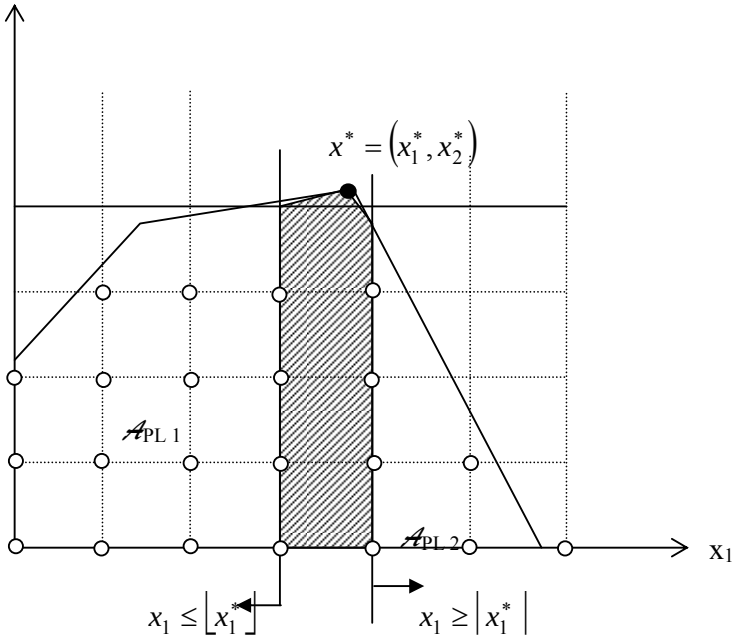


Figura 3.1

(comentariu: prin urmare, cercetarea soluțiilor întregi ale lui P_α se va realiza prin cercetarea separată, în etape ulterioare diferite, a soluțiilor întregi ale programelor $P_{\alpha 1}$ şi $P_{\alpha 2}$. Cercetarea va fi „oarecum" uşurată prin faptul că programele „succesoare" $P_{\alpha 1}$ şi $P_{\alpha 2}$ au fiecare mai puține soluții întregi decât programul „părinte" P_α)

Noile programe ($P_{\alpha 1}$) şi ($P_{\alpha 2}$) se adaugă **în capul listei** $\mathcal{L}$, în locul lui P_α deja şters:

$\mathcal{L}$ = { $P_{\alpha 1}$, $P_{\alpha 2}$, ...}

Se revine la pasul 2.

Observații:

1) Despre programele $P_{\alpha 1}$ şi $P_{\alpha 2}$ vom spune că au rezultat din **ramificarea** programului P_α **după variabila (întreagă)** x_k.
2) În cazul în care ramificarea se poate executa după mai multe variabile este necesar un criteriu de alegere. De exemplu, ramificarea se poate face după **prima** variabilă întreagă cu valoare fracționară în x^* sau după variabila întreagă cu cea mai mare **parte fracționară** în x^*.

3) Procedura descrisă este aplicabilă şi programelor întregi **mixte** cu precizarea că ramificarea, atunci când este necesară, se va face numai după variabilele **întregi**!

Pentru înțelegerea metodei vom vizualiza procesul de ramificare printr-un graf **arbore** T ale cărui noduri sunt **relaxatele** diferitelor programe rezultate din ramificare – vezi figura 3.2 Un nod din T, identificat cu programul liniar (PL_α), va fi un nod **terminal** dacă şi numai dacă programul (P_α) nu a mai fost ramificat şi aceasta se întâmplă în trei situații:

- (PL_α) este un program incompatibil;

sau

- (PL_α) este compatibil dar marginea ataşată arată că nu are soluții întregi mai bune decât soluția existentă în x_{CMB};

sau

- (PL_α) are soluție optimă întreagă mai bună decât cea din x_{CMB}.

Cu excepția **rădăcinii** (PL) fiecare nod din T are un unic **predecesor**. Orice nod care nu este nod terminal are doi **succesori**.

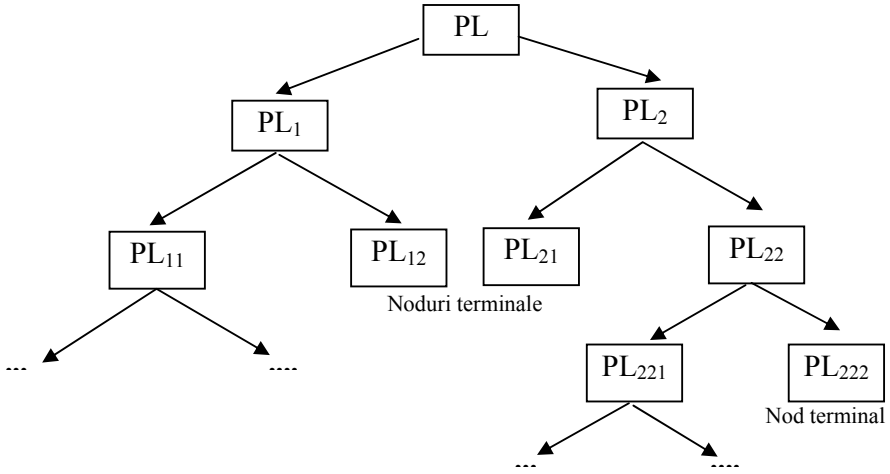


Figura 3.2

Arborele T nu există de la început! La start el se reduce la rădăcina (PL) şi în continuare primeşte noi noduri şi arce de legătură în funcție de problemele rezultate din ramificare şi efectiv rezolvate.

Aceasta ar fi o descriere de principiu a metodei B&B. Desigur, sunt necesare precizări privind modul de alegere al subproblemei ce urmează a fi ramificată sau al variabilei după care se face ramificarea. La fel, manipularea şi stocarea datelor condiționează nemijlocit performanțele metodei. Nu mai puțin importantă s-a dovedit etapa de pregătire a programului de rezolvat în care acesta este simplificat şi reformulat prin eliminarea eventualelor restricții redundante şi a variabilelor a căror valoare optimă poate fi dedusă direct din structura problemei (**preprocesare**). Metoda B&B este astăzi implementată în mai toate programele comerciale destinate rezolvării programelor întregi.

Principalele dezavantaje ale metodei descrise sunt:

- **creşterea foarte rapidă**, chiar explozivă a numărului problemelor de rezolvat şi de aici a volumului de calcule o dată cu creşterea dimensiunilor programului original şi mai cu seamă a numărului de variabile întregi;
- comportare **impredictibilă** pe probleme de dimensiuni apropiate: arborele problemelor rezolvate poate fi extrem de „stufos" pentru o problemă şi foarte simplu pentru o alta.

Branch and Cut este o procedură şi mai performantă, rezultată din combinarea metodei B&B cu metoda planelor de secțiune. Versiunea hibridă a fost utilizată cu succes la rezolvarea unor probleme de tip „comisvoiajor" de dimensiuni impresionante.

3.3 Aplicarea metodei B&B. Exemple numerice

Exemplul 1 Vom determina soluția optimă întreagă a programului

$$(P)\begin{cases}(\max) f = 2x_1 + 5x_2 \\ 2x_1 + 2x_2 \le 9 \\ x_1 + 3x_2 \le 11 \\ x_1, x_2 \ge 0 \\ x_1, x_2 \text{ intregi}\end{cases}$$

folosind metoda B&B descrisă mai sus. Convenim ca în lista $\mathcal{L}$ toate adăugirile şi ştergerile să se opereze **în capul** listei iar variabila după care se face ramificarea unui program (P_α) să fie **prima** variabilă întreagă cu valoare fracționară în soluția optimă a relaxatei (PL_α).

Start Inițializăm:

$x_{CMB} \leftarrow \varnothing$ (locația vidă); $z_{CMB} = -\infty$ (în problemele de maximizare !); $\mathcal{L} = \{ P \}$

Iterația 1 Operații:

- Selectăm programul (P), singurul element al listei $\mathcal{L}$.
- Ştergem P din listă $\rightarrow$ lista $\mathcal{L}$ este acum vidă.
- Cu algoritmul simplex rezolvăm programul relaxat

$$(PL)\begin{cases}(\max) f = 2x_1 + 5x_2 \\ 2x_1 + 2x_2 \le 9 \\ x_1 + 3x_2 \le 11 \\ x_1, x_2 \ge 0\end{cases}$$

(în cazul de față se poate folosi şi metoda grafică) Se găseşte soluția optimă fracționară:

$$x_1^* = 1\frac{1}{4} \quad , \quad x_2^* = 3\frac{1}{4} \quad ; \quad z = f(x^*) = 18\frac{3}{4}$$

- Optimul întreg nu depăşeşte marginea superioară $\left\lfloor z = 18\frac{3}{4} \right\rfloor = 18$ .Condiția de ramificare $\lfloor z \rfloor > z_{CMB} \Leftrightarrow 18 > -\infty$ este banal satisfăcută (suntem abia la început şi încă nu a fost găsită nici o soluție întreagă a programului (P)…). Ramificăm (P) după variabila x_1, **prima** variabilă întreagă cu valoare fracționară în x^*. Aceasta înseamnă înlocuirea programului (P) cu programele întregi:

$$(P_1) \equiv \begin{cases} P \\ x_1 \le 1 \end{cases} \equiv \begin{cases} (\max) f = 2x_1 + 5x_2 \\ 2x_1 + 2x_2 \le 9 \\ x_1 + 3x_2 \le 11 \\ x_1 \le 1 \\ x_1, x_2 \ge 0 \text{ , intregi} \end{cases} \qquad \text{şi} \qquad (P_2) \equiv \begin{cases} P \\ x_1 \ge 2 \end{cases}$$

- Înscriem programele rezultate din ramificare în capul listei $\mathcal{L}$:

$$\mathcal{L} = \{ P_1 , P_2 \}$$

Iterația 2 Operații:

- Selectăm programul (P_1), primul program înscris în lista $\mathcal{L}$.
- Ştergem programul selectat din listă $\rightarrow \mathcal{L} = \{ P_2 \}$.
- Rezolvăm problema relaxată (PL_1); se găseşte soluția optimă, din nou fracționară:

$$x_1^* = 1 \quad , \quad x_2^* = 3\frac{1}{3} \quad ; \quad z_1 = f(x^*) = 18\frac{2}{3}$$

- Deoarece $\lfloor z_1 \rfloor = 18 > -\infty = z_{CMB}$ ramificăm (P_1) după variabila x_2.
- Noile programe:

$$(P_{11}) \equiv \begin{cases} P_1 \\ x_2 \le 3 \end{cases} \qquad \text{şi} \qquad (P_{12}) \equiv \begin{cases} P_1 \\ x_2 \ge 4 \end{cases}$$

vor fi înscrise în capul listei $\mathcal{L}$:

$$\mathcal{L} = \{ P_{11} , P_{12} , P_2 \}$$

Iterația 3 Selectăm programul (P_{11}) şi actualizăm lista $\mathcal{L}$:

$$\mathcal{L} = \{ P_{12} , P_2 \}$$

după care rezolvăm programul relaxat (PL_{11}) ; obținem soluția optimă întreagă:

$$x_1^* = 1 \quad , \quad x_2^* = 3 \quad ; \quad f(x^*) = 17$$

În acest stadiu al derulării algoritmului B&B nu putem şti dacă această soluție este şi soluția optimă a programului original (P). O vom reține ca fiind **C**ea **M**ai **B**ună soluție întreagă găsită până acum:

$$x_{\text{CMB}} \leftarrow (1,3) \qquad z_{\text{CMB}} \leftarrow 17$$

Iterația 4 Acum se va rezolva relaxata programului (P_{12}). Lista programelor întregi de studiat se reduce la:

$$\mathcal{L} = \{ P_2 \}$$

Se găseşte că programul liniar (PL_{12}) este incompatibil.

Iterația 5 Se selectează programul (P_2); lista $\mathcal{L}$ devine vidă. Programul relaxat (PL_2) are soluția optimă fracționară:

$$x_1^* = 2 \quad , \quad x_2^* = 3\frac{1}{2} \quad ; \quad z_2 = f(x^*) = 16\frac{1}{2}$$

Deoarece $\lfloor z_2 \rfloor = 16 < 17 = z_{\text{CMB}}$ nici o soluție întreagă a programului (P_2) nu este "mai bună" decât soluția existentă în x_{CMB}; abandonăm (P_2) şi revenim la consultarea listei $\mathcal{L}$.

Iterația 6 În acest moment $\mathcal{L} = \varnothing$: aceasta înseamnă că toate soluțiile întregi ale programului original (P) au fost cercetate şi că soluția depozitată în x_{CMB} este cea mai bună. În concluzie, soluția optimă întreagă căutată este:

$$x_1^0 = 1 \quad x_2^0 = 3 \qquad f(x^0) = 17$$

Considerațiile precedente sunt sintetizate în graful arbore T din figura 3.3; nodurile lui T sunt cele cinci programe liniare efectiv rezolvate cu algoritmul simplex în ordinea PL, PL_1, PL_{11} , PL_{12} şi PL_2.

După cum s-a mai spus, arborele T se construieşte **progresiv** pe măsura derulării algoritmului.La start el se reduce la "rădăcina" (PL) şi în continuare primeşte noi noduri şi arce corespunzătoare programelor rezultate din ramificare.

Un nod (PL_α) – unde α este o succesiune de 1 şi 2 – corespunzător unui program întreg (P_α) ramificat, are doi succesori ($PL_{\alpha 1}$) şi ($PL_{\alpha 2}$) ce corespund relaxatelor programelor întregi:

$$(P_{\alpha 1}) \equiv \begin{cases} (P_\alpha) \\ x_j \le \lfloor x_j^* \rfloor \end{cases} \qquad \text{şi} \qquad (P_{\alpha 2}) \equiv \begin{cases} (P_\alpha) \\ x_j \ge \lceil x_j^* \rceil \end{cases}$$

rezultate din ramificarea după variabila x_j (reamintim că x^* este soluția optimă a programului relaxat (PL_α) în care x_j^* este **prima** componentă fracționară!)

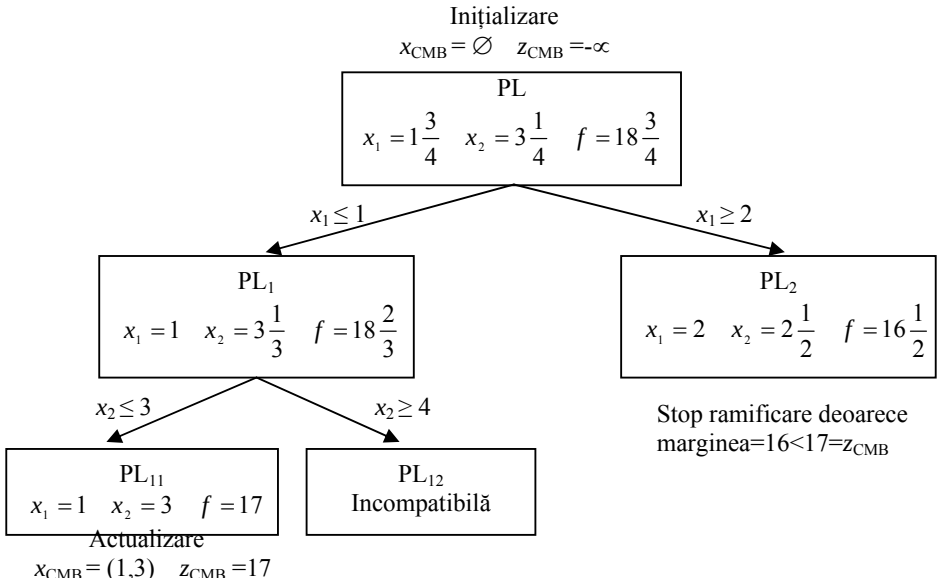


Figura 3.3

Am putea spune că, o dată stabilită oportunitatea ramificării programului (P_α) sau, altfel spus, a nodului (PL_α), există **"două direcții de înaintare"** în arborele T: una **"spre stânga"** către nodul ($PL_{\alpha 1}$), şi cealaltă **"spre dreapta"** către nodul ($PL_{\alpha 2}$). Ca "regulă de înaintare", întotdeauna se va merge mai întâi "spre stânga", către ($PL_{\alpha 1}$) şi apoi, într-o etapă ulterioară şi nu neapărat imediată, pe cealaltă direcție către ($PL_{\alpha 2}$).

Examinarea direcțiilor de înaintare "spre dreapta" sugerează **"întoarceri"** dintr-un nod (PL_α) către unicul predecesor. "Pasul înapoi" se face atunci când din nodul (PL_α) "nu se mai poate înainta" şi aceasta se întâmplă numai dacă:

- Programul (P_α) nu mai poate fi ramificat;

sau dacă

- Ambele direcții de înaintare din nodul (PL_α) au fost examinate.

Considerațiile precedente justifică următoarea clasificare a nodurilor arborelui T, clasificare care depinde de stadiul construcției arborelui:

- noduri **active** ≡ noduri din care se poate face o "înaintare" în T pe una din cele două direcții posibile;
- noduri **moarte ≡** noduri din care înaintarea nu mai este posibilă fiind necesară "întoarcerea" în nodul predecesor.

Cu aceste precizări de terminologie procedura B&B se opreşte când rădăcina (PL) a arborelui T este declarată nod mort.

Utilizarea limbajului introdus permite o descriere foarte sugestivă a modului în care „a acționat" metoda B&B în ilustrarea numerică prezentată.

Astfel, în iterația 1, din nodul ramificat (PL) „am înaintat spre stânga" către succesorul (PL_1) – **mişcarea 1**. La iterația 2 am mai făcut o „înaintare spre stânga" către succesorul (PL_{11}) – **mişcarea 2**. Nodul (PL_{11}) nu a mai fost ramificat deoarece a produs o soluție întreagă (ca urmare a fost declarat nod mort). Atunci „ne-am întors" în unicul predecesor (PL_1) – **mişcarea 3** – şi „am înaintat spre dreapta" către al doilea succesor (PL_{12}) – **mişcarea 4**. Din nou ramificarea nu a mai fost posibilă din cauza incompatibilității programului (PL_{12}); nodul (PL_{12}) a fost declarat mort „Ne-am întors" iarăşi în (PL_1) –**mişcarea 5** – şi constatând că ambele direcții de înaintare din (PL_1) au fost examinate am mai făcut un „pas înapoi" –**mişcarea 6** – către predecesorul (PL), nodul (PL_1) fiind şi el declarat mort. Abia acum am cercetat a doua direcție de înaintare din (PL), cea care duce la succesorul (PL_2) – **mişcarea 7**. Aici am realizat că este inutilă continuarea ramificării (declarând nodul (PL_2) mort), drept care „ne-am întors" din nou în rădăcina (PL) – **mişcarea 8** –cu concluzia că am cercetat toate soluțiile întregi ale programului dat. Cele cinci noduri au fost declarate moarte în ordinea (PL_{11}), (PL_{12}), (PL_1), (PL_2), (PL). Problemele de programare liniară au fost rezolvate în ordinea (PL), (PL_1), (PL_{11}), (PL_{12}), (PL_2).

Exemplul 2 Rezultatul aplicării metodei B&B la rezolvarea programului întreg:

$$(P)\begin{cases}(\max) f = 20x_1 + 11x_2\\ 240x_1 + 145x_2 \le 696\\ x_1, x_2 \ge 0,\ \text{intregi}\end{cases}$$

este sintetizat în arborele din figura 3.4 A fost necesară rezolvarea a 11 programe liniare în ordinea:

$$PL\ ,\ PL_1\ ,\ PL_{11}\ ,\ PL_{12}\ ,\ PL_{121}\ ,\ PL_{1211}\ ,\ PL_{1212}\ ,\ PL_{12121}\ ,\ PL_{12122}\ ,\ PL_{122}\ ,\ PL_2$$

pentru a obține soluția optimă $x_1^0 = 1$, $x_2^0 = 3 \quad f(x^0) = 53$.
(cititorul este invitat să reconstituie conținutul celor 12 iterații). Se observă că, deşi acest program este „mai mic" decât programul întreg studiat în exemplul 1 (are o singură restricție faşă de două câte avea cel de dinainte), efortul de calcul a fost mai mare. Se confirmă faptul că metoda B&B – ca de altfel orice altă metodă exactă de rezolvare a programelor întregi – are un comportament ce nu poate fi anticipat!

Exemplul 3 Aşa cum s-a specificat deja în finalul secțiunii 3.1, combinarea metodei B&B cu „generarea de tăieturi" conduce, în general, la accelerarea rezolvării programelor întregi. Pentru ilustrare reluăm programul (P) din exemplul 2, pe care, mai întâi, îl vom „extinde" cu câteva tăieturi uşor de construit. Reamintim că o tăietură este o restricție suplimentară care elimină o parte dintre soluțiile neîntregi conservându-le însă pe cele întregi.(vezi indicațiile date în secțiunea 2.2 a unității de învățare 2!)

Înmulțim unica restricție din (P) cu scalarul $\frac{1}{145}$: $\frac{240}{145}x_1 + x_2 \le \frac{696}{145} \quad \Leftrightarrow \quad \left(1\frac{95}{145}\right)x_1 + x_2 \le 4\frac{116}{145}$.

Rotunjind inferior coeficienții fracționari rezultă tăietura $x_1 + x_2 \le 4$ (de ce?).

Înmulțim acum restricția originală cu scalarul $\frac{1}{240}$: $x_1 + \frac{145}{240}x_2 \le \frac{696}{240} = 2\frac{216}{240}$.Din nou rotunjim inferior coeficienții neîntregi; obținem o altă tăietură $x_1 \le 2$.

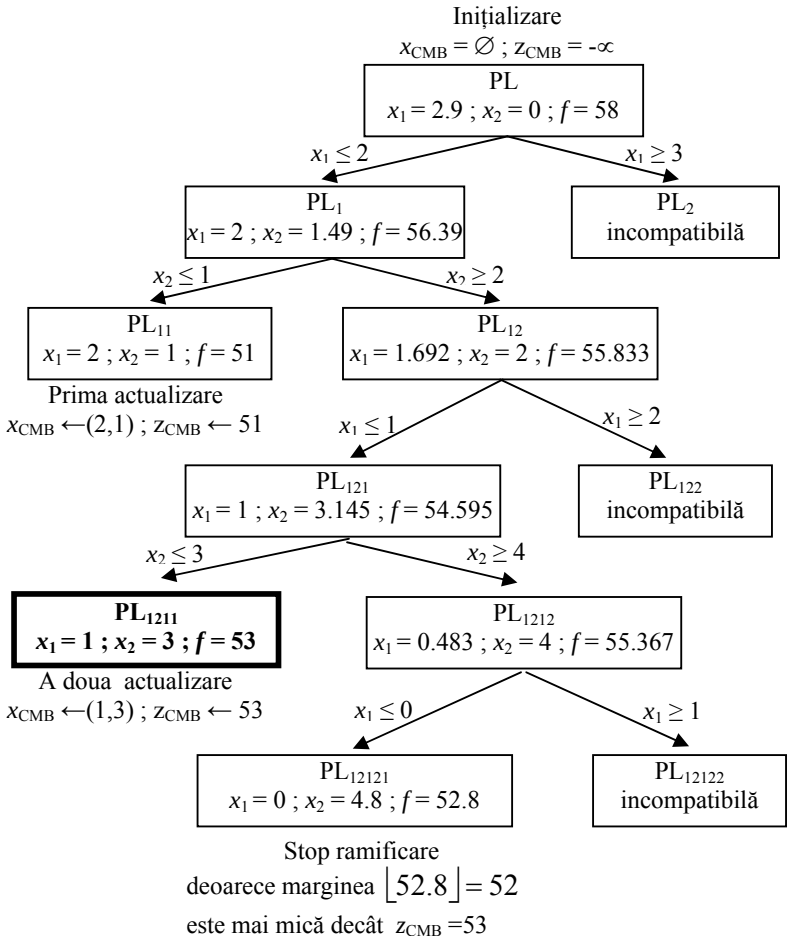


Figura 3.4

Prin construcție, programul extins:

$$(P')\begin{cases}(\max) f = 20x_1 + 11x_2 \\ 240x_1 + 145x_2 \le 696 \\ x_1 + x_2 \le 4 \\ x_1 \le 2 \\ x_1, x_2 \ge 0, \text{ intregi}\end{cases}$$

va avea aceleaşi soluții întregi ca şi (P) şi în particular aceeaşi soluție optimă întreagă. Aplicarea metodei B&B la programul (P') conduce „mai repede" la optim: din arborele dat în figura 3.5, rezultă că a fost nevoie să fie rezolvate numai 5 programe liniare față de 11 câte sunt în arborele programului original (P), vizualizat în figura 3.4!

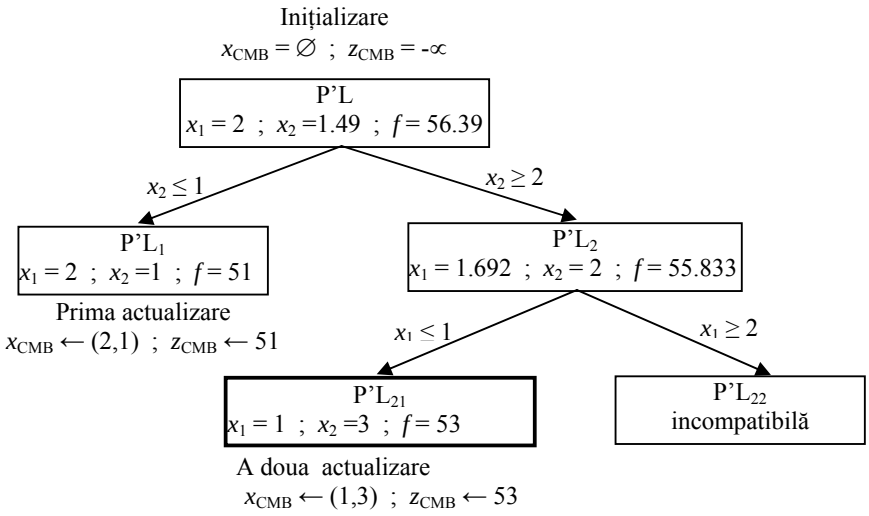


Figura 3.5

Probleme propuse

1. Recapitulați metoda B&B şi enumerați situațiile în care un program (P_α) din lista $\mathcal{L}$ **nu se mai ramifică** (echivalent, nodul (PL_α) este nod **terminal** în arborele programelor liniare efectiv rezolvate)

2. Ce modificări trebuie aduse procedurii B&B pentru a obține **toate** soluțiile optime întregi (în caz că sunt mai multe…)

3. Să se adapteze metoda B&B la rezolvarea problemelor în care funcția obiectiv se **minimizează**. Să se aplice metoda B&B la rezolvarea programului:

$$(P)\begin{cases}(\min) f = 14x_1 + 12x_2 \\ 35x_1 + 24x_2 \ge 107 \\ x_1, x_2 \ge 0 \ \text{ intregi}\end{cases}$$

(vezi "problema fermierului" din secțiunea 1.1 unitatea de învățare 1)

4. Rezolvarea unui program întreg prin metoda B&B a necesitat aplicarea algoritmului simplex programelor liniare indicate în arborele din figura 3.6. **În ce ordine** au fost rezolvate aceste programe dacă, de fiecare dată când un program (P_α) din lista $\mathcal{L}$ a fost ramificat, cele două noi programe ($P_{\alpha 1}$) şi ($P_{\alpha 2}$) au fost înscrise:

i) ambele **în capul** listei $\mathcal{L}$ în ordinea $P_{\alpha 1}$, $P_{\alpha 2}$ (regula uzuală);
ii) ambele **în coada** listei $\mathcal{L}$ în ordinea $P_{\alpha 1}$, $P_{\alpha 2}$;
iii) $P_{\alpha 1}$ **în capul** listei $\mathcal{L}$ iar $P_{\alpha 2}$ **în coada** acesteia.

Se reaminteşte că, în ceeace priveşte consultarea listei $\mathcal{L}$ (care are loc la începutul fiecărei iterații), programul care se selectează – şi se şterge din listă – este cel situat în capul listei!

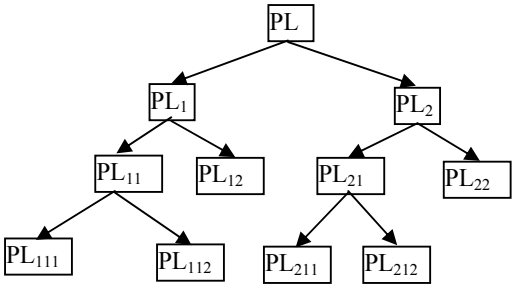


Figura 3.6

5. Rezolvarea cu metoda B&B a unui program liniar în numere întregi (P), în care funcția obiectiv se **maximizează**, a necesitat rezolvarea programelor liniare indicate în arborele din figura 3.7

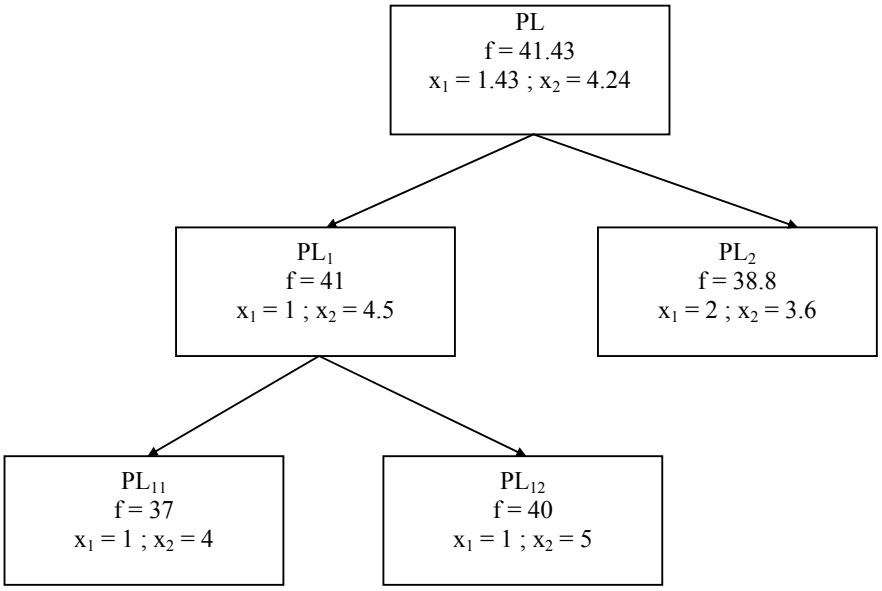


Figura 3.7

a) Precizați soluția optimă fracționară x^* şi optimul fracționar $f(x^*)$ precum şi soluția optimă întreagă x^0 şi optimul întreg $f(x^0)$;
b) Indicați pe arcele arborelui restricțiile după care s-a făcut ramificarea;
c) În ce ordine au fost rezolvate cele cinci probleme? Se va presupune că toate actualizările listei $\mathcal{L}$ s-au operat **în capul** acesteia!
d) De ce nu s-a continuat ramificarea din nodul (PL_2)?

6. Se consideră programele întregi:

$$a)\begin{cases} 2x_1 + 2x_2 \le 9 \\ 3x_1 + x_2 \le 11 \\ x_1, x_2 \ge 0 \text{ intregi} \\ (\max) f = 5x_1 + 2x_2 \end{cases} \qquad b)\begin{cases} -x_1 + 2x_2 \le 2 \\ x_1 - x_2 \le 2 \\ 2x_1 + 2x_2 \le 7 \\ x_1, x_2 \ge 0 \text{ intregi} \\ (\max) f = 3x_1 + 5x_2 \end{cases} \qquad c)\begin{cases} -3x_1 + 8x_2 \le 19 \\ 6x_1 + 3x_2 \le 17 \\ x_1, x_2 \ge 0 \text{ intregi} \\ (\max) f = 3x_1 + 2x_2 \end{cases}$$

Rezolvați prin metoda Branch & Bound programele date (în rezolvarea programelor liniare generate de metodă puteți folosi metoda grafică). Pentru fiecare caz, construiți arborele T al programelor liniare efectiv rezolvate.

7. Folosiți utilitarul QM pentru rezolvarea programului întreg:

$$\begin{cases} (\max)\ x_4 \\ 8x_1 + 5x_2 + 3x_3 \le 100 \\ 6x_1 + 9x_2 + 8x_3 \le 200 \\ 7x_1 + 6x_2 + 8x_3 - 4x_4 \ge 0 \\ 5x_1 + 9x_2 + 4x_3 - 3x_4 \ge 0 \\ x_1, x_2, x_3, x_4 \ge 0 \text{ intregi} \end{cases}$$

Reconstituiți arborele programelor liniare rezolvate pe baza informațiilor furnizate de utilitar.

Unitatea de învățare 4

INTRODUCERE ÎN OPTIMIZAREA COMBINATORIALĂ
Obiectul optimizării combinatoriale

Cuprins

4.1 Elemente de teoria grafurilor
4.2 Ce este optimizarea combinatorială
4.3 Alte probleme reprezentative de optimizare combinatorială

Probleme propuse
Bibliografie

Obiectivele unității de învățare 4

- Recapitularea unor noțiuni din teoria grafurilor;
- Evidențierea – prin exemple – a particularităților optimizării combinatoriale;
- Formalismul problemelor de optimizare combinatorială;
- Optimizarea combinatorială, sursă a programării în numere întregi;
- Prezentarea unor probleme reprezentative ce vor fi studiate în următoarele secțiuni.

4.1 Elemente de teoria grafurilor

Pentru înțelegerea corectă şi clară a problematicii expuse în această unitate de învățare recomandăm cititorului să studieze şi să-şi însuşească minimul de cunoştințe despre grafuri dat în această secțiune (firește, dacă este cazul; cei avizați pot trece direct la următoarea secțiune).

A) Conceptul de graf

În esență, un graf este un desen compus dintr-un număr **finit** de **puncte** şi de **linii** care unesc unele dintre aceste puncte.

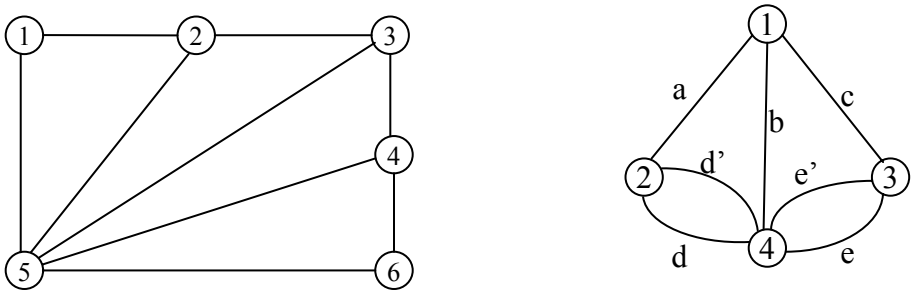


Figura 4.1

Punctele se numesc de obicei **noduri** sau **vârfuri** în timp ce liniile se numesc **muchii**. Dacă se notează cu V mulțimea nodurilor şi cu E mulțimea muchiilor, graful va fi notat prin sigla G = (V, E).

Graful G se va numi **simplu** dacă oricare două vârfuri sunt extremități pentru cel mult o muchie, altminteri se va numi **multigraf**.Graful din figura 4.1a) este simplu; graful din figura 4.1b) este un multigraf.Într-un graf simplu orice muchie este perfect determinată de extremitățile sale.

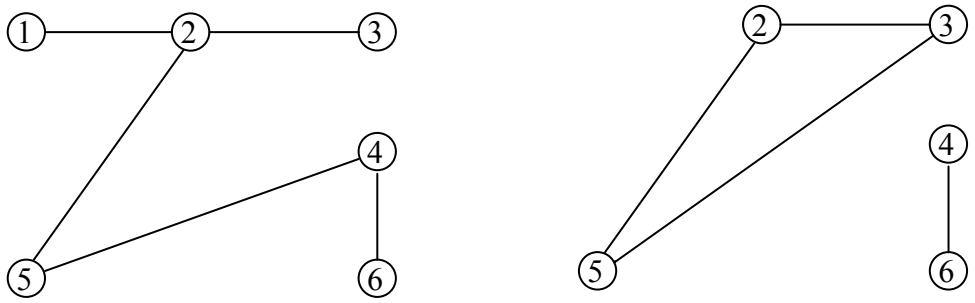
Două **noduri** din graful G se vor zice **adiacente** dacă sunt extremitățile unei muchii; două **muchii** se vor numi adiacente dacă au o extremitate în comun.

Graful G'=(V',E') se va numi **subgraf** al grafului G=(V,E) dacă $V' \subseteq V$, $E' \subseteq E$ şi orice muchie din E'are aceleaşi extremități în G şi G'. În figura 4.2 sunt date două subgrafuri ale grafului din figura 4.1a); cel din stânga are aceleaşi noduri ca şi graful original dar mai puține muchii.

Graful este un instrument util în studiul unor colecții de entități între care există anumite legături ca de exemplu:

- Rețele de transport în care entitățile sunt localități, fabrici, depozite iar legăturile sunt drumuri auto, feroviare, aeriene, navale;

- Rețele de comunicație audio, video, calculator prin care se conectează localități, locuințe, instituții, birouri;
- Rețele electrice;
- Rețele genealogice etc.



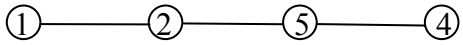
a) b)

Figura 4.2

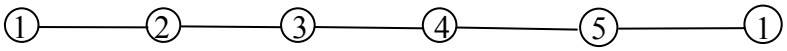
Un **lanț** în graful G este o succesiune de muchii $\lambda = (e_1, e_2, ..., e_p)$ cu proprietatea că fiecare muchie e_i are o extremitate în comun cu muchia precedentă e_{i-1} şi cealaltă extremitate în comun cu muchia următoare e_{i+1}. Natural, extremitățile lanțului λ sunt extremitățile „libere" ale primei şi ultimei muchii din lanț. Numărul *p* al muchiilor din succesiune se numeşte **lungimea** lanțului. Un **ciclu** este un lanț ale cărui extremități coincid.

O muchie este un lanț de lungime 1; două muchii adiacente constituie un lanț de lungime 2.

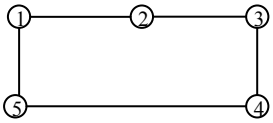
În graful din figura 4.1a) succesiunea de muchii:



este un lanț de lungime 3 iar succesiunea:



este un ciclu cu lungimea 5, reprezentat mai sugestiv astfel:



Graful G se zice **conex** dacă oricare două noduri distincte sunt extremitățile a cel puțin un lanț de muchii. Intuitiv, un graf conex este un graf „dintr-o bucată" aşa cum sunt cele din figurile 4.1 sau 4.2a). Graful din figura 4.2b) nu este conex fiind format din două „bucăți" numite **componente conexe**.

În continuare vom avea în vedere numai grafuri simple.

B) Clase importante de grafuri

1) Un graf se zice **complet** dacă oricare două noduri diferite sunt extremitățile unei singure

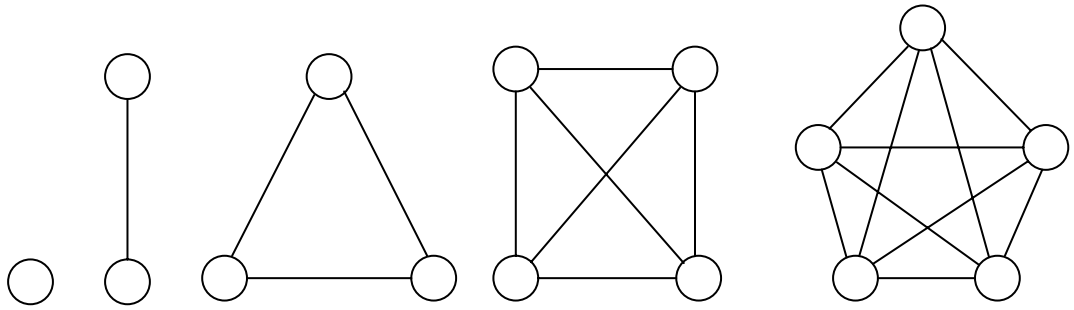
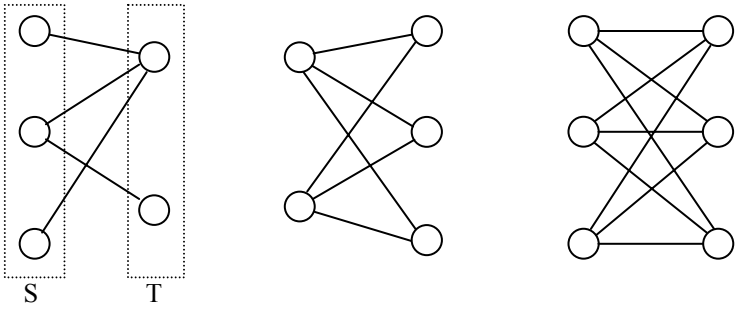


Figura 4.3

muchii. Un graf complet este perfect determinat de numărul *n* al nodurilor sale de unde şi notația $\mathbf{K_n}$. În figura 4.3 sunt vizualizate grafurile complete $\mathbf{K_1} - \mathbf{K_5}$.

2) Un **graf bipartit** este un graf simplu G = (V,E) a cărui mulțime de noduri V poate fi partiționată în două submulțimi disjuncte şi nevide S şi T astfel încât orice muchie din E are o extremitate în S şi cealaltă în T.Dacă în plus, pentru orice $x \in S$ şi $y \in T$ există muchia cu extremitățile *x* şi *y* graful se numeşte **bipartit complet** şi se notează cu sigla $\mathbf{K_{m,n}}$ fiind perfect determinat de numărul nodurilor $m = |S|$ şi $n = |T|$ din S respectiv T. În figura 4.4 sunt reprezentate câteva grafuri bipartite.



a)
graf bipartit care nu este complet

b)

$\mathbf{K_{2,3}}$

c)

$\mathbf{K_{3,3}}$

Figura 4.4

Calitatea unui graf de a fi bipartit nu depinde de reprezentarea sa grafică. În figura 4.5, graful din stânga nu pare a fi bipartit; o redesenare a sa ne convinge de contrariu.

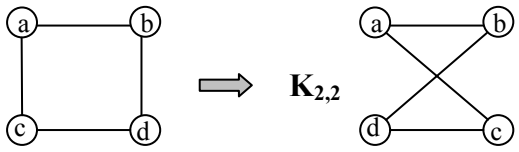


Figura 4.5

Iată o caracterizare elegantă a grafurilor bipartite:

Un graf este **bipartit** dacă şi numai dacă nu conține cicluri de lungime **impară**.

3) Un graf se numeşte **arbore** dacă este conex şi **nu are cicluri** – vezi figura 4.6

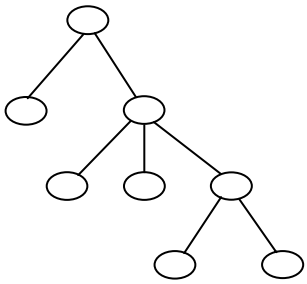


Figura 4.6

Caracterizări echivalente: Graful H este un arbore dacă şi numai dacă:

a) H este simplu şi oricare două noduri diferite sunt extremitățile unui **unic** lanț de muchii;
b) H este simplu şi numărul muchiilor este cu unu mai mic decât cel al nodurilor.

Fie G = (V,E) un graf oarecare. Se numeşte **arbore generator** al grafului G un subgraf H = (V,A) cu aceleaşi noduri ca şi G şi care de sine stătător este un arbore.De exemplu, graful din figura 4.2a) este un arbore generator al grafului din figura 4.1a).Este uşor de văzut că **un graf are arbori generatori dacă şi numai dacă este conex.**

4) Un graf se numeşte **planar** dacă există o reprezentare grafică a sa cu proprietatea că oricare două muchii nu se intersectează decât cel mult în extremități.

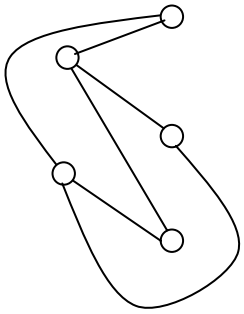


Figura 4.7

De exemplu, cele două grafuri din figura 4.1 sunt planare. La prima vedere, graful bipartit complet $\mathbf{K_{2,3}}$ vizualizat în figura 4.4b) pare a nu fi planar; o redesenare a sa ne convinge de contrariu – vezi figura 4.7. De fapt, calitatea unui graf de a fi planar nu depinde de reprezentarea grafică ci ține de structura sa intrinsecă; caracterizarea este însă mai dificilă. Se poate demonstra că graful complet $\mathbf{K_5}$ şi graful bipartit complet $\mathbf{K_{3,3}}$ nu sunt planare şi mai mult orice graf care conține $\mathbf{K_5}$ sau $\mathbf{K_{3,3}}$ ca subgrafuri nu este planar!

C. Orientarea unui graf

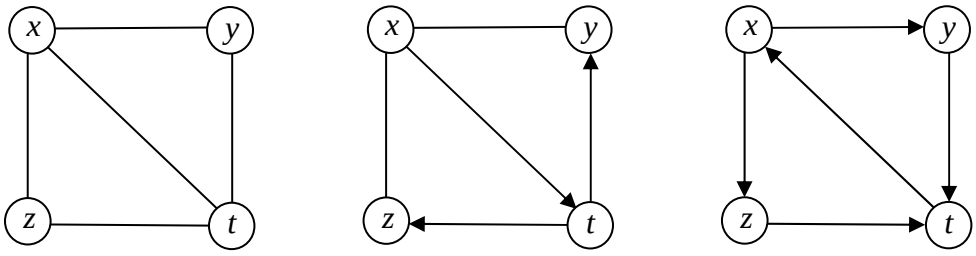
Fixăm un graf G = (V,E) pe care îl presupunem a fi **simplu**.Aceasta înseamnă că dacă o muchie din E are extremitățile *x* , *y* nu există nici o alta cu aceleaşi extremități şi ca urmare poate fi notată inambiguu prin sigla $\{x, y\}$.

Fie atunci $\{x, y\}$ o muchie oarecare din G. Perechile ordonate (*x, y*) şi (*y , x*) se vor numi **arce generate** de muchia $\{x, y\}$. Despre oricare dintre aceste arce vom spune că este **opus** celuilalt. Pentru arcul (*x, y*) nodul *x* va fi extremitatea **inițială** iar *y* extremitatea **finală**.

A da o **orientare pe muchia** $\{x, y\}$ înseamnă **a alege** unul din arcele (*x, y*) sau (*y, x*). Arcul ales se va numi arc **permis**, celălalt se va numi **blocat**.

Dacă muchia $\{x, y\}$ nu a fost orientată convenim ca ambele arce generate să fie considerate permise!

A da o **orientare în graful** G înseamnă a da o orientare pe unele muchii ale sale (posibil pe nici una, posibil pe toate) – vezi figura 4.8



Orientarea „**vidă**” a)

Orientare „**parțială**” b)

Orientare „**totală**” c)

Figura 4.8

Grafurile parțial sau total orientate servesc la reprezentarea:

- unor acțiuni complexe, compuse dintr-un mare număr de activități între care există unele relații de precedență în ceeace privește execuția;
- unei rețele stradale în care unele tronsoane pot fi parcurse în ambele sensuri, altele numai într-un sens;
- unor structuri organizaționale compuse din elemente – persoane fizice, departamente, birouri – între care există relații de subordonare.

Să presupunem că în graful fixat G = (V,E) s-a dat o **orientare** pe unele din muchiile sale. Fie U mulțimea arcelor permise; graful va fi renotat G = (V,E,U).

De exemplu, graful din figura 4.8b) are:

- patru noduri: x, y, z, t;
- cinci muchii: $\{x,y\}, \{x,z\}, \{x,t\}, \{y,t\}, \{z,t\}$;
- șapte arce permise: $(x,y), (y,x), (x,z), (z,x), (x,t), (t,y), (t,z)$

Un **drum** în graful G va fi o succesiune de arce permise $\mu = (u_1, u_2, ..., u_p)$ în care fiecare arc are ca extremitate finală extremitatea inițială a arcului următor. Drumul μ are o extremitate inițială – dată de extremitatea inițială a primului arc u_1 şi una finală ce coincide cu extremitatea finală a ultimului arc u_p. Lungimea drumului μ este dată de numărul p al arcelor alcătuitoare. Un **circuit** este un drum ale cărui extremități coincid.

De exemplu, în graful (parțial) orientat reprezentat în figura 4.8b) succesiunea $t \to z \to x \to y$ este un drum iar $t \to y \to x \to t$ este un circuit ambele având lungimea 3.

4.2 Ce este optimizarea combinatorială

Domeniul natural de manifestare al modelării cu variabile întregi îl constituie, de departe, **optimizarea combinatorială**. Ce este o problemă de optimizare combinatorială şi cum se modelează urmează să vedem în continuare.

Termenul **„combinatorică”** desemnează acea ramură a matematicilor care se ocupă cu studiul **„aranjamentelor”** unei mulțimi **finite**, aranjamente conforme unei **structuri** date.

Exemplul 1 Este posibil să grupăm v obiecte în b blocuri (mulțimi) astfel încât:

- fiecare bloc să conțină exact k obiecte diferite;
- fiecare obiect să se găsească în exact r blocuri;
- fiecare pereche de obiecte diferite să apară în exact λ blocuri ?

Este firesc să presupunem că existența unui asemenea aranjament implică anumite relații între parametrii *v, b, k, r,* λ şi aşa şi este. Iată un răspuns afirmativ în cazul particular

$$v = \mathrm{b} = 7 \quad , \quad k = r = 3 \quad , \quad \lambda = 1$$

în care obiectele au fost numerotate de la 1 la 7:

$$B_1 = \{1,2,4\}, B_2 = \{2,3,5\}, B_3 = \{3,4,6\}, B_4 = \{4,5,7\}, B_5 = \{5,6,1\}, B_6 = \{6,7,2\},$$
$$B_7 = \{7,1,3\}$$

În combinatorica clasică, prioritare erau chestiunile de **existență** şi de **numărare**:

- există un anumit tip particular de aranjament?
- Câte asemenea aranjamente se pot forma? Există o formulă analitică pentru numărul lor?

Exemplul 2 Câte permutări de ordinul *n*

$$\pi = \begin{pmatrix} 1 & 2 & ..... & n \\ \pi(1) & \pi(2) & .....\pi(n) \end{pmatrix}$$

au proprietatea $\pi(i) \neq i \quad i = 1,..,n$? (o asemenea permutare se numeşte **deranjament**).

Aici, obiectele sunt numerele 1, 2, ... , *n* iar aranjamentele sunt diferitele ordonări (listări) ale acestora. Numărul total de ordonări posibile este:

$$n! = 1 \cdot 2 \cdot \ldots \cdot n.$$

Se poate arăta că numărul D(*n*) al deranjamentelor de ordinul *n* este dat de formula:

$$D(n) = n!\left(1 - \frac{1}{1!} + \frac{1}{2!} - \ldots + (-1)^n \frac{1}{n!}\right)$$

În particular, din cele 4! = 24 permutări de ordinul 4, numai D(4) = 9 sunt deranjamente. Invităm cititorul să le construiască...

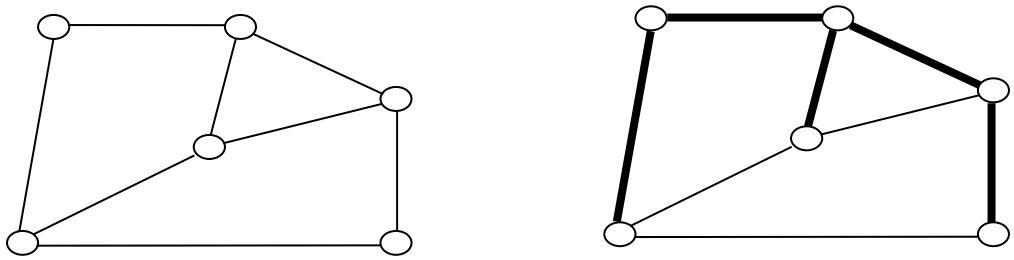
Dacă aranjamentele unei probleme combinatoriale pot fi **comparate** pe baza unui criteriu de apreciere, apare o **problemă de optimizare combinatorială**:

Care este cel mai bun aranjament din punctul de vedere al criteriului respectiv?

Aceste probleme sunt extrem de numeroase şi foarte multe provin din practica economică (programarea producției, ordonanțare, rețele de transport şi comunicații etc) dar şi din alte domenii (tehnic, militar, criptografie, sport, arheologie, chiar divertisment etc).

Exemplul 3. Problema arborelui de cost minim. Un număr de **puncte** (localități, locuințe, birouri, instituții etc.) trebuie conectate printr-o **rețea de comunicație** (telefonică, video, computer). Conectarea directă a două puncte oarecare – în cazul în care legătura este tehnic şi economic realizabilă – implică un anumit **cost**. Obiectivul este proiectarea unei rețele al cărei cost de instalare să fie **minim**.

Situația descrisă poate fi reprezentată pritr-un **graf simplu** G = (V,E) în care V este mulțimea



a) b)

Figura 4.9

nodurilor şi E este mulțimea **muchiilor** (vezi figura 4.9a)). Nodurile grafului corespund punctelor ce trebuie conectate în timp ce muchiile pun în evidență legăturile potențiale directe dintre puncte.

Rețeaua căutată corespunde unui **subgraf** H = (V,A) cu aceleaşi noduri ca şi G dar cu mai puține muchii. Subgraful H trebuie să îndeplinească următoarele cerințe:

- să permită conectarea oricăror două puncte, fie direct fie indirect, prin intermediul altor puncte;
- între două puncte oarecare să nu existe decât o singură modalitate de conectare (mai multe posibilități de legătură între două puncte înseamnă cheltuieli de instalare inutile).

În termenii teoriei grafurilor, H trebuie să fie – de sine stătător – un graf **conex** şi **fără cicluri** adică un **arbore** (vezi figura 4.9b – în care a fost pus în evidență un arbore prin îngroşarea unor muchii).

Asociem fiecărei muchii $e \in E$ o „pondere” $c(e) \geq 0$ reprezentând costul realizării legăturii directe dintre extremități. Costul instalării rețelei reprezentate de arborele H = (V,A) va rezulta din însumarea costurilor muchiilor alcătuitoare:

$$c(H) = \sum_{e \in A} c(e)$$

Am obținut astfel următoarea problemă de optimizare combinatorială:

Să se determine în graful G un arbore H^* de cost minim.

Observație: Cititorul atent a remarcat cu siguranță că, în situația „combinatorială" descrisă, „obiectele de lucru" sunt muchiile grafului G = (V,E). Un „aranjament" este aici o submulțime de muchii $A \subset E$ cu proprietatea că subgraful H = (V,A) este un arbore. Am menționat deja că graful G conține arbori dacă şi numai dacă este conex.Amintim doar în treacăt că există o formulă analitică care stabileşte numărul arborilor distincți din G (fireşte în cazul în care G este conex). De exemplu, dacă G este un graf complet cu *n* noduri, numărul arborilor săi este n^{n-2}.

Muchiile unui graf pot fi „aranjate" şi în alte moduri.

Exemplul 4. Problema cuplajului maxim Reamintim că un **cuplaj** în graful G = (V,E) este o submulțime $C \subseteq E$ de muchii, două câte două **neadiacente** (adică fără extremități comune).

Existența acestui nou tip de „aranjament" este o chestiune banală întrucât orice muchie constituie un cuplaj. Interesante sunt problemele de optimizare care se pun:

- să se determine cuplajul C^* cu cele mai multe muchii (**cuplajul maxim**);

sau, în cazul în care muchiile grafului G sunt „ponderate":

- să se determine cuplajul C^* de **pondere maximă**.

(ca şi în problema precedentă, ponderea unui cuplaj este dată de suma ponderilor muchiilor din cuplaj)

Exemplul 5. Problema drumului de valoare minimă Una dintre problemele importante ale activității de transport este aceea a determinării „celei mai bune" modalități de deplasare dintr-un loc în altul. Funcție de context, criteriul de apreciere poate să implice distanța totală de parcurs, timpul total al deplasării sau costul deplasării.

Chestiunea devine cu adevărat consistentă atunci când punctul de plecare şi destinația fac parte dintr-o rețea „densă" de puncte între care există „legături", numite şi rute. Am putea lua ca exemplu rețeaua stradală a unui mare oraş sau rețeaua rutieră a unei țări (cum ar fi cea a Germaniei, cea mai densă din Europa!). În asemenea cazuri, „simpla inspecție" – adică enumerarea tuturor posibilităților de deplasare – este total ineficientă: suntem pur şi simplu „copleşiți" de numărul acestor posibilități!

Pentru modelarea situației descrise vom utiliza un **graf simplu şi (parțial) orientat** G=(V,E): nodurile din V corespund centrelor importante ale rețelei reale în timp ce muchiile din E pun în evidență rutele de legătură. Orientarea unora dintre muchii sugerează posibilitatea ca rutele corespunzătoare să nu poată fi parcurse decât într-un singur sens, specificat. Fie U mulțimea **arcelor permise**; reamintim convenția potrivit căreia dacă o muchie $\{i, j\}$ nu este orientată, ambele arce generate (i, j) şi (j, i) să fie considerate permise!

Presupunem că fiecărui arc permis $u = (i, j) \in U$ i s-a asociat o valoare numerică notată ,după caz ,cu $c(u)$ sau c_{ij}. Funcție de context, aceste valori pot reprezenta distanțe, timpi de parcurgere sau costuri. Drumurile din graful G modelează diferitele posibilități de deplasare dintr-un punct al rețelei reale în altul.Valoarea unui **drum** va fi dată de suma valorilor arcelor alcătuitoare.

Fixând două noduri *s* şi *t* în G cu semnificația de punct de plecare respectiv destinație, se poate formula următoarea problemă de optimizare:

P(*s,t*): Să se determine în graful G drumul λ^* de valoare minimă de la *s* la *t*.

Mai generală este problema:

P(*s*): Să se determine în graful G drumurile de valoare minimă de la *s* la toate nodurile accesibile din *s*.

(vom spune că nodul *j* este **accesibil** din nodul *i* dacă există un drum – format numai din arce permise – de la *i* la *j*)

Este clar că rezolvând problema P(*s*) se rezolvă şi problema P(*s*,*t*) cu condiția ca nodul *t* să fie accesibil din *s*.

Caracterul combinatorial al problemelor formulate este evident: „obiectele" de lucru sunt arcele permise ale grafului iar „aranjamentele" sunt diferitele drumuri alcătuite numai din arce permise! Deoarece graful este finit şi numărul acestor aranjamente este finit.

În continuare, pentru aranjamentele unei probleme de optimizare combinatorială vom prefera denumirea consacrată de **„soluții (admisibile)"**. Evident, numărul acestora este întotdeauna **finit**. La prima vedere, s-ar părea că găsirea „soluției optime" este o chestiune banală:

„Ce mare lucru este să găseşti cel mai bun element dintr-o mulțime finită? N-ai decât să listezi toate elementele mulțimii şi să le examinezi pe rând, reținând tot timpul pe cel mai bun. În final vei găsi elementul dorit…"

În realitate, generarea tuturor soluțiilor unei probleme de optimizare combinatorială în vederea determinării soluției optime – operație cunoscută sub numele de **enumerare completă** – este practic imposibil de făcut în majoritatea cazurilor, deoarece numărul acestor soluții, deşi finit, este "astronomic" de mare chiar şi pentru probleme de talie moderată. Următorul exemplu este edificator.

Exemplul 6. Problema comisvoiajorului. Un comisvoiajor situat în localitatea 0 trebuie să viziteze localitățile 1, 2, ... *n* în final întorcându-se în locul de unde a plecat. Cunoscând distanțele dintre localități (sau costurile deplasărilor sau timpul de deplasare) el doreşte să urmeze traseul cu lungimea totală (sau costul total sau timpul total) **minim.**

Aceasta este probabil cea mai cunoscută problemă de optimizare combinatorială fiind considerată ca exemplu tipic de problemă „uşor de explicat dar foarte greu de rezolvat".

Enunțul de mai sus constituie forma uzuală de prezentare a problemei comisvoiajorului. Există o mare varietate de situații practice a căror descriere este asemănătoare:

- un trasportator are o listă de clienți pe la care trebuie să treacă în decursul unei zile; în ce ordine îi va vizita pentru ca treaba să se termine cât mai repede?

- Pe o placă cu circuite integrate trebuie date găuri într-un număr de poziții prestabilite. Numărul acestor poziții este foarte mare (de ordinal miilor) astfel că timpul necesar executării întregii operații are două componente, ambele „apreciabile": una constantă, egală cu suma timpilor de execuție propriu zisă a găurilor şi alta **variabilă**, formată din timpii de deplasare ai burghiului de la o poziție la alta. Din motive evidente de productivitate apare chestiunea: în ce ordine vor fi date găurile astfel ca întreaga operație să dureze cât mai puțin?

Revenind la enunțul standard, numărul total al traseelor posibile este $n!$, un traseu fiind perfect determinat de ordinea (permutarea) în care vor fi vizitate localitățile 1 , 2 , ... , n. Odată fixată această ordine, calculul lungimii (costului) traseului este o banală operație de adunare a lungimilor (costurilor) tronsoanelor care compun traseul.

În cazul particular a $n = 20$ de localități de vizitat, cu un calculator în stare să examineze un miliard de trasee pe secundă, găsirea traseului optim ar dura circa 80 de ani iar dacă în loc de 20 de localități am avea cu una in plus, căutarea ar dura în jur de 1680 de ani!!

În lumina acestui exemplu, **rezolvarea** unei probleme de optimizare combinatorială, adică determinarea **efectivă** a soluției optime din mulțimea **finită** a tuturor soluțiilor posibile, devine o chestiune **serioasă**: am dori să găsim soluția optimă in **timp util** şi, bineînțeles, cu un efort de calcul **rezonabil.** În termeni ceva mai preciși, am dori ca timpul de rezolvare al problemei să depindă **polinomial** de „dimensiunile" problemei.

Pentru unele probleme de optimizare combinatorială acest lucru este posibil; despre aceste probleme vom spune că sunt **uşoare**. Problema arborelui de cost minim descrisă în exemplul 3, problema cuplajului maxim din exemplul 4 sau problema drumului de valoare minimă din exemplul 5 sunt dovedite a fi probleme uşoare!

Pentru marea majoritate a problemelor combinatoriale, găsirea soluției optime **în timp polinomial** nu este încă posibilă, altfel spus metodele exacte cunoscute necesită un timp de rulare care creşte **exponențial** o dată cu dimensiunile problemei de rezolvat. Mai mult, există bănuiala că datorită structurii lor interne, nici nu va fi posibilă vreodată elaborarea de metode exacte de rezolvare cu comportament „polinomial".

Din clasa acestor probleme, socotite **grele**, face parte problema generală a programării în numere întregi anterior studiată dar şi problema comisvoiajorului din exemplul 5 ca şi altele care vor fi descrise şi studiate mai departe.

Din cele de mai sus nu trebuie înțeles că problemele „grele" sunt intractabile! În dimensiune „relativ moderată" aceste probleme pot fi soluționate exact, în timp acceptabil. Folosind metode de căutare din ce în ce mai sofisticate – care exploatează eficient structura combinatorială internă – au putut fi rezolvate cu succes şi probleme de dimensiuni apreciabile. Totuşi, nu se poate spune că aceste

metode, oricât de sofisticate ar fi ele, sunt în stare „să facă față” oricărei alte probleme similare şi mai mult, pot avea comportamente radical diferite pe probleme asemănătoare şi de aceeaşi talie!

Având în vedere aceste observații, de mare importanță, mai cu seamă practică, sunt metodele şi algoritmii care determină în timp polinomial o soluție **„acceptabilă”, suboptimală**. Pentru aceste procedee denumite generic **euristici**, aprecierea eficacității lor va fi dată de „distanța” dintre soluția suboptimală şi soluția optimă veritabilă. Am putea aprecia că o anumită euristică este „bună” dacă valoarea suboptimală a criteriului de performanță ar fi, de exemplu, de cel puțin 90% din valoarea optimă!

Să recunoaştem că până acum, domeniul optimizării combinatoriale a fost introdus prin câteva exemple şi printr-o descriere „în cuvinte”, destul de aproximativă, cu totul insuficientă pentru înțelegerea tehnicilor de abordare ale subiectului. Se impune adoptarea unui limbaj matematic concis şi adecvat.

Formal, o problemă de optimizare combinatorială P înseamnă o clasă de **concretizări** (**situații**) plus un **scop**. O concretizare a problemei P este un cuplu $(\mathcal{A},c)$ în care $\mathcal{A}$ este o mulțime **finită** şi c: $\mathcal{A} \to R$ este o **funcție** oarecare. Pentru fiecare situație $(\mathcal{A},c)$ scopul problemei P este găsirea unui element $x^* \in \mathcal{A}$ care maximizează sau, după caz, minimizează valoarea funcției c.

De fapt, aceasta este descrierea formală a oricărei probleme de optimizare şi în consecință este firesc să menținem terminologia generală recunoscută. Astfel, elementele mulțimii $\mathcal{A}$ se numesc **soluții admisibile**, c este **funcția obiectiv** iar x^* este **soluția optimă** a problemei P relativ la concretizarea $(\mathcal{A},c)$.

Singurul amănunt distinctiv este deocamdată faptul că, indiferent de concretizare, soluțiile admisibile sunt în număr **finit** şi ca atare, determinarea elementului optimal poate fi făcută prin **enumerarea** tuturor acestor soluții. Din acest punct de vedere, pentru ca problem P să fie nebanală, este necesar ca orice concretizare $(\mathcal{A},c)$ să fie **„structurată”** într-un fel oarecare, adică $\mathcal{A}$ şi c să fie descrise printr-un număr de „relații şi proprietăți” mult mai mic decât, să zicem, numărul elementelor din $\mathcal{A}$!!

Pentru foarte multe probleme de optimizare combinatorială, printre care se numără şi problemele din exemplele precedente o concretizare $(\mathcal{A},c)$ se defineşte după următoarea „schemă”.

Este dată o mulțime **finită** E ale cărei elemente au fost **„ponderate”** printr-o funcție oarecare c:E $\to R$. Funcția c se extinde **„liniar”** la orice submulțime A $\subseteq$ E prin relația:

$$c(A) = \sum_{e \in A} c(e)$$

Soluțiile admisibile vor fi acele submulțimi din E care satisfac o anumită proprietate $\mathcal{J}$:

$$\mathcal{A} = \{A \subseteq E \mid A \text{ satisface proprietatea } \mathcal{J}\}$$

De multe ori, mulțimea de bază E derivă dintr-un **graf finit** putând fi constituită fie din nodurile grafului fie din muchiile sau din arcele acestuia dacă graful este (parțial)orientat. Bineînțeles că în acest caz proprietatea $\mathcal{J}$ este exprimată în termeni specifici teoriei grafurilor.

Exemplul 6. O concretizare a problemei comisvoiajorului din exemplul 5 este dată de mulțimea E a arcelor unui **graf complet K_{n+1}** cu n+1 noduri notate 0,1,2,.., n .Graful are $\binom{n+1}{2} = \frac{n(n+1)}{2}$ muchii şi fiecare muchie $\{i,j\}$ produce două arce opuse (i,j) şi (j,i). Fiecare arc (i,j) este „ponderat" cu o valoare nenegativă c_{ij} ce reprezintă fie distanța dintre localiățile i şi j fie costul sau timpul de deplasare al comisvoiajorului de la i la j. Nu este exclus cazul $c_{ij} \neq c_{ji}$, cu alte cuvinte ponderarea unei muchii poate să depindă de sensul de parcurgere al acesteia!

Soluțiile admisibile ale problemei sunt acele submulțimi de arce $A \subset E$ cu proprietatea că formează un **traseu** adică un **circuit care trece – o singură dată – prin toate nodurile grafului**. Prin urmare, în acest caz, proprietatea $\mathcal{J}$ are expresia:

$\mathcal{J}$: $A \subset E$ este un **traseu**

Lungimea (sau costul) unui traseu A se obține prin însumarea ponderilor arcelor din A.

Astfel, problema comisvoiajorului se încadrează în schema formală prezentată mai sus. Acelaşi lucru se poate spune şi despre problema arborelui de cost minim: în notațiile exemplului 3, soluțiile admisibile ale problemei vor fi acele submulțimi de muchii $A \subset E$ cu proprietatea:

$\mathcal{J}$: subgraful H= (V,A) este un **arbore**

În continuare fixăm o problemă de optimizare combinatorială P şi o concretizare a acesteia $(\mathcal{A}, c)$ dată prin tripletul $(E, c, \mathcal{J})$. În unele situații, simplul fapt că E şi $\mathcal{J}$ sunt descrise cu ajutorul unui graf este suficient pentru rezolvarea problemei P. Este cazul problemei arborelui de cost minim, a problemei cuplajului maxim sau a problemei drumului de valoare minimă.

În alte situații, modelarea cu variabile întregi (bivalente) este mai profitabilă. Schema generală de modelare este următoarea:

- Fiecărui element $e \in E$ i se asociază o variabilă bivalentă x_e. Atunci, orice submulțime $A \subset E$ este perfect determinată de **vectorul** ei **de incidență** $x = (x_e)$ în care:

$$x_e = \begin{cases} 1 & \text{daca } e \in A \\ 0 & \text{daca } e \notin A \end{cases}$$

- Restricțiile modelului, adică ecuațiile şi/sau inecuațiile în variabilele x_e , $e \in$ E vor „traduce" proprietatea $\mathcal{J}$ care defineşte soluțiile admisibile A $\subset$ E ale problemei.

Traducerea în limbaj ecuațional a proprietății $\mathcal{J}$ este în general o operație dificilă şi de multe ori poate fi făcută în mai multe moduri (vezi exemplul 8)

- Funcția obiectiv a modelului, notată prin abuz tot cu c are expresia:

$$c(x) = \sum_{e \in E} c_e x_e$$

unde c_e este „ponderea" asociată elementului $e \in$ E.

În principiu, orice problemă de optimizare combinatorială poate fi modelată mai uşor sau mai greu ca problemă de programare în numere întregi.

Exemplul 7. Problema afectării. Un număr de **lucrări** (proiecte, operații, sarcini de serviciu) $L_1, L_2, .. ,L_n$ sunt propuse spre executare **agenților** (persoane fizice, agenții specializate) $A_1, A_2, .. ,A_m$. Fiecare agent ştie să execute una sau mai multe dintre lucrările propuse. În **problema afectării simple** se cere determinarea numărului **maxim** de lucrări ce pot fi atribuite agenților abilitați astfel încât:

- O lucrare să nu fie atribuită decât cel mult unui agent;
- Un agent să nu poată primi mai mult de o lucrare.

Se observă că dacă mai mulți agenți pot executa o aceeaşi lucrare, nu se face nici o referire asupra modului în care lucrarea este terminată: mai bine sau mai puțin bine, mai repede sau mai încet, la un cost mai mare sau mai mic etc.În situația în care se poate cuantifica măsura c_{ij} a abilității agentului A_i în executarea lucrării L_j , un criteriu de performanță care se impune este acela al identificării lucrărilor şi a agenților executanți astfel încât suma eficacităților de execuție să fie **maximă**. Aceasta este **problema generală a afectării**.

Evident, afectarea simplă este un caz particular al afectării generale dacă punem:

$$c_{ij} = \begin{cases} 1 & \text{daca agentul } A_i \text{ poate executa lucrarea } L_j \\ 0 & \text{in caz contrar} \end{cases}$$

Problema afectării este o problemă de cuplaj maxim (de pondere maximă) într-un **graf bipartit** în care:

- nodurile corespund agenților $A_1, A_2, .. ,A_m$ şi lucrărilor $L_1, L_2, .. ,L_n$;

- muchiile au forma $\{A_i , L_j\}$ unde A_i este un agent calificat să execute lucrarea L_j cu calificativul $c_{ij} > 0$ (se presupune tacit că „notele" c_{ij} satisfac cerințele naturale:

$c_{ij} = 0$ dacă agentul A_i nu este în măsură să execute lucrarea L_j;

$c_{i_1 j} > c_{i_2 j}$ dacă pe aceeaşi lucrare L_j agentul A_{i_1} este „mai bun" decât A_{i_2}).

Problema se încadrează în schema de formalizare descrisă mai înainte. Într-adevăr, mulțimea de bază E este mulțimea muchiilor grafului G iar soluțiile admisibile sunt acele submulțimi $C \subset E$ cu proprietatea:

$\mathcal{J}$: C este un **cuplaj** în G. (1)

Pentru a obtine modelul echivalent în numere întregi, conform aceleaşi scheme vom asocia fiecărei muchii $\{A_i , L_j\}$ din E o variabilă bivalentă $x_{ij} \quad i = 1,..,m \; ; \; j = 1,..,n$

O submulțime oarecare de muchii $C \subseteq E$ va fi determinată de vectorul său de incidență $x = (x_{ij})$ în care:

$$x_{ij} = \begin{cases} 1 \text{ daca muchia } \{A_i , L_j\} \text{ este in C} \\ 0 \text{ in caz contrar} \end{cases}$$

Este clar atunci că sumele $\sum_{j=1}^{n} x_{ij}$ respectiv $\sum_{i=1}^{m} x_{ij}$ reprezintă numărul muchiilor din C care au o extremitate în nodul A_i, respectiv în nodul L_j, şi asta pentru orice i = 1,..,*m* respectiv j = 1,..,*n*. În consecință, C va fi un cuplaj dacă şi numai dacă:

$$\sum_{j=1}^{n} x_{ij} \le 1 \quad i = 1,..,m \qquad (2)$$

$$\sum_{i=1}^{m} x_{ij} \le 1 \quad j = 1,..,n \qquad (3)$$

Cele *m* + *n* relații din (2) şi (3) constituie traducerea în limbaj de inecuații a proprietății $\mathcal{J}$ din (1).

Dacă C este un cuplaj (reprezentând o modalitate de repartizare a unor agenți pe anumite lucrări), relațiile (2) , (3) arată că în expresia:

$$z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \qquad (4)$$

vor apare numai calificările c_{ij} corespunzătoare atribuirilor din C. În concluzie, (4) reprezintă funcția obiectiv a modelului, urmând a fi maximizată.

Reunind (2) , (3) , (4) obținem programul bivalent:

$$\begin{cases} (\max) z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \sum_{j=1}^{n} x_{ij} \le 1 \qquad i = 1,..,m \\ \sum_{i=1}^{m} x_{ij} \le 1 \qquad j = 1,..,n \\ x_{ij} \in \{0,1\} \end{cases} \qquad (5)$$

care reprezintă o altă posibilitate de modelare a problemei afectării.

Observații: 1) Urmând aceeaşi cale, pentru problema cuplajului de pondere maximă într-un graf oarecare G = (V, E) din exemplul 4 se obține modelul:

$$\begin{cases} (\max) z = \sum_{E \in E} c_e x_e \\ \sum_{e \in E(v)} x_e \le 1 \,, v \in V \\ x_e \in \{0,1\} \end{cases} \qquad (6)$$

unde x_e este o variabilă bivalentă asociata muchiei $e \in E$ iar *E(v)* reprezintă mulțimea muchiilor grafului care au o extremitate în nodul $v \in V$ .Evident, (5) este un caz particular al modelului (6).

2) În cazul $m = n$ se obişnuieşte să se ia în considerare toate atribuirile posibile $\{A_i , L_j\}$ deci şi cele pentru care $c_{ij} = 0$. **Cuplajele maximale** vor avea *n* muchii şi numărul lor va fi *n*! Evident soluția optimă a problemei (5) trebuie căutată în submulțimea acestor cuplaje. Pentru a caracteriza un cuplaj maximal relațiile (2) , (3) trebuie să devină egalități. Astfel, în cazul $m = n$, modelul (5) devine:

$$\begin{cases} (\max) z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,..,n \\ \sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,..,n \\ x_{ij} \in \{0,1\} \end{cases} \tag{7}$$

Cazul general $m \neq n$ se reduce imediat la acesta adăugând, după necesități, fie un număr de „agenți" fictivi, fie un număr de „lucrări" fictive.Orice notă c_{ij} care implică un agent fictiv A_i sau o lucrare fictivă L_j se va lua egală cu 0. În final din soluția optimă se exclud atribuirile $\{A_i , L_j\}$ pentru care c_{ij}=0.

3) În alte contexte, "nota" (aprecierea) c_{ij} are semnificația unui **timp** necesar agentului A_j pentru a finalize lucrarea L_i. În acest caz, obiectivul optimizării va fi repartizarea lucrărilor pe agenți astfel încât suma timpilor de execuție a lucrărilor să fie **minimă**. Modelul (7) se menține, schimbându-se numai sensul optimizării "din max în min".

Putem spune că (7) reprezintă **modelul standard** al problemei generale de afectare.

3) Remarcăm faptul că, modelarea problemei afectării ca problemă de programare bivalentă, a condus la un model cu un număr „rezonabil" de restricții. Aceasta este o situație fericită şi destul de rar întâlnită. De regulă, modelarea cu variabile întregi a problemelor de optimizare combinatorială conduce la programe cu un număr „înfiorător" de mare de restricții (în alți termeni, traducerea proprietății $\mathcal{J}$ din limbajul foarte concis al teoriei grafurilor de exemplu, în limbaj ecuațional,poate fi „neplăcut de stufoasă".Următorul exemplu este edificator.

Exemplul 8. Modelarea cu variabile întregi (bivalente) a problemei comisvoiajorului

Am văzut în exemplul 6 cum se formalizează problema comisvoiajorului în limbajul teoriei grafurilor.Din nefericire, această formalizare nu este suficientă – cel puțin până acum – pentru rezolvarea „eficientă" a problemei (prin rezolvare eficientă înțelegem o procedură de găsire a traseului oprtimal care să excludă enumerarea completă...)

Un traseu de vizitare a localităților 1,2,…,n cu plecarea şi întoarcerea în localitatea 0 poate fi descris cu ajutorul variabilelor bivalente:

$$x_{ij} = \begin{cases} 1 \text{ daca comisvoiajorul merge direct din orasul } i \text{ in orasul } j \text{ si } i \neq j \\ 0 \text{ in caz contrar sau daca } i = j \end{cases}$$

unde $i, j = 0,1,...,n$

(este clar că variabilele x_{ij} sunt asociate arcelor $(i, j) \in E$ ale grafului complet cu n+1 noduri din exemplul 6.)

Valoarea traseului (adică distanța totală parcursă sau costul total sau timpul total de deplasare) va fi dată de funcția:

$$f = \sum_{i,j=0}^{n} c_{ij} x_{ij} \qquad (8)$$

Evident, un set de valori 0 sau 1 acordate componentelor vectorului $x = (x_{ij})$ va caracteriza o anumită submulțime de arce $A \subset E$. Să vedem acum cum se traduce proprietatea:

$\mathcal{J}$: $A \subset E$ este un **traseu**

- este clar că dintr-un oraş *i* comisvoiajorul trebuie să plece către un (unic) alt oraş şi deci:

$$\sum_{j=0}^{n} x_{ij} = 1 \quad i = 0,1,..,n \qquad (9)$$

- o dată ajuns într-un oraş *j* comisvoiajorul vine dintr-un (unic) alt oraş şi prin urmare:

$$\sum_{i=0}^{n} x_{ij} = 1 \quad j = 0,1,..,n \qquad (10)$$

Observăm că relațiile (9) şi (10) constituie restricțiile problemei de afectare din modelul standard (7) în care funcția obiectiv (8) va fi minimizată!

O examinare mai atentă a problemei arată că aceste relații nu definesc numai trasee ce trec prin toate oraşele (o singură dată!) ci şi "reuniuni" de circuite disjuncte mai mici! Astfel, într-o problemă cu 7 oraşe 0,1,...,6 ansamblul:

$$x_{03} = x_{35} = x_{50} = x_{21} = x_{14} = x_{46} = x_{67} = x_{72} = 1 \quad , \quad x_{ij} = 0 \quad \text{în rest}$$

satisface restricțiile (9) şi (10) dar nu constituie un traseu complet aşa cum se vede din figura 4.10.

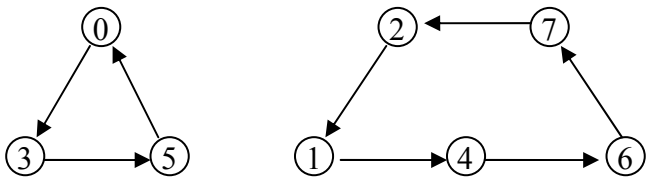


Figura 4.10

Pentru eliminarea apariției subtraseelor este necesar să completăm sistemul (9) –(10) cu noi restricții. Există mai multe moduri de a face acest lucru.

i) Se introduc inegalitățile:

$$\sum_{i,j\in U} x_{ij} \le |U| - 1 \qquad (11)$$

pentru orice submulțime $U \subset \{0,1,...,n\}$ cu $2 \le |U| \le \frac{n}{2}$ (unde s-a notat cu $|U|$ **numărul** elementelor mulțimii finite U). Într-adevăr, orice subtraseu care trece prin nodurile unei submulțimi U are $|U|$ arce, astfel că inegalitatea corespunzătoare (11) le va exclude!

ii) Se introduc inegalitățile:

$$\sum_{i\in U, j\notin U} x_{ij} \ge 1 \qquad (11')$$

pentru orice submulțime $U \subset \{0,1,...,n\}$ cu $2 \le |U| \le \frac{n}{2}$ .Justificarea este simplă: deoarece un traseu trece prin toate localitățile, oricum am lua o submulțime strictă U, traseul va trebui "să iasă din U" aceasta însemnând existența unui oraş *i* în U şi a altuia *j* din afara lui U în care comisvoiajorul va ajunge după ce a trecut prin *i*. Formal: există $i \in U$ şi $j \notin U$ astfel încât $x_{ij} = 1$ în orice traseu!

iii) Evitarea subtraseelor se poate face şi cu ajutorul următorului set de restricții care are avantajul de a fi "mai compact":

$$y_i - y_j + n \cdot x_{ij} \le n - 1 \quad i, j = 1,2,...,n \; ; \; i \ne j \qquad (11'')$$

unde $y_1, y_2, ..., y_n$ sunt noi variabile ce pot lua, în principiu, orice valoare reală.

Fie acum $\bar{x} = (\bar{x}_{ij})$ o soluție cu componente 0,1 a problemei de afectare (8) – (10). Vom arăta că dacă $\bar{x}$ reprezintă un traseu complet există (cel puțin) un set de valori $\bar{y}$ pentru variabilele $y = (y_1,...,y_n)$ care, împreună cu $\bar{x}$ satisfac (11''). Fie $(i_1, i_2, ..., i_n)$ ordinea în care localitățile 1,2,… *n* sunt vizitate (evident $\{i_1, i_2, ..., i_n\} \equiv \{1,2,...,n\}$!). Vom pune $\bar{y}_{i_k} = k$; altfel spus, $\bar{y}_i$ este **locul** pe care îl ocupă oraşul *i* în secvența $(i_1, i_2, ..., i_n)$. Pentru orice pereche *i,j* cu $i \ne j$ avem două posibilități:

- $\bar{x}_{ij} = 0$, ceeace înseamnă că traseul definit de $\bar{x}$ nu trece direct de la *i* la *j*

Inegalitatea (11'') corespunzătoare devine $\bar{y}_i - \bar{y}_j \le n - 1$ şi este banal îndeplinită dacă avem în vedere definiția valorilor $\bar{y}$!

- $\bar{x}_{ij} = 1$, altfel spus traseul definit de $\bar{x}$ trece de la *i* direct la *j* . Este clar atunci că $\bar{y}_j = \bar{y}_i + 1$ şi (11'') este verificată cu egalitate.

Să presupunem acum că $\bar{x} = (\bar{x}_{ij})$ defineşte o reuniune de "subtrasee" disjuncte. Alegem unul care nu trece prin localitatea 0 şi fie k numărul deplasărilor ce îl compun. Presupunând că ar exista valorile numerice $\bar{y} = (\bar{y}_1,...,\bar{y}_n)$ care să satisfacă (11'') le însumăm pe acelea corespunzătoare deplasărilor de la un oraş la altul din subtraseul ales. Obținem $n \cdot k \leq (n - 1) \cdot k$,ceeace este evident fals.

Recapitulând, am arătat că problema comisvoiajorului poate fi modelată ca un program bivalent şi mai précis ca o problemă de afectare – vezi relațiile (8) – (10) - cu restricțiile **speciale** (11) sau (11') sau (11''). Notă: adăugarea setului (11'') conduce de fapt la un program mixt…

Observație: Numărul relațiilor (11) sau (11') este foarte mare, comparabil cu 2^n , ceeace la prima vedere duce la concluzia că problema de afectare (8) – (10) extinsă cu aceste relații este "mai greu de mânuit" decât extinderea cu grupul de relații (11'') care are numai $n^2 - n$ inegalități.Paradoxal, numărul "imens" de restricții (11) sau (11') este relativ simplu de controlat în toate procedurile exacte de rezolvare aproblemei comisvoiajorului.

4.3 Alte probleme reprezentative ale optimizării combinatoriale

Lista problemelor de optimizare combinatorială este impresionantă şi se îmbogățeşte continuu. În afara celor prezentate în secțiunea precedentă mai putem aminti următoarele:

Problema croirii În multe domenii cum ar fi industria mobilei, a sticlei, a hârtiei, a confecțiilor, industria metalurgică apar frecvent probleme de debitare (sau croire) a unor repere din suporți mai mari. Importanța economică a acestor probleme rezidă în dependența directă a reducerii consumului de materiale de utilizarea unor scheme eficiente de croire.

De obicei, clasificarea problemelor de croire se face după dimensiunile relevante ale reperelor: avem probleme unidimensionale, bidimensionale sau tridimensionale. În croirea unidimensională, reperele se diferențiază printr-o singură dimensiune chiar dacă ele au mai multe. Cazul tipic este ecela al debitării unor bare de diferite lungimi din bare mai mari.Iată şi o altă situație concretă: o firmă produce un carton special în rulouri având o anumită lățime. Clienții solicită rulouri având aceeaşi lungime ca şi ruloul standard dar de lățimi mai mici. În acest caz reperele ca şi suporții sunt bidimensionali dar relevantă este o singură dimensiune – lățimea.

Aspectul combinatorial al unei probleme de croire este dat de modalitățile de tăiere ale unui suport în repere, modalități numite **rețete de croire**.Chestiunea de optimizare constă în a determina de câte ori una sau alta dintre rețetele de croire trebuie folosită pentru producerea unor cantități specificate de repere astfel încât numărul total de suporți consumați să fie minim.

Problema generală a ordonanțării. Elementele primare sunt:

- o mulțime de utilaje;
- o mulțime de repere (joburi) de realizat (îndeplinit).

Realizarea unui reper necesită efectuarea unei secvențe de operații pe (unele din) utilajele date.Ordinea de parcurgere a utilajelor poate fi aceeaşi pentru toate reperele sau poate diferi de la un reper la altul.Fiecare operație necesită un anume timp, presupus cunoscut, şi două operații distincte, executabile pe un acelaşi utilaj, nu pot fi programate simultan.Uneori pentru unele repere sunt impuse termene de livrare.

În aceste ipoteze,cum trebuie organizată lansarea reperelor în execuție astfel încât:

- timpul total de inactivitate al utilajelor să fie minim

sau

- durata de execuție a tuturor reperelor să fie minimă

sau

- numărul reperelor al căror termen de livrare este depăşit să fie cât mai mic etc.?

Există o mare varietate de tipuri de probleme de ordonanțare care se înscriu în acest enunț general. Modelarea lor - prin programare în numere întregi sau prin grafuri - este în general dificilă, ca să nu mai vorbim de rezolvarea lor "exactă", aproape imposibil de realizat în timp rezonabil, chiar şi atunci când numărul reperelor şi al utilajelor este relativ mic. Ca urmare, au fost elaborate euristici de rezolvare suboptimală, marea lor diversitate fiind explicată prin aceea că ele încearcă să exploateze particularitățile structurii acestor probleme, particularități care pot să difere radical de la o problemă la alta.

Vom studia unele cazuri particulare într-o altă unitate de învățare.

Problema alegerii traseelor. O firmă trebuie să satisfacă un număr de cereri de aprovizionare într-o anumită perioadă. Livrările se fac de la un anumit depozit şi sunt operate cu mai multe de mijloace de transport, fiecare cu o capacitate limitată. Fiecare vehicul pleacă încărcat de la depozit, vizitează unul sau mai mulți clienți, cărora le livrează comenzile făcute după care, complet descărcat, se întoarce la depozit pentru a fi eventual reîncărcat. Problema care se pune în acest context este aceea de a stabili traseele de vizitare ale clienților astfel încât distanța totală parcursă de vehicule să fie minimă.

O posibilitate de rezolvare suboptimală este descrisă într-o viitoare unitate de învățare.

Problema orariilor Într-o universitate trebuie programată activitatea de învățământ pe următorul semestru. Se cunosc:

- numărul cursurilor care trebuie ținute;
- numărul claselor disponibile;
- profesorii abilitați să țină cursurile.

Cum trebuie făcută repartizarea profesorilor pe cursuri şi a cursurilor pe clase astfel încât:

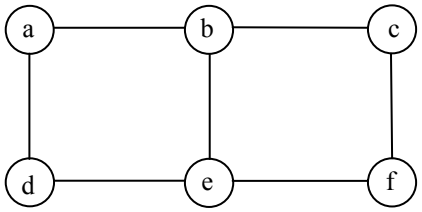
- două cursuri diferite să nu fie programate în aceeaşi clasă şi în acelaşi timp;
- cursurile să fie atribuite profesorilor de specialitate;
- doi profesori cu aceeaşi specialitate să nu fie repartizați pe acelaşi curs.

O asemenea repartizare complexă se numeşte **orar** şi până aici s-a formulat doar o problemă de **existență.** “Optimizarea” elaborării unui orar este o chestiune delicată existând multe criterii de apreciere care nu întotdeauna sunt concordante. Din punctul de vedere al studenților un orar este "bun" dacă minimizează totalul deplasărilor zilnice de la o clasă la alta. Atât pentru studenți cât şi pentru profesori contează numărul "ferestrelor" adică al intervalelor de timp dintre două activități zilnice consecutive care ar trebui să fie cât mai mic etc.

Probleme propuse

1. Recapitulați următoarele chestiuni:
- ce este o problemă combinatorială?
- ce este o problemă de optimizare combinatorială? Dați un exemplu diferit de cele oferite în text şi discutați-l cu colegii şi cu profesorul dvs.
- ce înseamnă ca o problemă de optimizare este uşoară (grea)?

2. Arătați (prin redesenare) că graful



este bipartit.

3. Dacă într-o rețea rutieră, drumul cel mai scurt între localitățile A şi B trece prin localitatea C, arătați ca porțiunea din drum situată între A şi C este şi cel mai scurt drum posibil între A şi C.

4. Într-un graf oarecare G, **gradul** unui nod *x* este prin definiție **numărul muchiilor** având o extremitate în *x*.Probați că suma gradelor tuturor nodurilor din G este egală cu dublul numărului de muchii din graf. Deduceți că în orice graf numărul nodurilor de grad impar este întotdeauna par.

5. În timpul celui de al doilea război mondial mulți aviatori din țările ocupate de Germania s-au refugiat în Anglia înrolându-se în Forțele Aeriene Regale (RAF). Pentru a conduce un aparat erau necesari doi piloți iar dificultățile de alcătuire a echipajelor proveneau din faptul că cei doi coechipieri trebuiau să se înțeleagă între ei atât din punctul de vedere al limbii cât şi al altor "tipicuri" ce țin de arta de a conduce un avion pe timp de luptă. O problema curentă a comandamentului era aceea de a şti în orice moment câte avioane putea ridica în aer, altfel spus câte echipaje operaționale putea forma.

Să traducem chestiunea în limbaj de grafuri. Alcătuim un graf ale cărui noduri reprezintă piloții disponibili; două noduri vor fi unite printr-o muchie dacă piloții respectivi pot forma un echipaj apt pentru luptă. La ce problemă “teoretică” de optimizare combinatorială (din cele prezentate!) se reduce acest context?

6. a) Se consideră un graf simplu şi conex G=(V,E). O mulţime de muchii $A \subseteq E$ se numeşte **acoperire** a grafului G dacă orice nod $x \in V$ este extremitate pentru (cel puţin) o muchie din A. Evident, mulţimea E a tuturor muchiilor este o acoperire a lui G. Ce problemă de optimizare (combinatorială) se poate pune în acest context?

b) In Congresul SUA s-a pus problema alcătuirii unui comitet naţional cuprinzând cel puţin un reprezentant din partea celor 50 de state si cel puţin un reprezentant din fiecare din cele 65 de grupuri etnice recunoscute. Un număr de 170 de congresmeni si-au oferit serviciile pentru a lucra în cadrul acestui comitet. Din motive lesne de înteles, comitetul trebuia format din cât mai puţine persoane.

Deşi greu de crezut, multe chestiuni de politică se pot rezolva prin grafuri! În cazul de faţă, alcătuim un graf bipartit în care nodurile sunt împărţite în două submulţimi disjuncte: 50 noduri reprezentând statele si alte 65 de noduri corespunzătoare etniilor. Fiecăreia din cele 170 de persoane dispuse sa lucreze în cadrul comitetului îi va corespunde o muchie care uneşte nodul ce reprezintă statul de baştină cu nodul ce reprezintă etnia din care face parte. Este clar acum că un comitet care sa satisfacă condiţiile impuse se identifică cu o acoperire a grafului construit. Este această chestiune practică o ilustrare (poate chiar o justificare) a problemei teoretice de la a)?

7. Pentru o reparaţie la o instalaţie sanitară, nea Ion are nevoie de bucată de ţeavă lungă de 15 dm, două bucăţi de câte 10 dm şi alte 4 bucăţi de câte 6 dm.Deoarece la depozit nu găseşte decât ţevi lungi de 30 dm, nea Ion se întreabă de câte ar avea nevoie şi face următoarea socoteală: bucăţile mici de ţeavă însumează $1 \times 15 + 2 \times 10 + 4 \times 6 = 59$ dm aşa că pentru debitarea lor, două ţevi de 30 dm ar fi suficiente. Este o socoteală bună?

8. Aveţi trei vase: unul de 8 litri plin cu apă (sau vin sau altceva care vă place...) şi alte două vase goale unul de 5 litri şi celălalt de 3 litri. Cum se poate împărţi cantitatea de lichid în două părţi egale a 4 litri fiecare ştiind că nu este permisă decât o singură operaţie, aceea de a turna lichid dintr-un vas în altul (fără pierderi...). Care este numărul minim de operaţii pentru a obţine rezultatul dorit?

Încercaţi să formalizaţi problema construind un graf orientat în felul următor. Nodurile grafului corespund tuturor stărilor posibile în care se poate afla ansamblul celor trei vase după mutarea, repetată, a unor cantităţi de lichid dintr-un vas în altul. Mai precis, un nod va fi un triplet de numere întregi (*a,b,c*) reprezentând cantităţile de lichid existente în vasul de 8 l, de 5 l respectiv de 3 l după efectuarea mai multor operaţii de transfer de lichid (este clar că, indiferent de situaţie, $a + b + c = 8$). Se va trasa un arc de la nodul (*a,b,c*) la nodul (*a',b',c'*) numai în cazul în care din situaţia reprezentată de tripletul (*a,b,c*) se poate ajunge la situaţia (*a',b',c'*) printr-o singură operaţie de turnare de lichid dintr-un vas în altul.De exemplu, situaţia iniţială este reprezentată de tripletul (8,0,0). Dacă se toarnă lichid în vasul de 5 l până la umplere rezultă situaţia (3,5,0) iar în graf apare arcul (8,0,0) $\rightarrow$ (3,5,0). Din starea iniţială se poate ajunge şi în situaţia (5,0,3) umplând vasul de 3 l etc

La ce problemă de optimizare combinatorială (deja prezentată în curs!) se reduce problema dată?

Bibliografie

Nica, V., Capitole speciale ale Cercetărilor Operaționale, Centrul de învățământ la distanță, ASE, Bucureşti, 2001

Ciobanu, Gh., Nica, V. T., Mustață, Fl., Mărăcine, V., Mitruț, D., Cercetări Operaționale. Optimizări în rețele. Teorie şi aplicații economice, Editura Matrix Rom, Bucureşti, 2002

Minieka, E., Optimization Algorithms for Networks and Graphs, Marcel Dekker, New York and Basel, 1978

Lawler, E. L., Combinatorial Optimization. Networks and Matroids, Holt Rinehart and Winston, New York …, 1976

Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., Schrijver, A., Combinatorial Optimization, John Wiley&Sons, Inc., 1998

Unitatea de învățare 5

INTRODUCERE ÎN OPTIMIZAREA COMBINATORIALĂ
Drumuri şi arbori

Cuprins

5.1 Problema drumului de valoare minimă (Shortest Path Problem)
- **5.1.1 Definiții şi notații**
- **5.1.2 Algoritmul lui Dijkstra**
- **5.1.3 Algoritmul lui Dijkstra. Ilustrare numerică**

5.2 Problema arborelui de cost minim (Minimum Spanning Tree Problem)
- **5.2.1 Algoritmul lui Kruskal. Algoritmul lui Prim**
- **5.2.2 Versiunea "operațională a algoritmului lui Kruskal**

Probleme propuse

Obiectivul unității de învățare 5

Prezentarea unor metode algoritmice de rezolvare pentru:

- problema drumului de valoare minimă
- problema arborelui de cost minim

Cele două probleme sunt exemple reprezentative în clasa problemelor **"uşoare"** de optimizare combinatorială.

5.1 Problema drumului de valoare minimă (Shortest Path Problem)

Într-o rețea de transport sau de comunicare, problema determinării drumului de valoare minimă între două punct ale rețelei este des întâlnită.În contextele practice, valoarea unui drum poate însemna o distanță de parcurs, un timp de parcurgere, un cost sau măsura siguranței de transmisie.

Mulți algoritmi destinați rezolvării unor probleme de optimizare complexe utilizează ca subrutină proceduri de căutare a drumurilor de valoare minimă.

Problema determinării drumurilor de valoare minimă a fost deja anunțată în unitatea de învățare 4 ca una din problemele reprezentative „uşoare" ale optimizării combinatoriale.Rezolvarea problemei este scopul prezentei secțiuni.

5.1.1 Definiții şi notații

Fixăm un **graf simplu** şi **conex** G = (V, E) în care V este mulțimea **nodurilor** iar E este mulțimea **muchiilor**. Pentru maxima generalitate admitem că în G s-a dat şi o **orientare** a unora dintre muchii şi fie U mulțimea arcelor **permise** (pentru detalii conceptuale şi notaționale vezi secțiunea 4.1 din unitatea de învățare 4)

Reamintim că un **drum** în graful G este o succesiune de noduri $\lambda = (i_0, i_1, ..., i_p)$ cu proprietatea că perechile de noduri consecutive $(i_0, i_1), (i_1, i_2), ..., (i_{p-1}, i_p)$ sunt arce permise. Numărul *p* al acestor arce este prin definiție lungimea $l(\lambda)$ a drumului λ. Nodul i_0 este extremitatea inițială adrumului λ iar i_p este extremitatea finală. Celelalte noduri, adică $i_1, ..., i_{p-1}$ se numesc noduri intermediare, "prin care trece" drumul λ. Dacă $i_0 \equiv i_p$ spunem că λ este un **circuit** şi în acest caz convenim ca şi i_0 să fie socotit nod intermediar.

Vom spune că nodul *j* este un **vecin** al nodului *i* dacă există arcul permis (*i*,*j*); mai general vom spune că nodul *j* este **accesibil** din nodul *i* dacă există un drum de la *i* la *j* format numai din arce permise.

Chiar dacă graful G este presupus conex, este posibil ca un nod să nu poată fi accesat dintr-un altul şi asta din cauza orientărilor existente pe muchii! De exemplu, în graful din figura 5.1 nu există drumuri (formate din arce permise) de la nodul 1 la nodul 6 dar nodul 1 poate fi accesat din nodul 6 chiar în mai multe moduri!

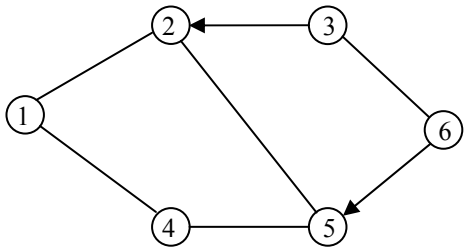


Figura 5.1

Presupunem în continuare că fiecărui arc permis $u \in U$ i s-a asociat un număr real $c(u)$ numit **valoarea** sau **ponderea** arcului *u*. Dacă $u = (i, j)$ vom folosi şi notația alternativă $c(i, j)$. Prin definiție,valoarea $c(\lambda)$ a drumului λ este suma valorilor arcelor alcătuitoare.
În continuare vom studia cazul cel mai simplu dar şi cel mai des întâlnit în aplicații în care toate ponderile arcelor permise sunt valori **nenegative.**

În graful G fixăm două noduri *s* şi *t* şi formulăm problema de optimizare:

P(*s*,*t*): Să se determine drumul de valoare minimă de la *s* la *t*.

Mai generală este problema:

P(*s*): Să se determine toate nodurile care sunt accesibile din *s* precum şi drumurile de valoare minimă de la *s* către ele.

Aceste drumuri formează un subgraf orientat numit graful drumurilor de valoare minimă cu origina în *s* şi notat $G^*(s)$.

Noua problemă subsumează problema inițială a determinării drumului de valoare minimă de la *s* la un nod particular *t* .Într-adevăr, dacă în $G^*(s)$ figurează şi nodul *t* atunci în acest subgraf vom găsi şi un drum de valoare minimă dela *s* la *t*.

5.1.2 Algoritmul lui Dijkstra

Toate procedurile cunnoscute de căutare a drumurilor de valoare minimă rezolvă problema P(*s*). Unele sunt capabile să rezolve şi cazul mai general în care ponderile arcelor premise pot fi şi negative.Pentru cazul mai simplu al ponderilor nenegative în totalitate, algoritmul lui DIJKSTRA (citeşte Deikstra), 1959, pare a fi cel mai adecvat.

Prezentarea algoritmului necesită câteva pregătiri.

1) Pe parcursul derulării procedurii, nodurile grafului G se impart în două categorii:

- Noduri **cercetate**: un nod *i* se consideră cercetat în momentul în care algoritmul "a găsit" un drum de valoare minimă de la nodul de plecare *s* la nodul *i*;
- Noduri **necercetate**.

La start, singurul nod cercetat este nodul de plecare *s*; toate celelalte noduri sunt declarate necercetate.

2) Să considerăm un nod cercetat *i*. Dintre toate nodurile *j* necercetate şi vecine cu *i* (adică există arcul permis (*i*,*j*)) reținem pe acela pentru care valoarea $c(i, j)$ este cea mai mică. Nodul reținut se declară **nod candidat asociat nodului cercetat** *i*. (candidat la... dobândirea calității de nod cercetat!) Este posibil ca pentru acelaşi nod cercetat să existe mai mulți candidați sau să nu existe nici unul după cum este posibil ca acelaşi nod necercetat să „candideze" din partea mai multor noduri cercetate!

În figura 5.2 este vizualizat un nod **cercetat** *i* împreună cu un **candidat** al său *j*. Nodul *i* fiind presupus cercetat, aceasta înseamnă că, în etapele anterioare, algoritmul a găsit un drum λ de valoare

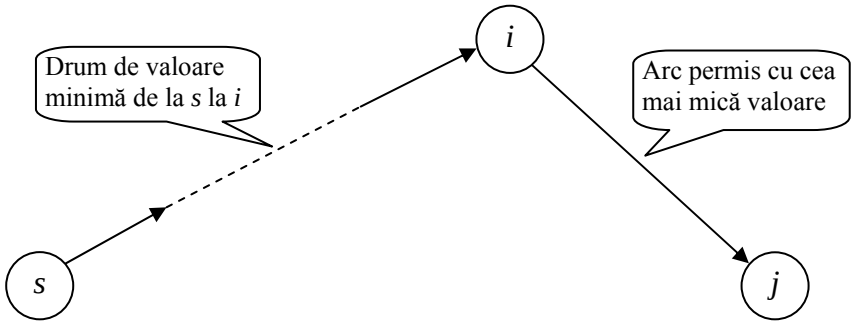


Figura 5.2

minimă de la *s* la *i*. Acest drum, completat cu arcul (*i*,*j*), reprezintă un drum de la nodul *s* la nodul (necercetat) *j* pe care îl vom numi **drumul asociat candidatului** *j* şi a cărui valoare este $c(\lambda) + c(i, j)$.

Ca urmare a faptului că un nod necercetat poate să candideze din partea mai multor noduri cercetate este posibil ca el să aibe mai multe drumuri asociate!

Cu aceste pregătiri putem trece la prezentarea algoritmului lui Dijkstra.

Start Nodul de plecare se declară nod cercetat şi toate celelalte noduri din graf se declară necercetare.

Pasul iterativ Pentru fiecare nod cercetat *i* se identifică candidatul sau candidații asociați lui *i* şi se calculează valorile drumurilor asociate acestor candidați. Nodul candidat *j* al cărui drum asociat are cea mai mică valoare va fi declarat nod cercetat. Teoria ne asigură că drumul asociat lui *j* este un drum de valoare minimă de la *s* la *j*. Se reia pasul iterativ.

Algoritmul se opreşte în momentul în care:

- Nu mai există noduri candidate:

sau

- Nodul de destinație *t* a fost declarat nod cercetat.

5.1.3 Algoritmul lui Dijkstra. Ilustrare numerică

În graful din figura 5.3 valorile numerice înscrise pe muchii reprezintă distanțe. Se cere determinarea celui mai scurt drum de la nodul O la nodul T. Atenție:absența orientărilor vrea să însemne că orice muchie poate fi parcursă în ambele sensuri (deci graful are 11 muchii şi 22 arce permise). Aplicăm algoritmul lui Dijkstra.

Start Nodul de plecare O este declarat cercetat; celelalte şase noduri ale grafului sunt declarate necercetate.

Iterația 1

Singurul nod cercetat O are trei vecini A,B,C dintre care, cel mai apropiat, este A. Nodul A va fi unicul candidat asociat nodului cercetat O. Declarăm cercetat nodul A; drumul cel mai scurt de la O la A se reduce la arcul OA. Reținem arcul OA pentru graful $G^*(O)$ al drumurilor de valoare minimă cu origina în O.

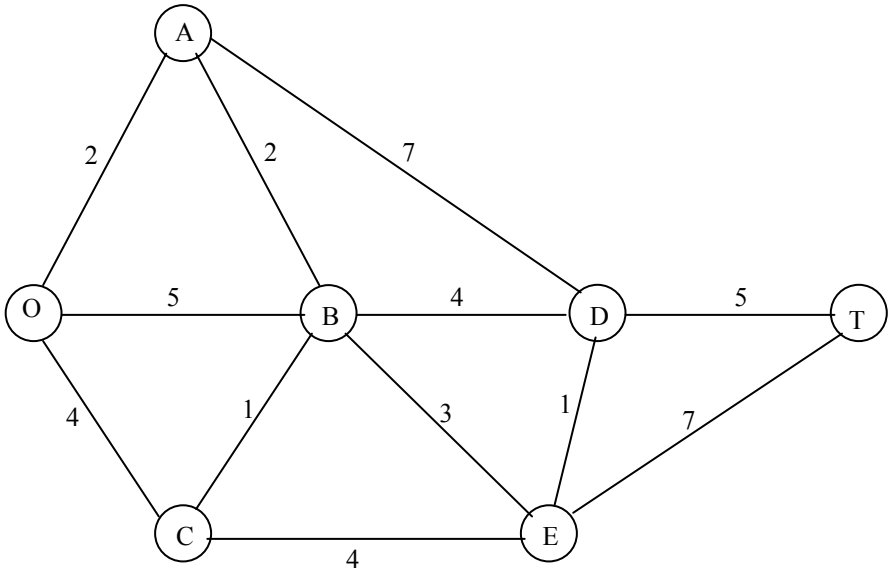


Figura 5.3

Iterația 2 Avem două noduri cercetate O şi A.

- Vecinii necercetați ai lui O sunt nodurile B şi C; deoarece este mai apropiat de O, nodul C este nodul candidat asociat – în această iterație – nodului O.Drumul asociat lui C se reduce la arcul OC şi are valoarea 4.
- Vecinii necercetați ai lui A sunt B şi D; candidat va fi nodul mai apropiat B. Drumul asociat nodului B se obține „prelungind" drumul de valoare minimă de la O la A – găsit la iterația 1 – cu arcul AB ; lungimea drumului este egală cu 2 + 2 = 4.
- Avem două noduri candidate C şi B ale căror drumuri asociate au aceeaşi lungime 4. Le declarăm pe amândouă ca fiind noduri cercetate. Conform teoriei ‚drumurile asociate sunt cele mai scurte drumuri către ele.

Iterația 3 În acest moment nodurile O, A, B şi C sunt cercetate.

- Nodul O nu mai are vecini necercetați.
- Nodul A are un singur vecin necercetat, nodul D, care va fi şi candidatul său. Drumul asociat are lungimea 2 + 2 = 4 iar ultimul său arc component este AD.
- Vecinii necercetați ai lui B sunt D şi E; candidatul lui B va fi nodul mai apropiat E. Drumul minim până la B are lungimea 4 (iterația 2) astfel că drumul asociat candidatului E va avea lungimea 4 + 3 = 7 (s-a adăugat lungimea ultimului arc BE).

- E va candida şi din partea lui C ca unic vecin necercetat şi ca urmare va avea un al doilea drum asociat, obținut prelungind drumul minim până la C (în fapt, arcul OC) cu arcul CE. Lungimea noului drum este 4 + 4 = 8.
- Comparăm lungimile drumurilor asociate construite. Cel mai scurt se termină în E, înainte de a ajunge în E trece prin B şi are lungimea 7. În consecință declarăm E nod cercetat şi reținem arcul BE ca ultim arc pe drumul cel mai scurt către E.

Iterația 4

La acest stadiu al derulării algoritmului sunt declarate cercetate nodurile O,A,B,C şi E.Sunt cunoscute valorile minimale ale drumurilor da la O către A,B,C şi E. Sunt reținute deasemenea „ultimele" arce ale drumurilor minimale; după cum vom vedea cunoaştera acestor arce va fi suficientă pentru reconstituirea drumurilor minimale din O către orice alt nod al grafului!
D este singurul nod candidat, asociat însă la numai puțin de trei noduri deja cercetate A,B şi E.Este clar că D va fi următorul nod declarat cercetat; rămâne să stabilim lungimea drumului minimal de la O la D şi ultimul său arc! Algoritmul ia în considerare trei drumuri de la O la D:

- drumul minim de la O la A, prelungit cu arcul AD; lungime: 2 + 7 = 9;
- drumul minim de la O la B, prelungit cu arcul BD; lungime: 4 + 4 = 8;
- drumul minim de la O la E, prelungit cu arcul ED; lungime: 7 + 1 = 8;

În concluzie, cele mai scurte drumuri de la O la D au lungimea 8 şi înainte de a ajunge în D trec, fie prin B fie prin E!

Iterația 5 Nodul T candidează:

- o dată din partea lui D - drumul asociat va avea lungimea 8 + 5 = 13;
- altă dată din partea lui E - drumul asociat va avea lungimea 7 + 7 = 14.

Prin urmare, cel mai scurt drum de la O la T va avea lungimea 13 iar ultimul său arc va fi DT.

Deoarece am „atins" nodul T algoritmul se opreşte.În cazul de față toate nodurile grafului au fost cercetate aşa încât algoritmul a găsit şi toate drumurile minimale cu origina în O!

Considerațiile de mai sus au fost sintetizate în tabelul 5.1

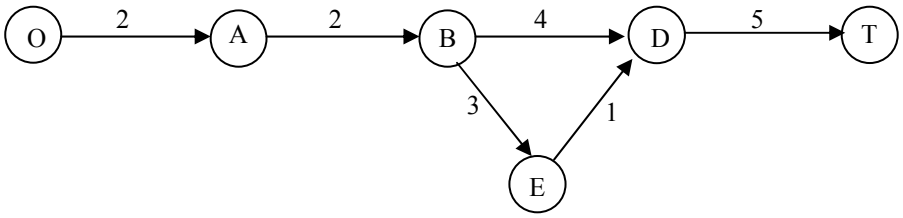


Figura 5.4

Determinarea efectivă a nodurilor prin care trece drumul cel mai scurt de la O la T sau la oricare alt nod se face din aproape în aproape „de la sfârşit către începutul drumului" folosind arcele reținute pe parcurs.

Există două drumuri optime indicate în figura 5.4:

În fapt, algoritmul a determinat drumurile de valoare minimă de la O la toate celelalte noduri – vezi figura 5.5

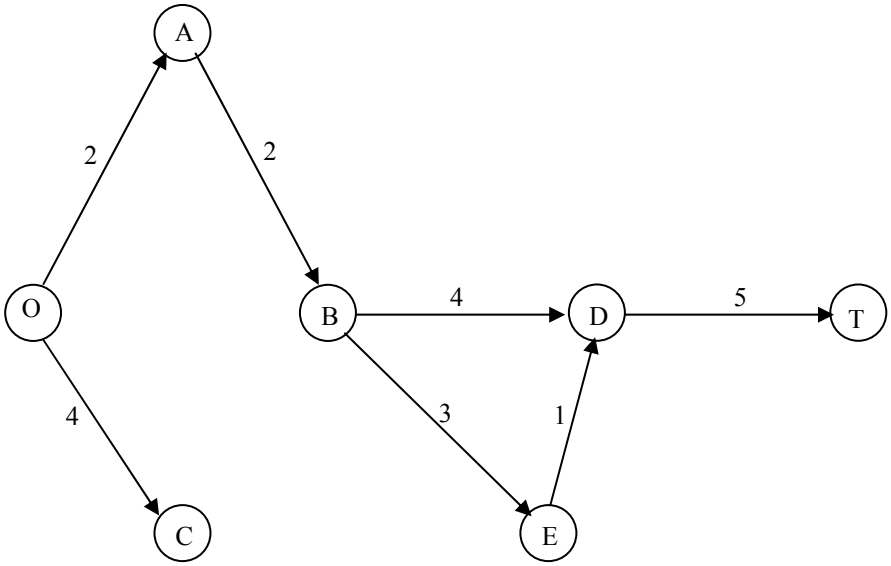


Figura 5.5

Tabelul 5.1

Iterația	Noduri cercetate cu vecini necercetați	Noduri (necercetate) candidate	Valoarea drumului asociat	Care nod a fost declarat cercetat	Valoarea minimă a drumului către nodul declarat cercetat	Ultimul arc pe drumul de valoare minimă
1	O	A	2	A	2	OA
2	O	C	4	C	4	OC
	A	B	2+2=4	B	4	AB
3	A	D	2+7=9			
	B	E	4+3=7	E	7	BE
	C	E	4+4=8			
4	A	D	2+7=9			
	B	D	4+4=8	D	8	BD
	E	D	7+1=8	D	8	ED
5	D	T	8+5=13	T	13	DT
	E	T	7+7=14			

5.2 Problema arborelui de cost minim (Minimum Spanning Tree Problem)

Dată în unitatea de învățare 4 ca un prim exemplu de problemă de optimizare combinatorială, problema arborelui de cost minim are numeroase aplicații practice în special în proiectarea rețelelor de comunicații. Contextul general este următorul: un număr de "puncte" trebuie conectate între ele pentru facilitarea transmiterii unui anumit serviciu (telefon, TV, calculator) Între puncte există "legături

potențiale" a căror realizare implică un anumit cost. Problema care apare este aceea de a vedea ce legături vor fi efectiv realizate astfel încât:

- oricare două puncte să fie conectate – direct sau indirect – în vederea utilizării serviciului;
- suma costurilor legăturilor realizate să fie minimă.

Problema arborelui de cost minim este importantă atât în sine cât şi prin faptul că apare ca subproblemă în chestiuni de optimizare combinatorială mai complexe cum ar fi problema comisvoiajorului

Exemplul 1 Centrul regional de calculatoare X trebuie să instaleze un număr de linii speciale de comunicații care să lege cinci utilizatori la un nou computer. Deoarece instalarea rețelei este costisitoare, conducerea centrului doreşte ca lungimea totală a liniilor instalate să fie cât mai mică. Deşi computerul central poate fi conectat direct cu toți utilizatorii ar fi mai economic să fie instalate linii directe doar către unii, ceilalți intrând în rețea prin legarea lor la utilizatorii deja conectați.Rețeaua legăturilor posibile este vizualizată prin graful din figura 5.6 Valorile numerice înscrise pe muchii reprezintă distanțe în km.

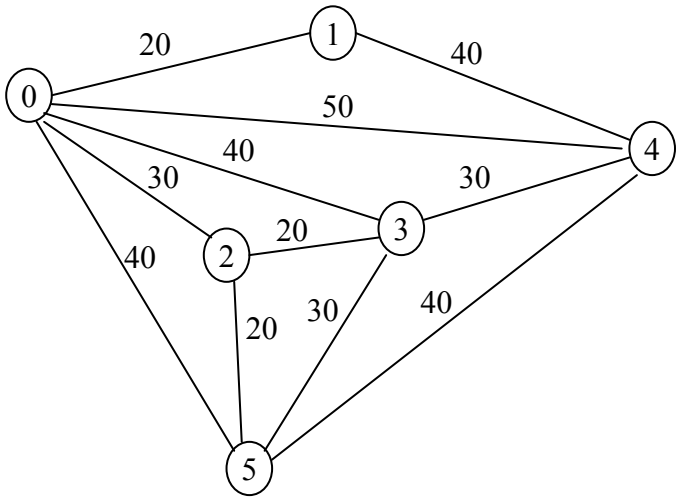


Figura 5.6

Exemplul 2 Prefectura județului X şi-a fixat ca obiectiv modernizarea rețelei drumurilor care leagă între ele localitățile 1,2,…,8, rețea vizualizată în figura 5.7. Pe fiecare muchie este înscrisă o valoare numerică reprezentând costul modernizării tronsonului respectiv (în milioane lei). Din cauza bugetului limitat, prefectura doreşte ca într-o primă etapă să modernizeze numai unele drumuri astfel încât:

- între oricare două localități să se poată circula – dacă se doreşte – pe tronsoane modernizate;
- costul întregii operații de modernizare parțială să fie minim.

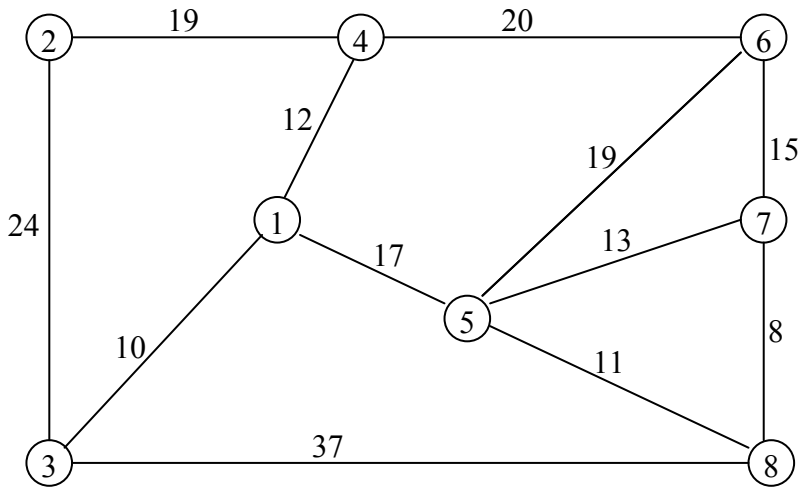


Figura 5.7

5.2.1 Algoritmul lui Kruskal. Algoritmul lui Prim

Reamintim datele problemei. Este dat un graf G = (V,E) ale cărui muchii au fost "ponderate" cu valori nenegative. Se doreşte determinarea unui **arbore** generator $H^* = (V,A)$ al cărui "cost" – dat de suma costurilor muchiilor alcătuitoare – să fie **minim**.

(Recomandăm cititorului să revadă noținiile de teoria grafurilor prezentate în unitatea de învățare 4, secțiunea 4.1)

Următorul algoritm construieşte arborele "minimal" H^* "muchie cu muchie" şi ca urmare construcția ar trebui să se termine în $n - 1$ paşi,dacă graful are n noduri (după cum este cunoscut, un arbore cu n noduri are exact $n - 1$ muchii…) Terminarea "mai devreme" a algoritmului semnalează faptul că graful original nu este conex şi în consecință nu are arbori generatori.

Algoritmul lui Kruskal (1956)

Start Muchiile grafului G vor fi cercetate în ordinea nedescrescătoare a costurilor asociate (muchiile cu acelaşi cost vor fi ordonate arbitrar).
Se selecteză o muchie e^1 cu cel mai mic cost posibil.

Pasul 1 Fie e^1 , e^2 ,…, e^k muchiile deja alese în etapele anterioare. Dacă $k = n - 1$, n fiind numărul nodurilor grafului, **Stop**: muchiile selectate sunt muchiile unui arbore (generator) de cost minim. Dacă $k < n - 1$ se trece la:

Pasul 2 Printre muchiile încă necercetate se caută o muchie, e^{k+1} , cu cel mai mic cost posibil şi care nu formează un ciclu cu (o parte din) muchiile e^1 , e^2 ,…, e^k. dacă o asemenea muchie nu există, **Stop**: graful G nu este conex şi prin urmare nu conține arbori generatori. Dacă e^{k+1} există, ea se adaugă la muchiile deja selectate şi se revine la pasul 1.

Exemplul 3 Vom aplica algoritmul lui Kruskal în graful conex din figura 5.6 Există trei muchii cu costul minim 20; ele nu formează un ciclu astfel că pot fi selectate în orice ordine, de exemplu: {0,1}, {2,3}, {2,5}. Mai departe a fost aleasă muchia {0,3} urmată de {3,4}; muchia {3,5}a fost respinsă deoarece forma un ciclu cu muchiile {2,3} şi {2,5} deja selectate! Procedura s-a oprit deoarece graful are şase noduri şi au fost alese cinci muchii.Arborele cu costul minim 120 (≡ lungimea totală a liniilor ce urmează a fi instalate!) este reprezentat în figura 5.8

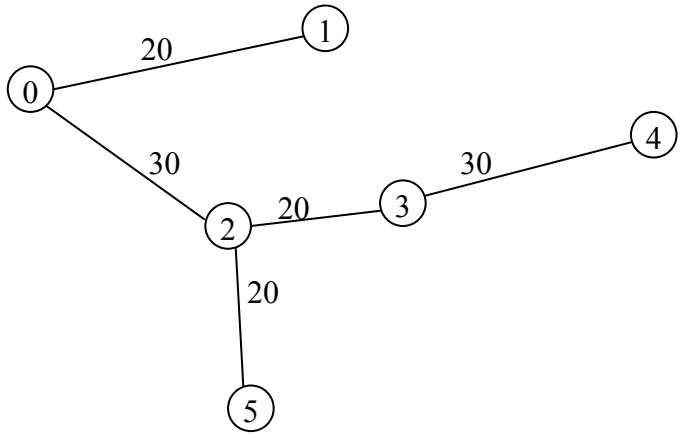


Figura 5.8

Foarte asemănător cu algoritmul lui Kruskal este:

Algoritmul lui Prim (1957)

Acesta pleacă de la un nod v^0 arbitrar ales şi caută să "atingă" progresiv toate celelalte noduri din G utilizând muchiile cu cel mai mic cost. Astfel, la start, muchia e^1 este muchia cu cel mai mic cost printre muchiile care au o extremitate în v^0 .Mai departe, dacă muchiile $e^1, e^2, ..., e^k$ au fost deja alese, e^{k+1} va fi – în caz că există – cea mai "ieftină" muchie care are o extremitate în comun cu **exact una** dintre muchiile $e^1, e^2, ..., e^k$.

Exemplul 4 Vom aplica algoritmul lui Prim grafului conex din figura 5.7 alegând nodul 1 ca "rădăcină" a arborelui.

- Selectăm muchia {1 ,3} ca fiind muchia cu cel mai mic cost şi cu o extremitate în nodul 1.
- Există patru muchii având o extremitate fie în nodul 1 fie în nodul 3; dintre ele selectăm muchia {1,4} fiind cea mai "ieftină".
- În continuare sunt alese muchiile {1,5} apoi {5,8}, {7,8}, {6,7} şi la urmă {2,4}

Arborele de cost minim şi care indică drumurile ce vor fi modernizate este vizualizat în figura 5.9. Costul operației de modernizare s-ar ridica la 92 milioane lei.

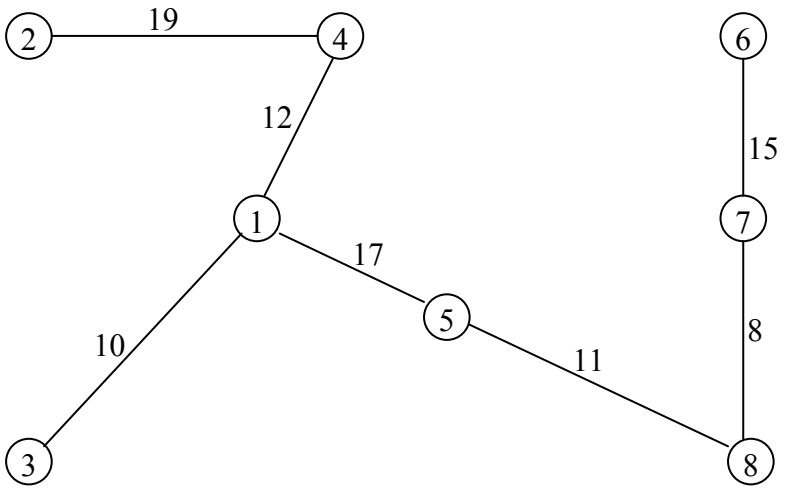


Figura 5.9

S-a specificat deja că problema arborelui de cost minim este una dintre problemele **„uşoare”** ale optimizării combinatoriale.Într-adevăr, se poate arăta că procedurile descrise mai sus rezolvă problema într-un **timp proporţional cu patratul numărului de muchii** din graful G!

5.2.2 Versiunea „operaţională” a algoritmului lui Kruskal

Pentru grafurile de dimensiuni moderate, care se pot „desena”, prezentarea de mai sus este suficientă: putem recunoaşte „cu ochiul liber” dacă graful original este conex sau dacă o muchie formează un ciclu cu altele.

Pentru grafurile cu un număr apreciabil de noduri şi muchii, cum sunt cele care provin din aplicaţiile reale este necesară o codificare adecvată a datelor şi instrucţiunilor.

În continuare se prezintă varianta **„operaţională”** a algoritmului lui Kruskal. Ca şi în algoritmul original muchiile grafului G sunt examinate una câte una, o singură dată, bineînţeles după listarea acestora în ordinea nedescrescătoare a costurilor.

În orice stadiu al aplicării procedurii, muchiile din G se împart în trei categorii:

- muchii examinate şi selectate pentru a face parte din arborele de cost minim;
- muchii examinate dar care nu au fost incluse în arborele de cost minim;
- muchii neexaminate.

La start toate muchiile sunt neexaminate. Diferenţierea pe parcurs a muchiilor examinate se face printr-o operaţie uzuală în teoria grafurilor, aceea de **colorare**.Concret, o muchie examinată şi inclusă în arborele de cost minim va fi „colorată” în **roşu** iar dacă a fost respinsă va fi colorată în **albastru**.

Subgraful H format din muchiile selectate până la un anumit moment este , în general, neconex. Vom numi **buchet** totalitatea nodurilor care fac parte dintr-o componentă conexă a subgrafului H. Este clar

acum că o muchie încă neexaminată va forma un ciclu cu muchiile deja selectate numai în cazul în care extremitățile ei aparțin unuia dintre buchetele existente!
Cu aceste pregătiri, varianta operațională a algoritmului lui Kruskal arată astfel.

Start - Într-o **listă** $\mathcal{L}$ se ordonează toate muchiile grafului în ordinea **nedescrescătoare** a costurilor aferente (muchiile cu costuri egale se ordonează arbitrar).

- Selectăm prima muchie e^1 din lista $\mathcal{L}$ şi o colorăm în **roşu.**
- Inițializăm o listă $\mathcal{B}$ a buchetelor ce vor fi construite pe parcurs cu buchetul format din extremitățile muchiei e^1.

Conținutul unei iterații:

Pasul 1 Dacă toate nodurile grafului G fac parte din acelaşi buchet (de altfel şi singurul din lista $\mathcal{L}$!) **Stop**: muchiile colorate în roşu formează un arbore (generator) de cost minim.
(Pentru realizarea practică a acestui pas este suficient să se memoreze numărul buchetelor din lista $\mathcal{B}$ şi numărul nodurilor din cel mai mare buchet.)
În caz contrar se trece la:

Pasul 2 Se cercetează dacă mai sunt muchii neexaminate (≡ necolorate!). Dacă nu mai sunt, **Stop**: graful original G nu este conex şi ca urmare nu conține arbori generatori. Altminteri se trece la:

Pasul 3 Se consideră **prima** muchie neexaminată (necolorată) $e^k = \{x, y\}$ din lista $\mathcal{L}$. Sunt posibile patru situații:

- Nodurile x , y nu fac parte din nici un buchet existent în lista $\mathcal{B}$. Colorăm e^k în roşu şi includem în $\mathcal{B}$ buchetul format din nodurile x , y;
- Unul din cele două noduri, să zicem x se regăseşte într-un buchet $b \in \mathcal{B}$ însă nodul y nu face parte din nici un buchet existent în lista $\mathcal{B}$.Colorăm e^k în roşu şi includem nodul y în buchetul b;
- Nodurile x, y fac parte din două buchete diferite b_1 şi b_2 . Colorăm e^k în roşu iar în lista $\mathcal{B}$ înlocuim buchetele b_1 şi b_2 prin reuniunea lor $b_1 \cup b_2$;
- Nodurile x , y fac parte din acelaşi buchet. Colorăm e^k în albastru.

Indiferent de situație se revine la pasul 1 în cadrul unei noi iterații.

Exemplul 5 Un graf are cinci noduri notate a,b,c,d,e şi opt muchii ale căror extremități şi ponderi sunt date în tabelul 5.2

Tabelul 5.2

Muchia	ab	ac	ad	bc	be	cd	ce	de
Cost	1	7	3	6	4	8	5	2

Ne propunem să determinăm un arbore de cost minim utilizând algoritmul lui Kruskal în versiunea descrisă mai sus.

Mai întâi relistăm muchiile în ordinea nedescrescătoare a costurilor – vezi tabelul 3.

Tabelul 5.3

Muchia	ab	de	ad	be	ce	bc	ac	cd
Cost	1	2	3	4	5	6	7	8

Acțiunea algoritmului şi în particular dinamica listei buchetelor $\mathcal{B}$ pot fi urmărite în tabelul 5.4 Se observă că algoritmul s-a oprit mai înainte de examinarea tuturor muchiilor!

Tabelul 5.4

Iterația	Muchia examinată	Culoarea	Lista $\mathcal{B}$ a buchetelor	Nr. Buchetelor din $\mathcal{B}$	Numărul nodurilor din cel mai mare buchet
Start	-	Toate muchiile necolorate	$\varnothing$	0	0
1	ab	roşu	{a ,b}	1	2
2	de	roşu	{a ,b} , { d , e}	2	2
3	ad	roşu	{a , b , d , e}	1	4
4	be	albastru	{a , b , d , e}	1	4
5	ce	roşu	{a , b , c , d , e}	1	5 **stop!**

Invităm cititorul să „deseneze” graful original şi arborele de cost minim construit de algoritm.

Exemplul 6 Un graf G are nouă noduri notate a,b,c,d,e,f,g,h,k şi zece muchii ale căror extremități sunt:

ab , ac , ad , bc , bd , cd , ef , eg , fg , hk. Fără a „desena” graful să se cerceteze dacă el este conex.

Tabelul 5.5

Iterația	Muchia examinată	Culoarea	Lista $\mathcal{B}$ a buchetelor	Nr. Buchetelor din $\mathcal{B}$
Start	-	Toate muchiile necolorate	$\varnothing$	0
1	ab	roşu	{a ,b}	1
2	ac	roşu	{a ,b,c}	1
3	ad	roşu	{a , b , c , d}	1
4	bc	albastru	{a , b , c , d}	1
5	bd	albastru	{a , b , c , d }	1
6	cd	albastru	{a , b , c , d }	1
7	ef	roşu	{a , b , c , d },{e,f}	2
8	eg	roşu	{a , b , c , d },{e,f,g}	2
9	fg	albastru	{a , b , c , d },{e,f,g}	2
10	hk	roşu	{a , b , c , d },{e,f,g},{h ,k}	3

Ştim că un graf este conex dacă şi numai dacă are arbori generatori. Căutăm un asemenea arbore aplicând algoritmul lui Kruskal în versiunea „cu buchete” şi considerând că toate muchiile au aceeaşi pondere, să zicem 1. Muchiile vor fi examinate atunci într-o ordine oarecare, de exemplu în ordinea în care au fost date în enunț – vezi tabelul 5.5

Concluzii:

- Graful G nu este conex. El are trei componente conexe: G_1 cu vârfurile a, b, c, d, G_2 cu vârfurile e, f, g şi G_3 cu două vârfuri h şi k.
- Muchiile colorate în roşu indică câte un arbore generator în fiecare din aceste componente – vezi figura 5.10 în care muchiile arborilor au fost îngroşate.

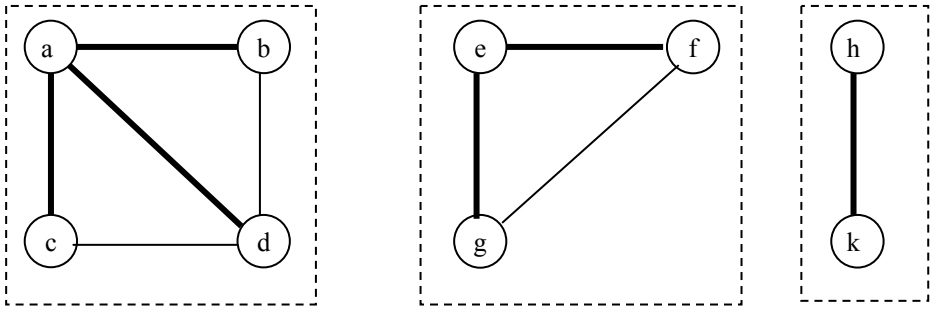


Figura 5.10

Probleme propuse

1. În rețeaua din figura 5.11, valorile numerice înscrise pe muchii reprezintă costuri de deplasare valabile în ambele sensuri de parcurgere. Aplicați riguros algoritmul lui Dijkstra pentru determinarea drumului de la nodul O la nodul T cu cel mai mic cost.

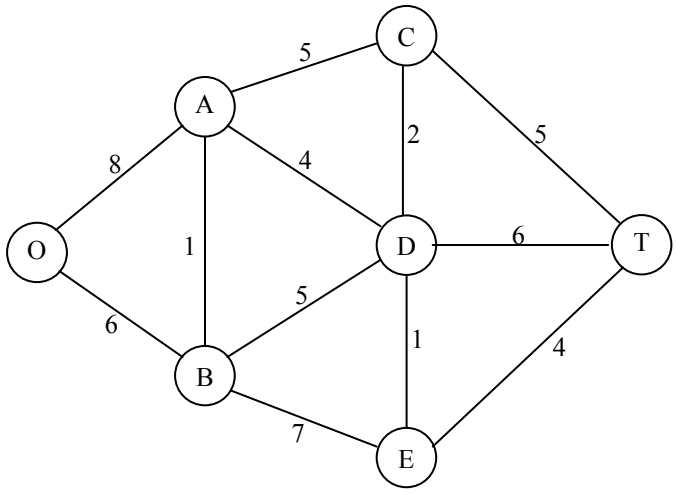


Figura 5.11

2. i) Utilizând algoritmul lui Dijkstra determinați cele mai scurte drumuri de la nodul A la celelalte noduri ale grafului din figura 5.12

ii) Ce modificări intervin în graful $G^*(A)$ al drumurilor de lungime minimă dacă muchiile {I, G} şi {D, J} nu pot fi parcurse decât de la I la G, respectiv de la D la J?

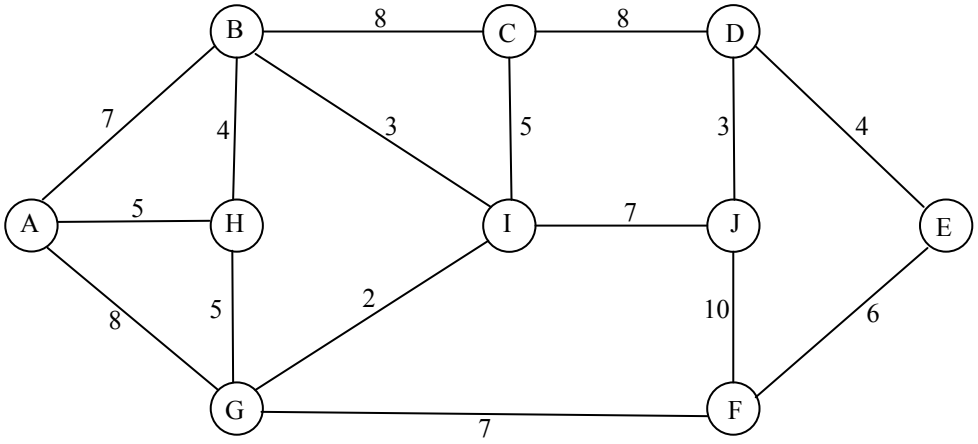


Figura 5.12

3. Utilizați algoritmul lui Kruskal pentru a determina arborele de valoare minimă în graful din figura 5.13

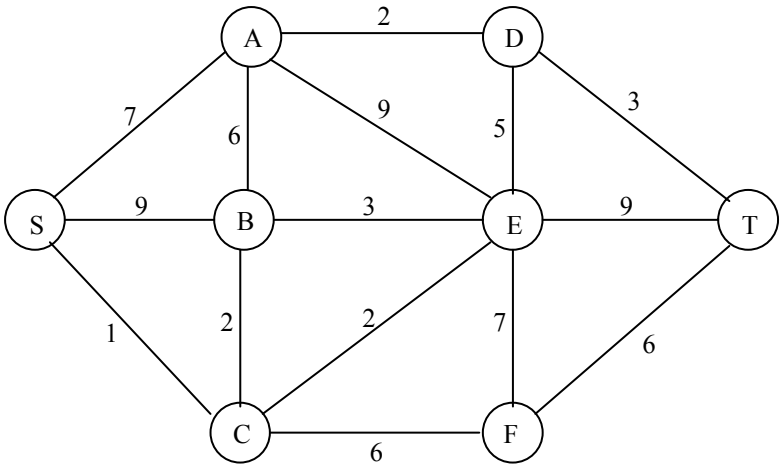


Figura 5.13

4. Agenția Națională de Turism a obținut autorizația de amenajare a unui nou parc natural într-o zonă cu un mare potențial turistic. Au fost identificate opt locații pentru construcția de cabane, căsuțe de odihnă, spații pentru picnic şi scene în aer liber. În zonă nu există drumuri astfel că ele trebuie amenajate. Experții agenției au stabilit ce drumuri ar putea fi construite şi ce lungime (în km) ar avea aceste drumuri, astfel încât orice locație să fie accesibilă – vezi figura 5.14 Pentru protejarea mediului este de dorit ca lungimea totală a acestor drumuri să fie minimă.

i (Ce drumuri ar trebui construite?

ii) Chiar înainte de demararea lucrărilor, o organizaşie ecologistă a obținut în instanță interzicerea construirii unuia dintre tronsoanele E – G sau E - F (altfel spus nu pot fi amenajate amândouă!). Care ar fi soluția cea mai avantajoasă pentru constructor?

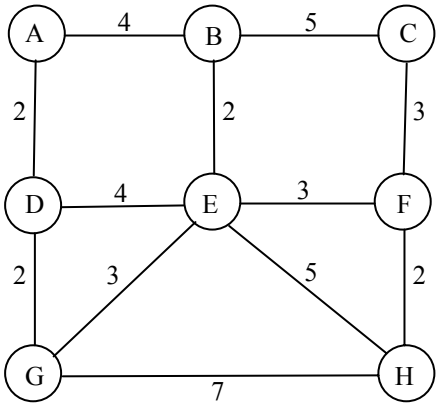


Figura 5.14

5. Departamentul Parcurilor Naționale intenționează să amenajeze în scopuri turistice o zonă puțin umblată. Au fost identificate patru locații pentru accesul automobilelor. Locațiile – inclusiv punctul de intrare în parc – şi distanțele dintre ele (în km) sunt date în tabelul 5.6. Pentru protejarea mediului dar şi pentru a calma protestul ecologiştilor, departamentul doreşte ca lungimea totală a drumurilor ce vor fi amenajate în scopul accesării oricărei locații să fie cât mai mică. Ce drumuri ar trebui construite în vederea atingerii scopului propus? Faceți şi o schiță.

Tabelul 5.6

	Dealul Negru	Stânca	Lunca Verde	Poiana Soarelui
Intrare în parc	7.1	19.5	19.1	25.7
Dealul negru	*	8.3	16.2	13.2
Stânca		*	18.1	5.2
Lunca Verde			*	17.2

Indicație: Este vorba de proiectarea unei rețele care să conecteze cinci locații: **I** ≡ Intrarea în parc; **D** ≡ Dealul Negru; **S** ≡ Stânca; **L** ≡ Lunca Verde; **P** ≡ Poiana Soarelui şi să aibe cea mai mică lungime. Echivalentul teoretic este determinarea unui arbore de lungime minimă într-un **graf complet** cu cinci noduri **I**, **D**, **S**, **L**, **P** şi $\binom{5}{2}$ = combinări de cinci luate câte două! = 10 muchii ale căror lungimi sunt date în tabel.

6. O bancă intenționează să instaleze un sistem de conectare a computerelor filialelor sale la computerul sediului central. O filială nu trebuie conectată neapărat la sediul central; este suficient ca ea să fie conectată la o altă filială deja conectată, fie direct fie indirect, la sediul central.

Tabelul 5.7

	Filiala 1	Filiala 2	Filiala 3	Filiala 4	Filiala 5
Sediul central	190	70	115	270	160
Filiala 1	*	100	240	215	50
Filiala 2		*	140	120	220
Filiala 3			*	175	80
Filiala 4				*	310

În tabelul 5.7 sunt date distanțele (în km) dintre sediul central şi filiale ca şi distanțele dintre filiale. Costul instalării unei linii telefonice speciale între două locații este direct proporțional cu distanța dintre locații şi suficient de mare pentru ca banca să-şi dorească elaborarea unei soluții de conectare cu cel mai mic cost.

Cum arată soluția care satisface obiectivul băncii? Faceți o schiță.

Unitatea de învățare 6

INTRODUCERE ÎN OPTIMIZAREA COMBINATORIALĂ
Problema comisvoiajorului (Traveling Salesman Problem ≡ TSP)

Cuprins

6.1 Formularea problemei. Clasificare. Aplicații
6.2 Problema firmei de curierat rapid
6.3 Un algoritm de tip B&B pentru TSP în cazul asimetric
6.3.1 Algoritmul lui Eastman
6.3.2 Aplicație la problema firmei de curierat rapid
6.4 Euristici de rezolvare suboptimală a TSP euclidiene
6.4.1 Euristica: mergi la cel mai apropiat vecin
6.4.2 Euristica: ajustare locală
6.4.3 Euristica: inserează puncul cel mai depărtat
6.4.4 Euristica: dublează muchiile unui arbore minimal
6.4.5 Euristica lui Cristofides

Probleme propuse

Obiectivul unității de învățare 6

Problema comisvoiajorului a fost déjà introdusă în unitatea de învățare 4 ca exemplu reprezentativ de problemă de optimizare combinatorială "grea". Tot acolo au fost examinate câteva posibilități de modelare cu ajutorul grafurilor sau a programării în numere întregi. TSP are numeroase aplicații practice; unele au fost déjà semnalate, altele vor fi date în continuare. În ciuda simplității sale, „se cere determinarea celui mai scurt traseu de vizitare a unor puncte cu întoarcerea în locul de plecare", rezolvarea problemei comisvoiajorului rămâne o veritabilă provocare mai cu seamă în cazul în care numărul punctelor este "mare".

Obiectivul unității de învățare 6 este de a trece în revistă câteva metode de rezolvare – în general suboptimală – a TSP cu ilustrările numerice de rigoare.

6.1 Formularea problemei. Clasificare. Aplicații

Exemplul 1 O companie petrolieră are mai multe platforme de foraj marin. Periodic, un elicopter cu baza pe uscat, vizitează platformele în vederea efectuării unor controale de rutină după care se întoarce la bază. În ce ordine vor fi vizitate platformele pentru ca distanța totală parcursă să fie minimă?

Exemplul de mai sus constituie una din numeroasele concretizări ale **problemei comisvoiajorului** (abreviat **TSP** ≡ **T**raveling **S**alesman **P**roblem):

Un comisvoiajor în localitatea 0 doreşte să viziteze localitățile 1,2,…,*n* la sfârşitul călătoriei el întorcându-se de unde a plecat. Nu există nici o restricție în ceeace priveşte deplasarea de la un oraş la altul. Pentru oricare două oraşe diferite $i \neq j$ se cunoaşte costul c_{ij} al deplasării comisvoiajorului din *i* în *j* (în alte contexte c_{ij} poate reprezenta distanța dintre oraşele *i* şi *j* sau timpul necesar efectuării deplasării din *i* în *j*) În ce ordine vor fi vizitate localitățile 1,2,…,*n* astfel încât să se minimizeze costul total al călătoriei (sau distanța totală parcursă sau durata totală a deplasării).

O problemă a comisvoiajorului se numeşte **simetrică** dacă:

$$c_{ij} = c_{ji} \text{ pentru orice } i \neq j$$

altminteri ea se zice **asimetrică**.

Între problemele simetrice, o clasă importantă este formată din acelea în care "costurile" c_{ij} verifică **inegalitatea triunghiului**:

$$c_{ij} + c_{jk} \geq c_{ik} \text{ pentru orice tripletă de indici diferiți } i, j, k.$$

Aceste probleme se numesc **euclidiene.**

De exemplu, o problemă care implică deplasări **în linie dreaptă** este o problemă euclidiană. Iată un exemplu de situație practică care nu implică "deplasări" şi care conduce totuşi la o TSP, în general asimetrică.

Exemplul 2 Stabilirea ordinii de lansare în execuție a unor repere. Pe un utilaj se execută mai multe repere (operații). Fiecare reper (operație) are o durată de execuție cunoscută. De multe ori se întâmplă ca trecerea de la executarea unui reper la executarea altuia să necesite un anumit timp de pregătire al utilajului mai mic sau mai mare funcție de caracteristicile celor două repere dar şi de ordinea în care acestea sunt procesate. De exemplu, într-o situație de croire, trecerea de la utilizarea (repetată) a unei rețete de debitare la o altă rețetă implică o "întrerupere" a activității, necesară repoziționării cuțitelor de tăiere.

În acest context, se pune problema stabilirii ordinii de lansare în execuție a reperelor astfel încât suma timpilor de pregătire să fie minimă.

6.2 Problema firmei de curierat rapid

Problema firmei de curierat rapid (**D**ial **a R**ide **P**roblem, abreviat **DARP**) are următoarea formulare generală:

Sunt date o rețea şi o listă de comenzi de transport. În fiecare comandă se specifică sursa şi destinația transportului care trebuie să fie noduri ale rețelei.Transporturile sunt efectuate de un server care pleacă dintr-un anumit nod al rețelei şi care, după satisfacerea tuturor comenzilor, se întoarce în punctul de plecare. Capacitatea serverului este limitată în sensul că el nu poate transporta mai mult de un anumit număr de comenzi simultan.Toate deplasările serverului se fac pe muchiile rețelei. Unele muchii pot fi parcurse în ambele sensuri altele doar într-un singur sens, prestabilit. Se presupun cunoscute lungimile tuturor muchiilor din rețea.
Problema de optimizare constă în a stabili cum vor fi transportate comenzile pentru ca distanța totală parcursă de server să fie minimă.

În unele situații, o comandă poate prevedea şi o fereastră de timp în care are loc fie preluarea comenzii de la sursă fie predarea acesteia la destinație.

Problema are numeroase aplicații:

- în activitatea firmelor de curierat rapid sau taximetrie;
- în transportul persoanelor cu handicap locomotor;
- în transportul pe verticală de mărfuri sau persoane cu ajutorul ascensorului;
- în deplasarea pe orizontală a unor obiecte cu ajutorul unui pod rulant, macara sau braț mecanic.

DARP este o generalizare a problemei comisvoiajorului. Într-adevăr, orice TSP se identifică cu o DARP în care:

- comenzile corespund oraşelor de vizitat;
- sursa şi destinația fiecărei comenzi coincid cu oraşul corespunzător comenzii.

În consecință, **DARP este o problemă de optimizare combinatorială grea.**

În continuare vom studia DARP în cel mai simplu caz, în care:

-serverul nu poate transporta o dată mai mult de o comandă;
- nu există ferestre de timp.

şi vom arăta că această problemă particulară este echivalentă cu o problemă a comisvoiajorului, asimetrică.

Vom folosi notațiile:

- G ≡ rețeaua de transport cu nodurile 0,1,…,*n*, nodul 0 fiind punctual de plecare şi întoarcere al serverului;

- $d_{ij} \equiv$ distanța de la nodul *i* la nodul *j* măsurată pe drumul cel mai scurt existent în rețea între *i* și *j*. După cum se știe, determinarea celor mai scurte drumuri între nodurile unui graf este o problemă de optimizare (combinatorială) **uşoară**, rezolvabilă de exemplu cu algoritmul lui FLOYD. Întrucât este posibil ca unele muchii să fie parcurse doar într-un sens, se poate întâmpla ca $d_{ij} \neq d_{ji}$ pentru anumite perechi de noduri.
- $R = \{r_1, r_2, ..., r_m\} \equiv$ lista comenzilor de transport cu $r_i = (s_i, t_i)$ unde s_i este sursa comenzii r_i iar t_i este destinația ei. s_i și t_i sunt noduri ale grafului G.

Definim un **traseu complet** al serverului ca fiind o succesiune de noduri și arce **permise** din graful G:

$$\lambda : 0 \equiv i_0 \rightarrow i_1 \rightarrow i_2 \rightarrow \cdots \rightarrow i_{p-1} \rightarrow i_p \equiv 0 \qquad (1)$$

care începe și sfârșește în nodul 0 și are proprietatea:

Pentru fiecare comandă $r_i = (s_i, t_i)$ nodurile s_i , t_i se găsesc printre nodurile din (1), nodul t_i apărând **în urma** nodului s_i.

Deoarece serverul nu poate transporta mai mult de o comandă o dată, orice traseu complet este o **reuniune de drumuri** de forma:

$$\lambda \equiv \mu_1 \cup \lambda_1 \cup \mu_2 \cup \lambda_2 \cup \quad \cdots \cup \mu_m \cup \lambda_m \cup \mu_{m+1} \qquad (2)$$

unde:

$\mu_1 \equiv$ porțiunea din λ de la nodul 0 la sursa primei comenzi transportate;
$\lambda_1 \equiv$ porțiunea din λ de la sursa la destinația primei comenzi transportate;
$\mu_2 \equiv$ porțiunea din λ de la destinația primei comenzi la sursa celei de a doua comenzi transportate ș.a.m.d.
…
$\mu_{m+1} \equiv$ porțiunea din λ pe care serverul o parcurge de la destinația ultimei comenzi transportate la locul de parcare 0.

Este posibil ca unele dintre drumurile "în gol" $\mu_1, \mu_2, ..., \mu_{m+1}$ să se reducă la un nod și astfel să aibe lungimea zero! Aceasta se întâmplă atunci când destinația unei comenzi coincide cu sursa următoarei comenzi transportate.
Este clar că, pentru minimizarea distanței totale parcurse:

- serverul va transporta fiecare comandă $r_i = (s_i, t_i)$ pe drumul **cel mai scurt** de la s_i la t_i;
- de la destinația unei comenzi transportate la sursa următoarei comenzi, serverul va trebui să se deplaseze pe drumul **cel mai scurt**!

În consecință, în structura unui traseu complet dată în (2) putem presupune că $\lambda_1, \lambda_2, ..., \lambda_m$ sunt cele mai scurte drumuri pe care se pot transporta comenzile și ca urmare suma

$d(\lambda_1)+d(\lambda_2)+\cdots+d(\lambda_m)$ a lungimilor acestor drumuri poate fi considerată o **constantă**, **independentă de ordinea** în care vor fi transportate comenzile.

Tot aşa, putem presupune că drumurile "în gol" $\mu_1,\mu_2,....,\mu_{m+1}$ sunt cele mai scurte drumuri posibile, însă structura lor şi de aici suma lungimilor lor depinde **esențial de ordinea** în care comenzile sunt transportate!

În concluzie, minimizarea distanței totale parcurse de server este echivalentă cu determinarea ordinii în care vor fi transportate comenzile astfel încât **suma lungimilor drumurilor "în gol" să fie minimă**.

Aceasta este o problemă a comisvoiajorului, asimetrică, în care:

- "oraşele" sunt comenzile $r_0,r_1,....,r_m$ unde $r_0=(0,0)$ este o comandă fictivă cu sursa şi destinația concentrate în nodul 0;
- pentru oricare două oraşe ≡ comenzi diferite $r=(s,t)$ şi $r'=(s',t')$ "costul $c_{r,r'}$ al deplasării de la r la r' " este lungimea celui mai scurt drum de la destinația t a comenzii r la sursa s' a comenzii următoare r'.

6.3 Un algoritm de tip B&B pentru TSP în cazul asimetric

Reamintim că problema comisvoiajorului a fost modelată ca o problemă de afectare:

$$\begin{cases} (\min) z=\sum\limits_{i=0}^{n}\sum\limits_{j=0}^{n} c_{ij}x_{ij} \\ \sum\limits_{j=0}^{n} x_{ij}=1 \qquad i=0,1,..,n \\ \sum\limits_{i=0}^{n} x_{ij}=1 \qquad j=0,1,..,n \\ x_{ij}\in\{0,1\} \end{cases} \tag{1}$$

cu restricții „speciale", menite să elimine apariția subtraseelor, variabilele bivalente x_{ij} având semnificația:

$$x_{ij}=\begin{cases} 1 \text{ daca comisvoiajorul merge direct din orasul } i \text{ in orasul } j \text{ si } i\neq j \\ 0 \text{ in caz contrar sau daca } i=j \end{cases}$$

unde $i,j=0,1,...,n$

(vezi unitatea de învățare 4, secțiunea 4.2)

6.3.1 Algoritmul lui Eastman

Următorul algoritm, datorat lui EASTMAN (1958), determină traseul optim al unei probleme **asimetrice** a comisvoiajorului prin rezolvarea unei liste de probleme de **afectare** care diferă de problema (1) prin unele costuri c_{ij} schimbate în $+\infty$.

Algoritmul utilizează:

- locație x_{CMB} care reține **C**el **M**ai **B**un traseu găsit pe parcurs;
- locație z_{CMB} care păstrează valoarea traseului depus în x_{CMB};
- listă $\mathcal{L}$ de probleme de afectare, rezolvabile cu algoritmul ungar.

Start Inițializăm:

$x_{CMB} = \varnothing$ $\quad z_{CMB} = +\infty$ $\quad \mathcal{L} = \{$problema de afectare (1)$\}$

(Locația x_{CMB} se poate inițializa şi cu un traseu fie cunoscut fie determinat printr-o euristică oarecare. Valoarea traseului se înregistrează în z_{CMB})

Conținutul unei iterații:

Pasul 1 Dacă lista $\mathcal{L}$ este vidă, **Stop**: traseul de valoare minimă se găseşte în x_{CMB}. În caz contrar selectăm **prima** problemă de afectare din listă. Problema selectată se şterge din $\mathcal{L}$. Se trece la:

Pasul 2 Se rezolvă problema de afectare selectată (cu algoritmul ungar). Dacă valoarea optimă a funcției obiectiv este $\geq z_{CMB}$ se revine la pasul 1. Dacă valoarea optimă a funcției obiectiv este $< z_{CMB}$ se trece la:

Pasul 3 Examinăm soluția optimă a problemei de afectare rezolvate la pasul 2. Dacă această soluție este un **traseu complet** se actualizează x_{CMB} şi z_{CMB} cu traseul găsit, respectiv cu valoarea acestuia, după care se revine la pasul 1. Dacă soluția examinată este o **reuniune de subtrasee disjuncte** se trece la:

Pasul 4 Se alege subtraseul **cu cele mai puține arce**. Pentru fiecare arc (i , j) al subtraseului ales se adaugă, **în capul listei** $\mathcal{L}$, problema de afectare rezultată din problema rezolvată punând $c_{ij} = +\infty$ (deci la lista $\mathcal{L}$ se vor adăuga atâtea probleme, câte arce are subtraseul ales!). Se revine la pasul 1.

Observații: 1) Rațiunea pasului 4 este următoarea: dacă $i \to j \to k \to \cdots \to i$ este subtraseul ales, este clar că traseul optim nu va conține unul din arcele $i \to j$ sau $j \to k$ sau … ,ce compun subtraseul.Altfel spus , în soluția optimă a TSP vom avea $x_{ij} = 0$ sau $x_{jk} = 0$ sau …În consecință, traseul optim va fi căutat traseele în care $x_{ij} = 0$ apoi cele în care $x_{jk} = 0$ etc. Or, restrângerea la traseele în care $x_{ij} = 0$ se face punând $c_{ij} = +\infty$!

Bineînțeles că putem considera orice subtraseu din soluția problemei rezolvate la pasul 2. Alegerea celui "mai mic" se face în ideea de a nu încărca "excesiv" lista $\mathcal{L}$!

2) În termenii generali ai principiului Branch & Bound, **ramificarea** are loc la pasul 4 iar **mărginirea** la pasul 2 (nu „strică" revederea secțiunii 3.2 din unitatea de învățare 3!!)

3) Algoritmul este aplicabil problemelor **asimetrice** de dimensiune **moderată.**Punctul slab al procedurii îl constituie **impredictibilitatea** dimensiunii listei $\mathcal{L}$ ca şi a dinamicii acestei liste!
În principiu, algoritmul poate rezolva exact orice TSP însă aplicarea lui la cazul simetric nu este recomandată fiind foarte greoaie!

Exemplu numeric Se dă problema asimetrică a comisvoiajorului cu cinci localități 0, 1,...,4 definită de următoarea matrice a costurilor de deplasare c_{ij} :

Tabelul 6.1

	0	1	2	3	4
0	∞	10	25	25	10
1	1	∞	10	15	2
2	8	9	∞	20	10
3	14	10	24	∞	15
4	10	8	25	27	∞

0 este punctul de plecare şi întoarcere al comisvoiajorului. După cum se vede costul deplasării între oraşele 0 şi 4 nu depinde de sens: $c_{04} = c_{40} = 10$. În schimb, deplasarea de la 0 la 1 este mai „scumpă" decât deplasarea în sens invers: $c_{01} = 10 > 1 = c_{10}$. Se pune problema determinării unui traseu de vizitare a localităților cu plecarea şi întoarcerea în 0, care să treacă o singură dată prin 1, 2, 3 şi 4 şi care să coste cât mai puțin. Pentru aceasta vom aplica algoritmul descris mai sus.

Start

Inițializăm: $x_{CMB} = \varnothing$ $z_{CMB} = +\infty$ $\mathcal{L} = \{\text{PA}\}$

unde PA este problema de afectare (1) cu costurile c_{ij} din tabelul 6.1

Iterația 1

1 Selectăm problema PA şi o ştergem din listă: $\mathcal{L} = \varnothing$;
2 Rezolvăm problema selectată; se obține soluția optimă:

$$x_{04} = x_{31} = x_{12} = x_{23} = x_{31} = 1 \quad , \quad x_{ij} = 0 \text{ în rest.}$$

care corespunde subtraseelor disjuncte $0 \to 4 \to 0$ şi $1 \to 2 \to 3 \to 1$.

- Selectăm subtraseul $0 \to 4 \to 0$ şi adăugăm la lista $\mathcal{L}$ problemele de afectare PA_1 şi PA_2 derivate din PA prin schimbarea $c_{04} = +\infty$ $c_{40} = +\infty$:

$$\mathcal{L} = \{\text{PA}_1 , \text{PA}_2\}$$

Iterația 2

- Selectăm problema PA_1 şi actualizăm lista : $\mathcal{L} = \{PA_2\}$.
- Rezolvarea problemei PA_1 conduce la soluția

$$x_{03} = x_{31} = x_{12} = x_{24} = x_{40} = 1 \quad , \quad x_{ij} = 0 \text{ în rest}$$

care este un traseu complet cu valoarea 65. Actualizăm:

$$x_{CMB} = \{0 \to 3 \to 1 \to 2 \to 4 \to 0\} \qquad z_{CMB} = 65$$

Iterația 3

- Selectăm problema PA_2. Lista $\mathcal{L}$ devine vidă.
- PA_2 are soluția optimă:

$$x_{04} = x_{41} = x_{12} = x_{23} = x_{30} = 1 \quad , \quad x_{ij} = 0 \text{ în rest}$$

care este un traseu complet cu valoarea 62. Actualizăm:

$$x_{CMB} = \{0 \to 4 \to 1 \to 2 \to 3 \to 0\} \qquad z_{CMB} = 62$$

Iterația 4

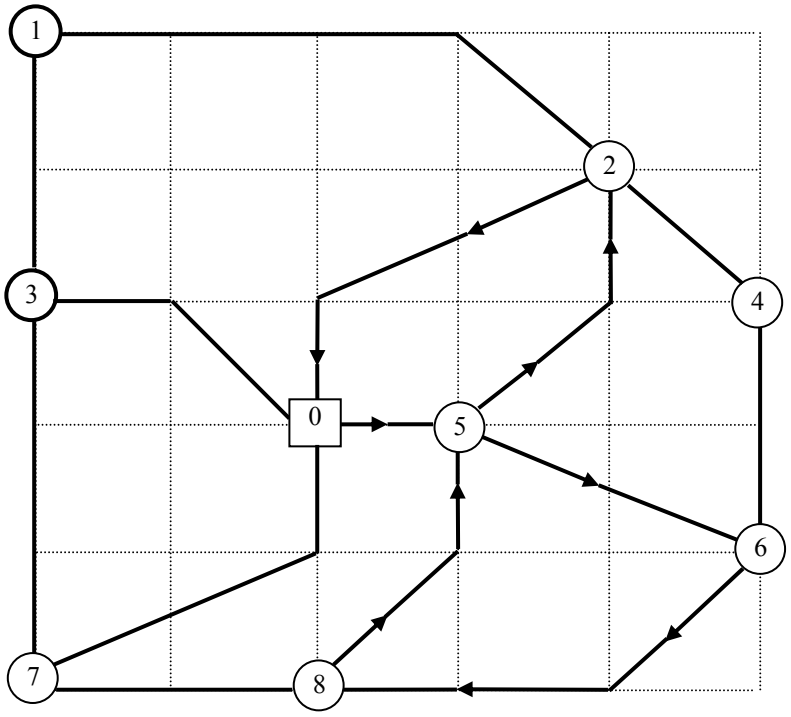
Deoarece lista $\mathcal{L}$ este vidă algoritmul se oprește. Traseul cu cel mai mic cost este reținut în x_{CMB}

6.3.2 Aplicație la problema firmei de curierat rapid

Se consideră rețeaua stradală din figura 6.1 Unele muchii pot fi circulate în ambele sensuri, altele numai în sensul indicat. Un server, localizat în punctul 0 are de transportat cinci comenzi ale căror surse şi destinații sunt date în tabelul 6.2 Se pune problema determinării traseului de lungime minimă care începe şi sfârşeşte în nodul 0 şi pe care serverul trebuie să-l parcurgă în vederea transportării celor cinci comenzi de la surse la destinații.

Tabelul 6

Comanda	Sursa	Destinația
r_1	1	6
r_2	2	7
r_3	6	3
r_4	4	1
r_5	8	2

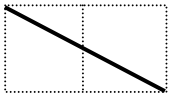


Latura patratului caroiajului măsoară 1 km = 10 hm

diagonala

măsoară 1,5 km =15 hm

diagonala



măsoară 2,3 km =23 hm

Figura 6.1

Tabelul 6.3

	0	1	2	3	4	5	6	7	8
0	*	**45** 0,3,1	**35** 0,5,2	**25** 0,3	**50** 0,5,2,4	**10** 0,5	**33** 0,5,6	**33** 0,7	**53** 0,7,8
1	**45** 1,3,0	*	**45** 1,2	**20** 1,3	**60** 1,2,4	**55** 1,3,0,5	**78** 1,3,0,5,6	**50** 1,3,7	**70** 1,7,8
2	**33** 2,0	**45** 2,1	*	**58** 2,0,3	**15** 2,4	**43** 2,0,5	**35** 2,4,6	**66** 2,0,7	**70** 2,4,6,8
3	**25** 3,0	**20** 3,1	**60** 3,0,5,2	*	**75** 3,0,5,2,4	**35** 3,0,5	**58** 3,0,5,6	**30** 3,7	**50** 3,7,8
4	**48** 4,2,0	**60** 4,2,1	**15** 4,2	**73** 4,2,0,3	*	**58** 4,2,0,5	**20** 4,6	**75** 4,6,8,7	**55** 4,6,8
5	**58** 5,2,0	**70** 5,2,1	**25** 5,2	**83** 5,2,0,3	**40** 5,2,4	*	**23** 5,6	**78** 5,6,8,7	**58** 5,6,8
6	**68** 6,4,2,0	**80** 6,4,2,1	**35** 6,4,2	**85** 6,8,7,3	**20** 6,4	**60** 6,8,5	*	**55** 6,8,7	**35** 6,8
7	**33** 7,0	**50** 7,3,1	**68** 7,0,5,2	**30** 7,3	**83** 7,0,5,2,4	**43** 7,0,5	**66** 7,0,5,6	*	**20** 7,8
8	**53** 8,7,0	**70** 8,7,3,1	**50** 8,5,2	**50** 8,7,3	**65** 8,5,2,4	**25** 8,5	**48** 8,5,6	**20** 8,7	*

- Începem prin a determina – cu o procedură specializată cum ar fi algoritmul lui FLOYD – drumurile de lungime minimă dintre nodurile rețelei. Vezi tabelul 6.3 unde, în celula (*i*, *j*), apare atât lungimea drumului minimal de la *i* la *j* cât şi nodurile prin care trece acesta.

De exemplu cel mai scurt drum de la nodul 3 la nodul 4 are lungimea de 75 hm = 7,5 km şi trece prin nodurile 3, 0, 5, 2 şi 4. Datorită faptului că unele muchii au sens unic de parcurgere, drumul „invers" cel mai scurt de la nodul 4 la nodul 3 are lungimea 73 hm = 7,3 km şi trece prin nodurile 4,2,0 şi 3.

- Construim apoi o problemă a comisvoiajorului, asimetrică, echivalentă cu problema dată. „Oraşele" se identifică cu cele cinci comenzi r_1 , ..., r_5 la care se adaugă „comanda fictivă" r_0 concentrată în locul de plecare şi întoarcere a serverului. „Costurile de deplasare" sunt lungimile drumurilor „în gol" de la destinația unei comenzi la sursa alteia – vezi tabelul 6.4

Tabelul 6.4

	r_0	r_1	r_2	r_3	r_4	r_5
r_0	∞	45	35	33	50	53
r_1	68	∞	35	0	20	35
r_2	33	50	∞	66	83	20
r_3	25	20	60	∞	75	50
r_4	45	0	45	78	∞	70
r_5	33	45	0	35	15	∞

Exemplu de calcul pentru $c(r_4, r_3) = 78$:

- destinația comenzii r_4 este nodul 1;
- sursa comenzii r_3 este nodul 6;
- drumul cel mai scurt de la nodul 1 la nodul 6 este $1 \to 3 \to 0 \to 5 \to 6$ cu lungimea 78 hm = 7,8 km.

- Cu algoritmul lui EASTMAN s-a găsit succesiunea optimă:

$$r_0 \to r_2 \to r_5 \to r_4 \to r_1 \to r_3 \to r_0$$

cu „costul" total minim 95 hm =9,5 km reprezentând lungimea totală a drumurilor „în gol".

Aceasta înseamnă că serverul va pleca din 0 şi va transporta comenzile în ordinea r_2, r_5, r_4, r_1, r_3 după care se va întoarce de unde a plecat.

În notația (2) din 6.2 structura traseului optim arată astfel:

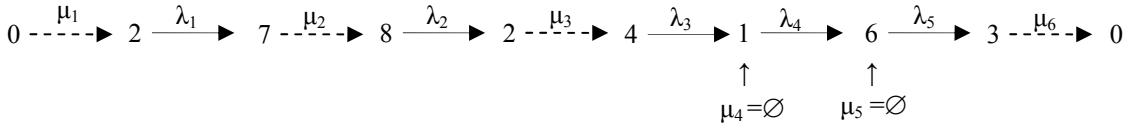


Figura 6.2

sau „în extenso”:

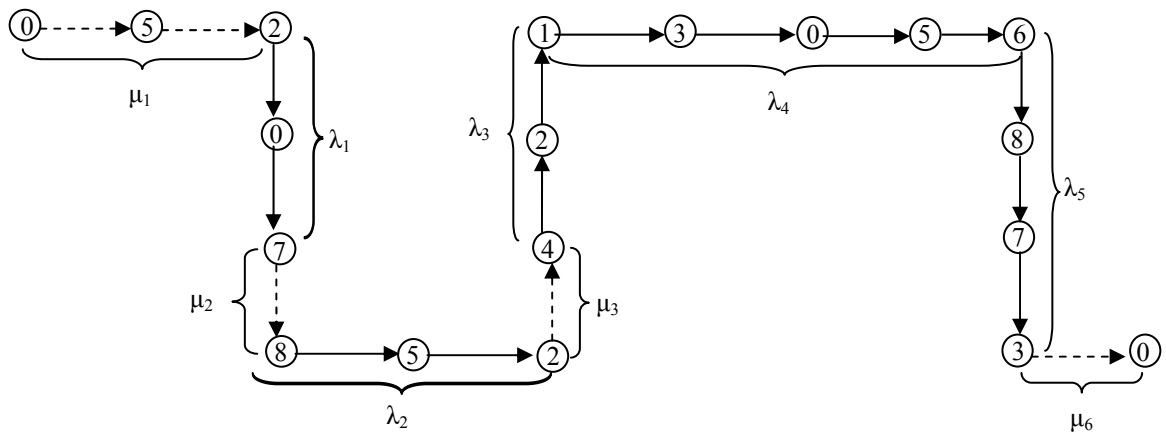


Figura 6.3

Drumurile „în plin” însumează 339 hm = 33,9 km. Întregul traseu optim măsoară:
95 + 339 = 434 hm = 43,4 km. Drumurile în gol reprezintă circa 22% din întregul parcurs.

6.4 Euristici de rezolvare suboptimală a TSP euclidiene

Problema simetrică a comisvoiajorului şi în special cea euclidiană are multe aplicații practice şi ca urmare, cunoaşterea unor metode de rezolvare fie şi suboptimală, este importantă. Deoarece complexitatea problemei face ca metodele exacte să fie aplicabile doar problemelor de dimensiune moderată, în continuare vor fi descrise unele metode euristice.

Metodele de rezolvare suboptimală se împart în două categorii:

a) euristici care construiesc efectiv un traseu complet căutând ca acesta să fie „cât mai bun”. Din această categorie amintim:

- euristica: **mergi la cel mai apropiat vecin**;
- euristica: **inserează punctul cel mai depărtat (apropiat)**;
- euristici bazate pe utilizarea arborelui de cost minim ca de exemplu euristica: **dublează muchiile arborelui de cost minim** sau **euristica lui Cristofides.**

b) euristici care ameliorează valoarea unui traseu complet dat sau construit printr-o procedură din prima categorie.

6.4.1 Euristica: mergi la cel mai apropiat vecin

- se pleacă din localitatea 0 către **cea mai apropiată** localitate;
- din ultima localitate **vizitată** se pleacă către **cea mai apropiată** localitate **nevizitată**; în caz că nu mai există localități nevizitate se revine la punctul de plecare.

Deşi **local** metoda face **cea mai bună** alegere, deseori în traseul construit apar deplasări „costisitoare" astfel că acesta nu este întotdeauna traseul optim! Euristica este simplă şi, dacă este urmată de o procedură ameliorativă poate conduce la rezultate foarte bune în sensul obținerii unui traseu foarte apropiat ca valoare de traseul optim. Deoarece un punct poate avea mai mulți vecini „la fel de apropiați" este posibil ca euristica să conducă la mai multe trasee!

Exemplul 1 În figura 6.4 este dată o concretizare a problemei din debutul secțiunii 6.1 Elicopterul are baza de plecare şi întoarcere în punctul 0 iar deplasările între platforme se fac în linie dreaptă. Latura patratului caroiajului măsoară 10 km; distanțele dintre platforme au fost calculate cu teorema lui Pitagora şi sunt date în tabelul 6.5

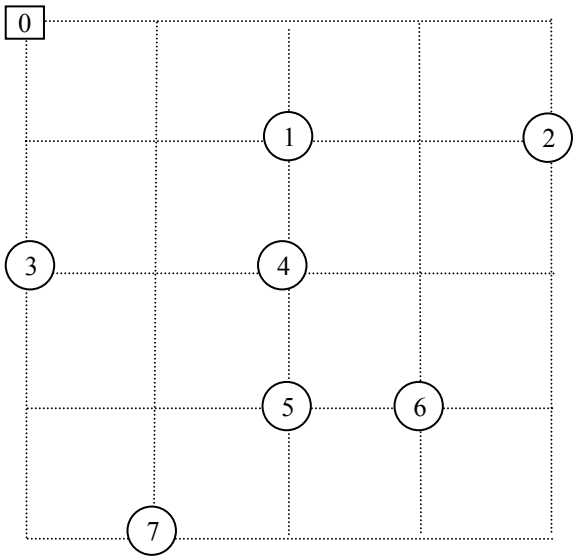


Figura 6.4

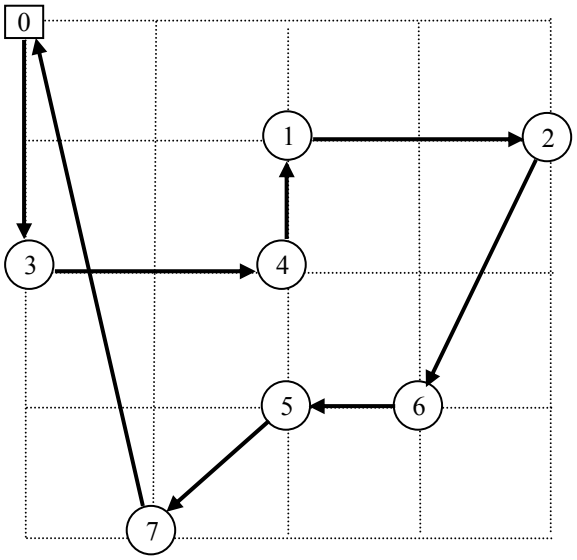
Tabelul 6.5

	1	2	3	4	5	6	7
0	22	41	20	28	36	42	41
1	*	20	22	10	20	22	32
2		*	41	22	28	22	42
3			*	20	22	32	22
4				*	10	14	22
5					*	10	14
6						*	22

Deplasându-ne permanent către cel mai apropiat vecin nevizitat, a rezultat traseul:

$$0 \to 3 \to 4 \to 1 \to 2 \to 6 \to 5 \to 7 \to 0$$

cu lungimea de 157 km, vizualizat în figura 6.5



Cu siguranță traseul construit nu este cel mai scurt din cauza „încrucișării" arcelor (3,4) şi (7,0). Un traseu mai scurt se va obține folosind euristica ameliorativă descrisă în următoarea secțiune.

Cititorul atent a observat că din nodul 4 se putea merge fie în nodul 1 fie în nodul 5; a fost ales nodul1.Îl invităm să continue aplicarea euristicii mergând de astă dată din 4 în 5.
La acest stadiu al discuției, puteți afirma că „înaintarea" din nodul 4 spre nodul 5 va fi mai „profitabilă" decât cea aleasă?...

Figura 6.5

6.4.2 Euristica: ajustare locală

Este vorba de o euristică din cea de a doua categorie care îmbunătățeşte un traseu complet dat T.

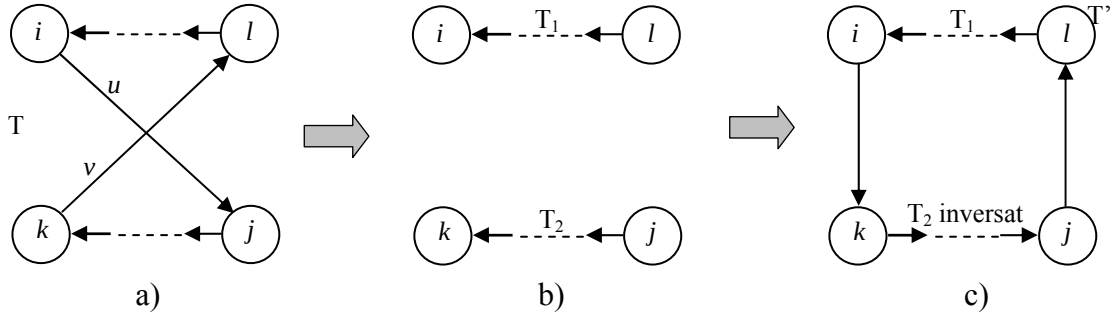


Figura 6.6

- Se examinează **pe rând** toate perechile de arce **neconsecutive** *u*,*v* din T – vezi figura 6.6a).
- Eliminarea arcelor *u* şi *v* „sparge" T în două drumuri disjuncte T_1 şi T_2 – vezi figura 6.6b).
- Aceste drumuri se pot asambla într-un nou traseu T' ≠ T într-un singur mod, indicat în figura 6.6c).

- Dacă:

$$c_{ij} + c_{kl} \leq c_{ik} + c_{jl}$$

noul traseu nu este mai bun ca valoare decât T şi dacă este aşa se trece la examinarea altei perechi de arce neconsecutive din T.

Dacă însă:

$$c_{ij} + c_{kl} > c_{ik} + c_{jl}$$

noul traseu T' are o valoare mai mică decât T. În acest caz, se înlocuieşte T cu T' şi se trece la examinarea perechilor de arce neconsecutive din T'.

Exemplul 2 Aplicăm procedura descrisă traseului din figura 6.5 luând în considerare perechea de arce (3,4) şi (7,0).

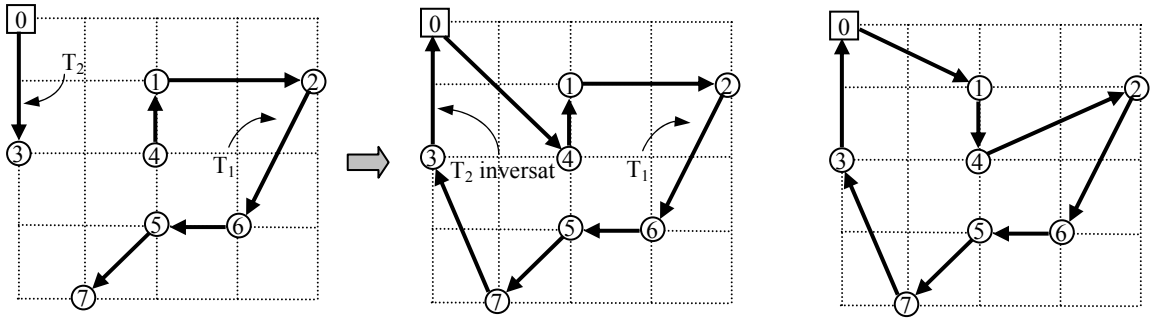


Figura 6.7

Eliminarea arcelor (3,4) şi (7,0) – vezi figura 6.7a) – şi înlocuirea lor cu arcele (0,4) şi (7,3) scurtează distanța totală de parcurs deoarece:

$$c_{34} + c_{70} = 20 + 41 = 61 > 50 = 28 + 22 = c_{04} + c_{73}$$

Noul traseu, indicat în figura 6.7 b) are lungimea $157 - (61 - 50) = 146$ km.

Atenție: pot exista şi perechi de arce care nu se încrucişează dar care conduc la micşorarea valorii unui traseu! (de altfel, „încrucişarea" se poate depista numai dacă traseul este vizualizat...)

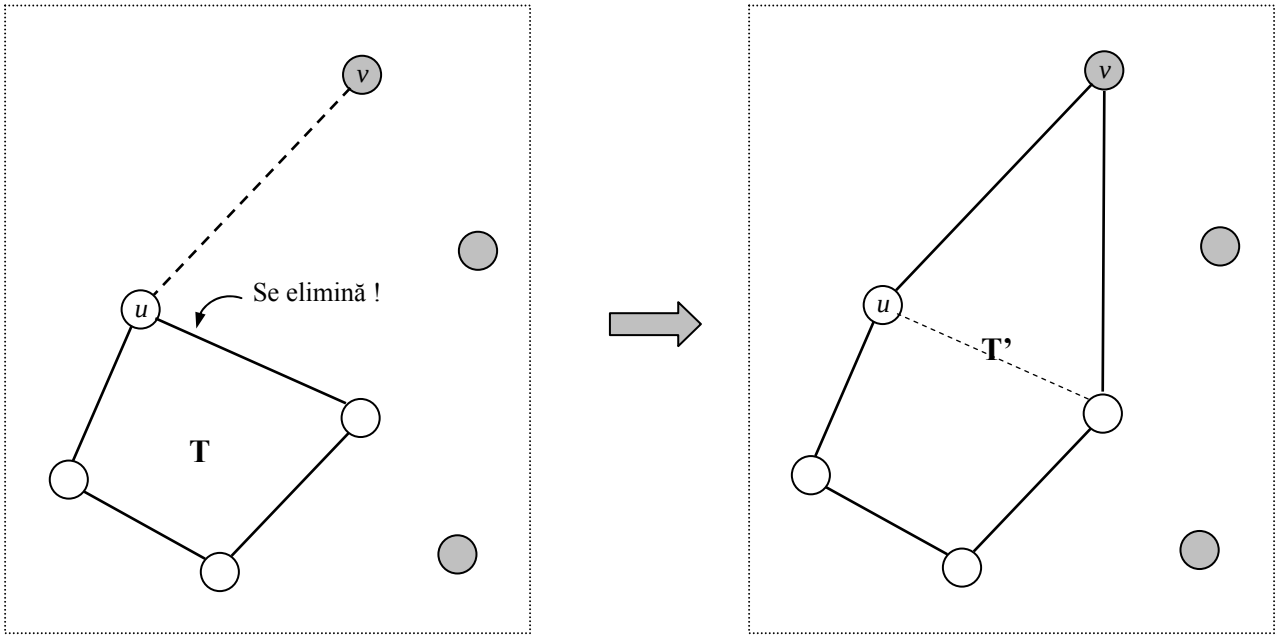
De exemplu, să luăm perechea de arce (0,4) şi (1,2) ale traseului din figura 6.7b). Deoarece:

$$c_{04} + c_{12} = 28 + 20 = 48 > 44 = 22 + 22 = c_{01} + c_{42}$$

prin eliminarea lor se obține traseul din figura 6.7c) cu lungimea $146 - (48 - 44) = 142$ km. Ajustarea locală continuă cu examinarea altor perechi de arce neconsecutive din ultimul traseu obținut.

6.4.3 Euristica: inserează punctul cel mai depărtat

Se consideră un subtraseu T care trece prin unele dintre nodurile 0,1,...,*n*. Printre nodurile nesituate pe traseul T se caută nodul *v*, „cel mai depărtat" de T (reamintim că „distanța" de la *v* la mulțimea nodurilor subtraseuluiT se defineşte ca fiind **minimul** distanțelor de la *v* la fiecare nod din T). Fie *u* nodul din T „cel mai apropiat" de *v* – vezi figura 6.8a). Se elimină una din muchiile traseului care are o extremitate în *u* şi se conectează extremitățile muchiei eliminate cu nodul *v*. Rezultatul este un nou subtraseu T' cu un nod „mai mare" decât subtraseul T – vezi figura 6.8b).



a) b)

Figura 6.8

De regulă, la start subtraseul T este redus la punctul 0 de plecare şi întoarcere al comisvoiajorului astfel că în *n* paşi rezultă un traseu care trece prin toate punctele de vizitat.
La prima vedere, s-ar părea că inserarea la fiecare iterație a „celui mai apropiat" nod nevizitat în locul celui mai depărtat ar fi o idee mai bună. Experimentele numerice arată „în medie" contrariul!!

Exemplul 3 Reluăm problema vizitării platformelor marine din exemplul 1.

- Inițializăm T = {0}.
- Cele mai depărtate puncte de 0 sunt 2 şi 7 pentru că $c_{02} = c_{07} = 41$ sunt cele mai mari distanțe din tabelul 4.Actualizăm T: $0 \to 2 \to 7 \to 0$.
- Distanțele de la nodurile 1, 3, 4, 5şi 6 la mulțimea {0, 2, 7}a nodurilor subtraseului curent T sunt 20, 20, 22, 14 respectiv 22. Alegând nodul 6, T devine: $0 \to 2 \to 6 \to 7 \to 0$.
- Față de noul T cele mai depărtate noduri sunt 1 şi 3.A fost ales şi inserat nodul 3 astfel că T: $0 \to 2 \to 6 \to 7 \to 3 \to 0$.

- Nodurile rămase au fost inserate în ordinea 1, urmat de 4 şi la urmă 5. În final a rezultat traseul complet: T: $0 \to 1 \to 2 \to 6 \to 5 \to 4 \to 7 \to 3 \to 0$ cu lungimea de 148 km.

Observație: În mai multe rânduri a fost necesar să facem o alegere a nodului ce urma a fi inserat în subtraseul curent dintr-o mulțime de noduri candidate. Ca urmare procedura poate conduce şi la alte trasee unele posibil mai bune, altele posibil mai proaste decât traseul efectiv construit!

6.4.4 Euristica: dublează muchiile unui arbore minimal

- În graful complet cu nodurile 0,1,...,*n* se determină un **arbore** H de valoare(lungime) minimă. Aceasta este o problemă de optimizare **uşoară** rezolvabilă cu algoritmul lui Kruskal – vezi unitatea de învățare 5.

- În continuare fiecare muchie din H se înlocuieşte cu două muchii de aceeaşi lungime. Se obține un multigraf H'.

Prin construcție, H' este un **graf eulerian** (deoarece gradele tuturor nodurilor sunt pare!) şi ca urmare există posibilitatea traversării **tuturor** muchiilor, **o singură dată,** cu plecarea şi întoarcerea în nodul 0.

- Fie

$$0 \to x \to y \to \cdots\cdots \to u \to v \to 0 \qquad (*)$$

Succesiunea în care toate muchiile multigrafului H' au fost parcurse. În aceeastă secvență este posibil ca unul sau mai multe oraşe să apară de mai multe ori. Procedăm la următoarea operație de **scurtcircuitare**:

În succesiunea (*) se determină **prima** tripletă $i \to j \to k$ de noduri (oraşe) consecutive în care „mijlocul" j mai apare ulterior în succesiune. Înlocuim secvența $i \to j \to k$ cu arcul direct $i \to k$. Prin această operație:

- fiecare oraş continuă să fie vizitat măcar o dată;
- noul traseu are o lungime mai mică deoarece $c_{ij} + c_{jk} \geq c_{ik}$ (atenție, avem în vedere în exclusivitate TSP euclidiene!!)

Scurtcircuitarea se repetă până când, în secvența (*) **actualizată,** fiecare oraş apare **o singură dată**, bineînțeles cu excepția punctului de plecare şi întoarcere 0.

Se poate demonstra că lungimea traseului obținut prin această euristică este cel mult de două ori mai mare decât lungimea traseului optim (în practică lucrurile stau însă mai bine...)

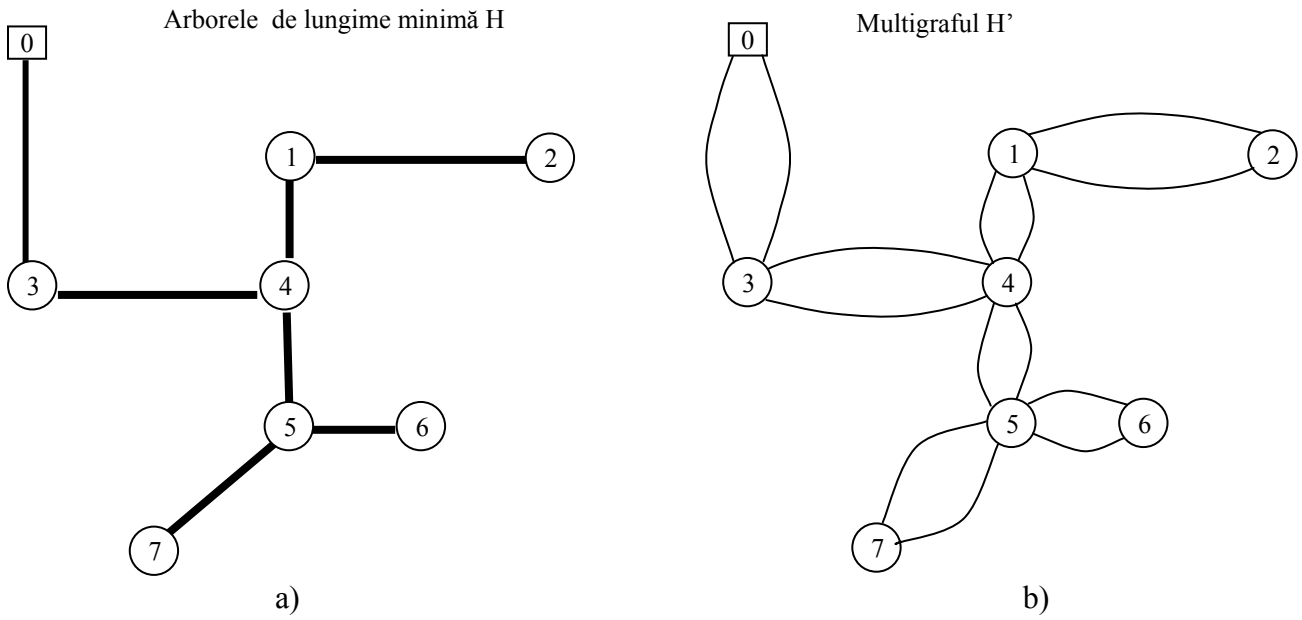


Figura 6.9

Exemplul 4 Aplicăm euristica descrisă problemei din exemplul 1. În figura 6.9a) este vizualizat un arbore H de valoare minimă; în figura 6.9b) apare multigraful H'. Traversarea tuturor muchiilor din H', fiecare o singură dată se poate realiza în mai multe feluri; iată unul din ele:

$$0 \to 3 \to 4 \to 5 \to 6 \to 5 \to 7 \to 5 \to 4 \to 1 \to 2 \to 1 \to 4 \to 3 \to 0$$

Traseul obținut nu este o soluție a TSP deoarece unele puncte sunt vizitate de mai multe ori. Eliminarea repetițiilor prin „scurtcircuitare" este arătată în diagrama din figura 6.10

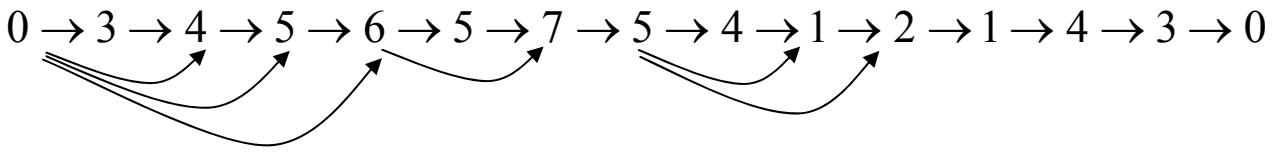


Figura 6.10

A rezultat traseul: $0 \to 6 \to 7 \to 5 \to 2 \to 1 \to 4 \to 2 \to 0$ cu lungimea de 176 km şi vizualizat în figura 6.11a). După aplicarea euristicii de ajustare locală – vezi figurile 6.11 b),c),d) e) se regăseşte traseul din figura 6.7c) care „pare a fi" traseul cel mai scurt. Perechile de arce neconsecutive înlocuite au fost evidențiate prin linii punctate.

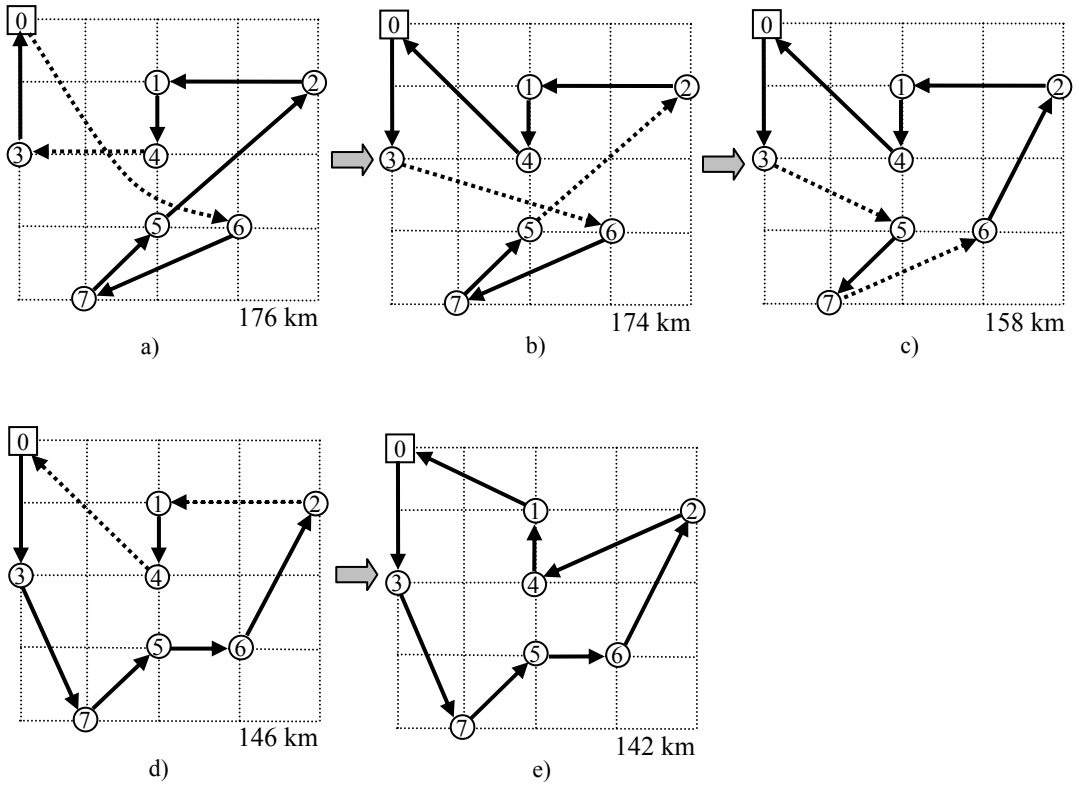


Figura 6.11

6.4.5 Euristica lui Cristofides

Această procedură, datorată lui CRISTOFIDES este o **rafinare** a euristicii precedente.**Deoarece suma gradelor nodurilor din arborele H este pară, numărul nodurilor de grad impar este par!** Folosind numai aceste noduri şi muchiile dintre ele se determină **cuplajul** C de **pondere minimă (**vezi unitatea de învățare 4!), ponderile muchiilor luate în calcul fiind distanțele dintre extremități. Fie H' (multi)graful rezultat din H prin adăugarea muchiilor cuplajului C, cu mențiunea că dacă între două noduri avem o muchie în H şi alta în C, graful H' va avea, între nodurile respective, două muchii! Prin construcție, în H' fiecare nod are grad **par**, deci H' este un **graf eulerian** şi ca urmare există posibilitatea traversării tuturor muchiilor sale exact o singură dată.Plecând de aici şi folosind **tehnica scurtcircuitării** se ajunge la un traseu complet.

Se poate arăta că lungimea acestui traseu nu depăşeşte 3/2 din lungimea traseului optim.

Exemplul 5 Pentru problema euclidiană a comisvoiajorului din exemplul 1, un arbore de lungime minimă este disponibil în figura 6.9a). Nodurile de grad impar sunt 0,2,6,7 cu gradul 1 şi 4,5 cu gradul 3.Prin simplă inspecție (sau utilizând un algoritm de determinare a cuplajului de pondere minimă în subgraful complet ale cărui noduri sunt 0,2,4,5,6,7) se constată că muchiile {0,2}, {4,6}, {5,7}au cea mai mică pondere totală: $c_{02} + c_{46} + c_{57} = 69$ .Adăugând aceste muchii la arborele H se obține multigraful H' din figura 6.12

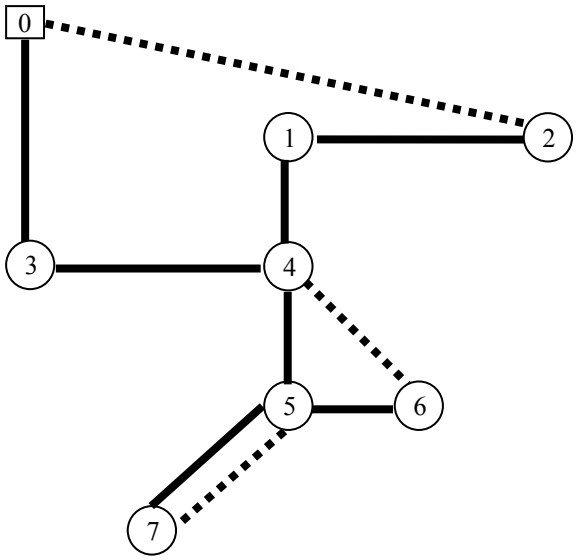


Figura 6.12

O posibilă traversare a tuturor muchiilor o singură dată este dată de succesiunea $0 \to 2 \to 1 \to 4 \to 6 \to 5 \to 7 \to 5 \to 4 \to 3 \to 0$. După scurtcircuitarea secvențelor $1 \to 4 \to 6$ şi $6 \to 5 \to 7$ resultă traseul $0 \to 2 \to 1 \to 6 \to 7 \to 5 \to 4 \to 3 \to 0$ cu lungimea de 169 km. Invităm cititorul să-l vizualizeze şi să cerceteze dacă poate fi scurtat folosind ajustarea locală.

Probleme propuse

1. Un comisvoiajor situat în localitatea 0 are de vizitat oraşele 1, 2, 3 şi 4.Costurile de deplasare c_{ij} , $0 \le i, j \le 4$, $i \ne j$ de la un oraş la altul sunt date în tabelul 6.6. Se observă că în general $c_{ij} \ne c_{ji}$ altfel spus costul deplasării între două localități depinde de sensul de deplasare – avem de a face cu o TSP asimetrică. Să se aplice algoritmul lui Eastman pentru a găsi un traseu de cost minim.

Tabelul 6.6

	0	1	2	3	4
0	□	10	7	9	11
1	8	□	11	12	4
2	9	11	□	13	6
3	6	12	14	□	7
4	10	5	8	8	□

2. Să se aplice algoritmul lui Eastman pentru rezolvarea problemei asimetrice a comisvoiajorului ale cărei costuri sunt date în tabelul 6.7

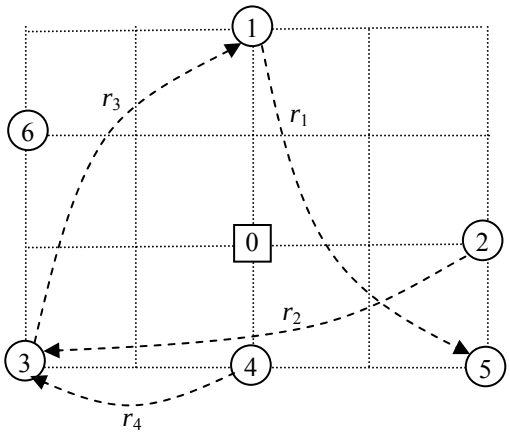
3. Problemele 1 şi 2 sunt „mici" şi cu un program de calculator destul de simplu ele pot fi rezolvate prin enumerarea completă a soluțiilor. Câte trasee complete ar trebui generate în fiecare caz?

Tabelul 6.7

	0	1	2	3	4	5
0	□	18	13	20	22	34
1	10	□	20	17	18	10
2	10	9	□	20	11	20
3	19	27	23	□	6	11
4	24	17	20	7	□	8
5	23	33	18	14	30	□

4. Algoritmul lui Eastman este o procedură de tip B&B. În ce situații nu mai are loc ramificarea unui nod?

5. Un server are de transportat patru colete mari dintr-un loc în altul. În figura 6.13 sunt indicate pozițiile locului de parcare al serverului şi ale surselor şi destinațiilor coletelor. Deplasările se fac numai pe liniile orizontale şi verticale ale caroiajului. Latura patratului caroiajului se va lua ca unitate de lungime.



Comanda	Sursa	Destinația
r_1	1	5
r_2	2	3
r_3	3	1
r_4	4	3

Figura 6.13

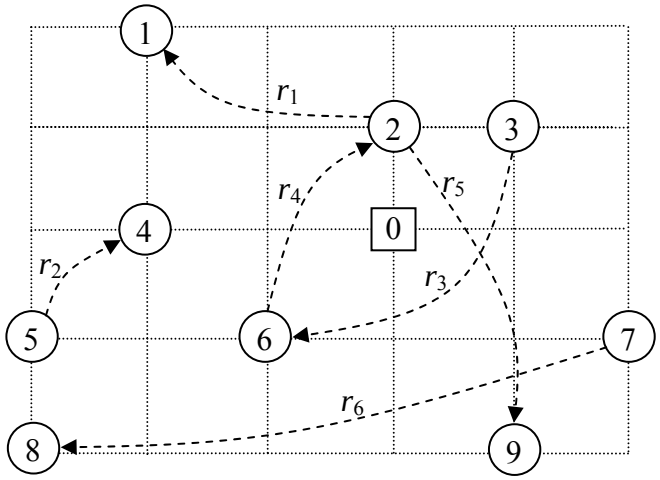
a) În ce ordine vor fi transportate comenzile pentru ca distanța totală parcursă de server să fie minimă, ştiind că acesta pleacă şi se întoarce în 0 şi nu transportă mai mult de o comandă o dată. Precizați distanța minimă de parcurs şi procentul parcursului „în gol" (se va construi o TSP asimetrică echivalentă care va fi rezolvată cu algoritmul lui Eastman).
b) Pe lângă comenzile indicate, serverul trebuie să treacă şi prin punctul 6, la un service, pentru remedierea unei mici defecțiuni. Cum va arăta traseul minim? (poanta: transformați „oprirea în punctul 6" într-o nouă comandă r_5 .

c) Elaborați (empiric) un program de transport în ipoteza că serverul poate duce şi două colete o dată!

6. În rețeaua din figura 6.14 nodurile 1, 2,...,9 sunt sursele şi destinațiile comenzilor de transport r_1 ,r_2,...,r_6. Un server cu locul de parcare în nodul 0 este desemnat să transporte aceste comenzi. Deplasările serverului se fac numai pe liniile orizontale şi verticale ale caroiajului. Latura patratului caroiajului măsoară 1 km.

În ce ordine vor fi efectuate transporturile pentru ca distanța totală parcursă de server să fie minimă în ipoteza că serverul nu poate transporta mai mult de o comandă odată.

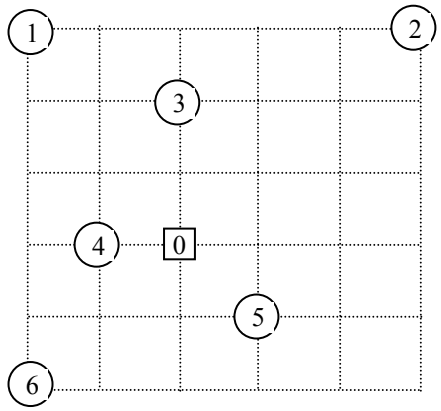
Elaborați în manieră empirică un program de lucru ştiind că serverul poate duce două comenzi odată.



Comanda	Sursa	Destinația
r_1	2	1
r_2	5	4
r_3	3	6
r_4	6	2
r_5	2	9
r_6	7	8

Figura 6.14

6. Agenția de turism HAI SĂ HAIDEM situată în localitatea 0 este interesată în determinarea unui traseu de vizitare a şase obiective de interes turistic şi istoric notate 1,2,...,6. Pozițiile relative ale celor şapte puncte sunt indicate în figura 6.15 iar distanțele dintre ele – măsurate în km în linie dreaptă – sunt date în tabelul alăturat.

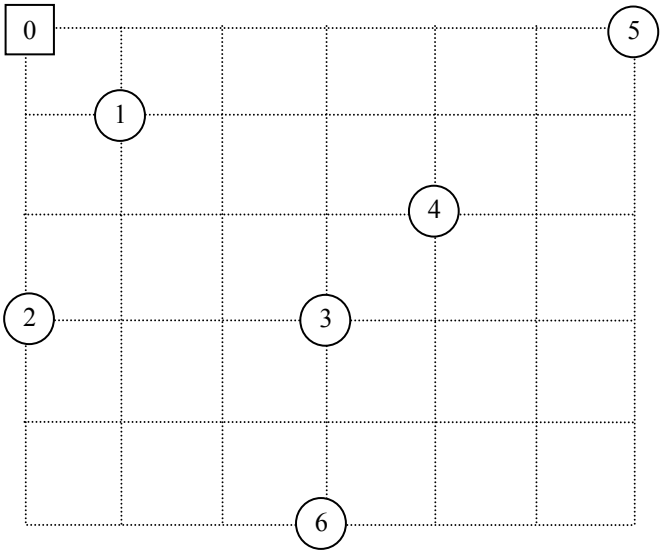


	1	2	3	4	5	6
0	36	42	20	10	14	28
1	*	50	22	32	50	50
2		*	32	50	45	71
3			*	22	32	45
4				*	22	22
5					*	32

Figura 6.15

i) Evident, obiectivul urmărit este determinarea celui mai scurt traseu. Câte soluții ≡ trasee complete ar fi de examinat în situația dată?
ii) Determinați un traseu cât mai scurt aplicând **riguros** euristica „mergi la cel mai apropiat vecin”;
iii) Dacă este cazul folosiți ajustarea locală pentru micşorarea parcursului total;
iv) Reluați chestiunea aplicând (riguros!) una sau alta din euristicile explicate în text;
v) Care este cel mai scurt traseu pe care l-ați găsit?

7. Se consideră problema (euclidiană) a comisvoiajorului cu datele din figura 6.16



	1	2	3	4	5	6
0	14	30	42	45	50	58
1	*	22	28	32	41	45
2		*	30	41	58	36
3			*	14	36	20
4				*	22	32
5					*	54

Figura 6.16

i) Aplicați riguros euristica „mergi la cel mai apropiat vecin”. Eliminați eventualele încrucişări cu euristica de ajustare locală;
ii) Aplicați euristicile „inserează puncul cel mai apropiat” respectiv „cel mai depărtat” – urmate eventual de „ajustarea locală” şi comparați rezultatele;
iii) Aplicați cele două euristici bazate pe arborele de lungime minimă;
iv) Care este cel mai scurt traseu găsit?

Unitatea de învățare 7

INTRODUCERE ÎN OPTIMIZAREA COMBINATORIALĂ
Problema stabilirii traseelor (Vehicle Routing Problem ≡ VRP)

Cuprins

7.1 Formularea problemei. Aplicații
7.2 Metode de rezolvare suboptimală a VRP
7.2.1 Euristica Cluster First - Route Second
7.2.2 Euristica Route First - Cluster Second
7.2.3 Euristica Clarke – Wright
7.3 Ilustrări numerice
7.3.1 Aplicarea euristicii Cluster First - Route Second
7.3.2 Aplicarea euristicii Route First - Cluster Second
7.3.3 Aplicarea euristicii Clarke – Wright

Probleme propuse

Obiectivul unității de învățare 7

În activitatea de aprovizionare desfacere, alegerea judicioasă a traseelor de deplasare joacă un rol important în reducerea cheltuielilor de transport şi **problema stabilirii traseelor** (**V**ehicle **R**outing **P**roblem, abreviat **VRP**) formalizează acest aspect des întâlnit în practică.

Obiectivul unității de învățare 7 este **prezentarea unor metode suboptimale de soluționare** a VRP, prezentare însoțită de ilustrări numerice adecvate.

7.1 Formularea problemei. Aplicații

O firmă are un număr de clienți cărora trebuie să le livreze mărfuri în cantități prestabilite. Livrările se fac de la un depozit cu ajutorul unor mijloace de transport (proprii sau închiriate) de diferite capacități. Orice comandă trebuie onorată la un singur transport. Vehiculele pleacă de la depozit,încărcate sau nu la capacitatea maximă, vizitează un număr de clienți şi, după ce se golesc, se întorc la depozit pentru o eventuală reîncărcare.

Se pune problema stabilirii traseelor de vizitare a clienților, a tipului şi a numărului de vehicule necesare satisfacerii tuturor comenzilor astfel încât distanța totală parcursă să fie minimă.

Enunțul de mai sus este cunoscut în literatura de specialitate ca **problema stabilirii traseelor** (**V**ehicle **R**outing **P**roblem, abreviat **VRP**)

VRP generalizează problema comisvoiajorului (TSP). Într-adevăr, dacă ar exista un singur vehicul cu o capacitate care să acopere suma tuturor cererilor, VRP s-ar reduce la stabilirea ordinii de vizitare a clienților astfel încât distanța totală de parcurs să fie minimă – vezi figura 7.1a).

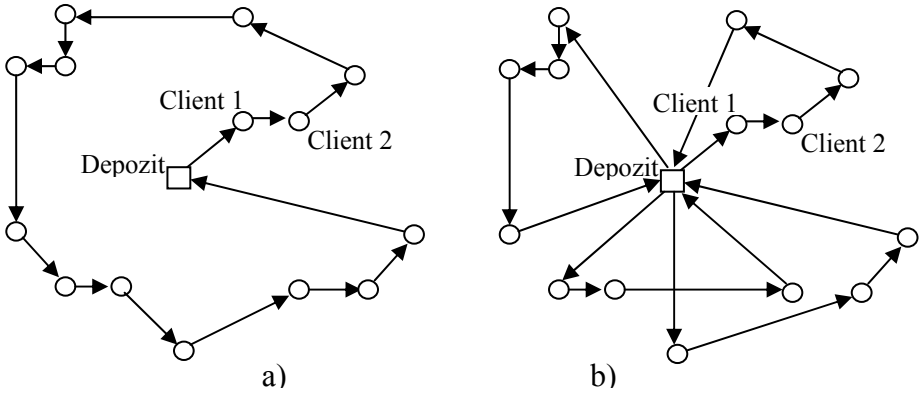


Figura 7.1

De obicei, vehiculele au capacități inferioare cererii totale astfel că vor fi necesare mai multe trasee, parcurse fie succesiv de acelaşi vehicul, fie simultan de mai multe – vezi figura 7.1b).

În consecință, **VRP este o problemă de optimizare combinatorială grea.** Chestiunea devine şi mai complicată dacă – aşa cum se întâmplă în realitate – livrările trebuie efectuate în anumite **ferestre de timp**, convenite în prealabil cu clienții.

Importanța VRP în activitatea de transport este covârşitoare.Transportul unor bunuri (produse de panificație, mărfuri alimentare, băuturi etc) de la producător la centrele de desfacere sau colectarea laptelui de la diverşi crescători de animale la o fabrică de industrializare sunt numai câteva aplicații ale VRP.

7.2 Metode de rezolvare suboptimală a VRP

Ca şi în cazul TSP, metodele de rezolvare exactă a VRP sunt în general neeficiente din cauza volumului de calcule şi a timpului necesar, care sunt imense chiar şi pentru aplicațiile de talie moderată.

Din acest motiv sunt preferate metodele **euristice** care, cu un efort computațional rezonabil, conduc la **soluții suboptimale acceptabile**.

7.2.1 Euristica Cluster First - Route Second

Etapa I Mulțimea clienților se partiționează într-un număr cât mai mic (de dorit minim...) de grupe (clustere) cu proprietatea că toate comenzile dintr-o grupă pot fi transportate cu un singur vehicul. Se urmăreşte totodată încărcarea cât mai bună a vehiculelor utilizate.

Etapa II Pentru fiecare grupă de clienți se stabileşte un traseu de vizitare cu lungimea totală cât mai mică.

Etapa III În măsura în care se poate, unii clienți vor fi mutați dintr-o grupă în alta, dacă această operație duce la micşorarea distanței de parcurs şi nu sunt depăşite capacitățile de transport ale vehiculelor repartizate grupelor.

Observații: 1) Chestiunea care face obiectul primei etape este un caz particular al unei alte importante probleme de optimizare combinatorială „grea" cunoscută sub numele de **Bin Packing Probem**:

Un număr de **obiecte** (repere) cu **ponderi** (volum, greutate) cunoscute trebuie aşezate în **cutii** cu capacități date (de obicei, cutiile au aceeaşi capacitate). Să se determine numărul **minim** de cutii necesar **împachetării** tuturor obiectelor.

În esență, această problemă este identică cu **problema croirii** (Cutting Stock Problem) enunțată în unitatea de învățare 4, secțiunea 4.3

2) În etapa a doua avem de rezolvat o **problemă a comisvoiajorului**.

7.2.2 Euristica Route First - Cluster Second

Etapa I Se determină traseul de vizitare al **tuturor** clienților care începe şi sfârşeşte în depozit şi are lungimea cea mai mică.

Etapa II Clienții sunt împărțiți în grupe astfel încât:

- comenzile clienților dintr-o grupă să încapă într-un singur vehicul care va fi repartizat grupei respective
- să fie respectată ordinea de vizitare de pe trasul definit în prima etapă – vezi figura 7.2

Comentariu: este posibil ca numărul de vehicule necesare transportului comenzilor de la depozit la clienți să fie mai mare decât cel din euristica precedentă. În schimb, distanța totală de parcurs este de regulă mai mică

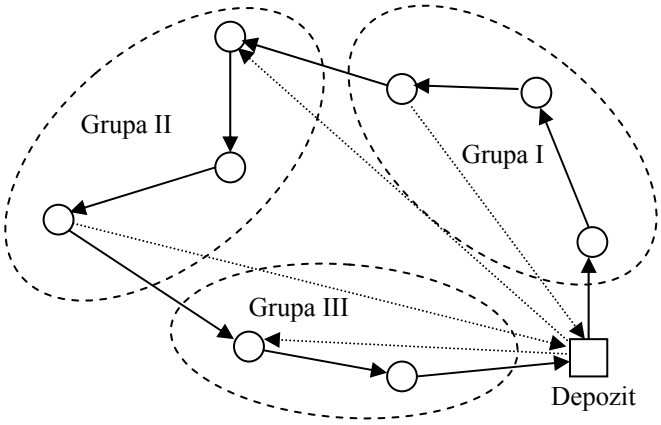


Figura 7.2

7.2.3 Euristica Clarke – Wright (metoda **reducerilor**)

Presupunem în continuare că depozitul este notat cu 0 iar clienții sunt numerotați 1,2,...,*n*. Fie $d_1, d_2, ..., d_n$ distanțele de la depozitul 0 la clienții 1,2,...,*n* şi fie d_{ij} distanța dintre clienții *i* şi *j* , $i \neq j$. Vom presupune că aceste distanțe sunt măsurate în linie dreaptă astfel că ele vor verifica **inegalitatea triunghiului.**

Euristica C –W are la bază ideea de **concatenare** a traseelor în scopul reducerii distanței totale de parcurs.
Considerăm două trasee de vizitare a unora dintre clienții 1,2,...,*n*:

$$T_1 : 0 \to a \to \cdots\cdots \to i \to 0$$
$$T_2 : 0 \to j \to \cdots\cdots \to b \to 0$$

vezi figura 7.3a)

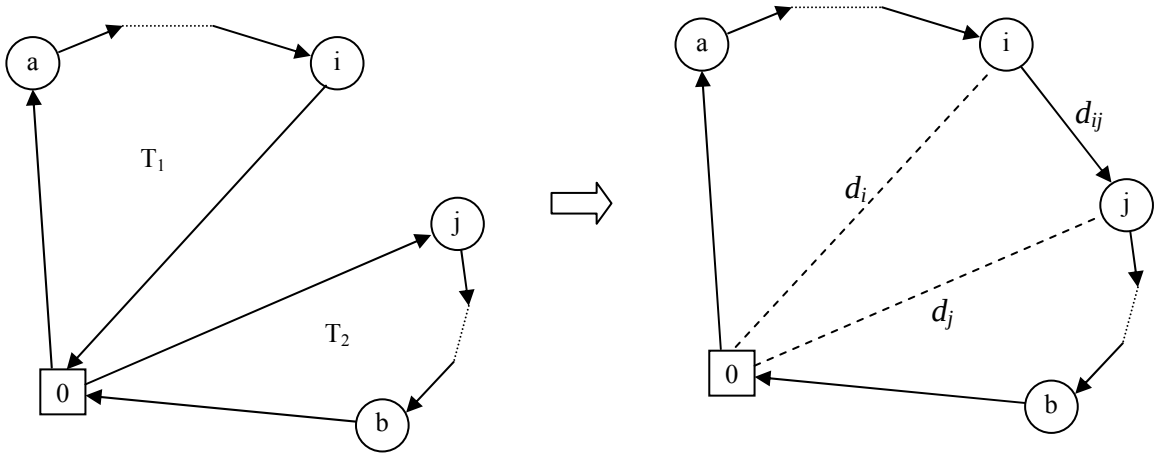
Presupunem că:

- clienții de pe un traseu nu se regăsesc printre clienții de pe celălalt traseu;
- comenzile clienților de pe un traseu pot fi transportate cu un singur vehicul.

Concatenarea traseelor T_1 şi T_2 înseamnă traseul:

$$0 \to a \to \cdots\cdots \to i \to j \to \cdots\cdots \to b \to 0$$

vezi figura 7.3b). După cum se vede rezultatul concatenării depinde de ordinea de parcurgere adoptate în cele două trasee originale.



a) b)

Figura 7.3

Dacă D_1 şi D_2 sunt lungimile traseelor T_1 şi T_2 atunci lungimea traseului rezultat din concatenare este:

$$D = (D_1 - d_i) + d_{ij} + (D_2 - d_j) = D_1 + D_2 - (d_i + d_j - d_{ij}) = D_1 + D_2 - s_{ij}$$

Mărimile:

$$s_{ij} = d_i + d_j - d_{ij}$$

se numesc **reduceri** şi au proprietăţile:

$s_{ij} \geq 0$ - consecinţă a inegalităţii triunghiului;

$s_{ij} = s_{ji}$

Prin urmare:

$$D \leq D_1 + D_2$$

adică prin concatenare, distanţa totală de parcurs se micşorează.

Concatenarea traseelor T_1 şi T_2 se va numi **admisibilă** dacă există un vehicul a cărui capacitate să acopere totalitatea comenzilor transportate separat pe cele două trasee.

Cu aceste pregătiri putem trece la prezentarea euristicii Clarke – Wright.

Start.

- Se calculează reducerile $s_{ij} = d_i + d_j - d_{ij}$ pentru $1 \le i < j \le n$ şi se ordonează **descrescător** într-o **listă a reducerilor**.
- Se iniţializează **lista traseelor** de vizitare a clienţilor cu traseele **directe** prin care fiecare client este servit de la depozit cu un singur vehicul special destinat pentru acest serviciu – vezi figura 7.4;
- Se iniţializează distanţa totală de parcurs D cu valoarea:

$$D = 2(d_1 + d_2 + ... + d_n)$$

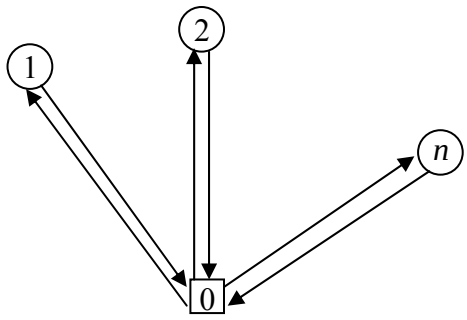


Figura 7.4

Pasul 1 Dacă lista reducerilor este vidă, **stop.**
În caz contrar, se consideră **primul** element s_{ij} din listă; elementul selectat se şterge din listă.

Pasul 2 În lista traseelor curente de vizitare a clienţilor se caută două trasee T_1 şi T_2 cu proprietăţile:

- parcurgând T_1 şi T_2 în sensuri convenabile, **ultimul** client din T_1 este clientul i şi **primul** client din T_2 este clientul j;
- există un vehicul a cărui capacitate acoperă suma comenzilor tuturor clienţilor de pe cele două trasee.

Dacă T_1 şi T_2 nu există, se revine la pasul 1. Altminteri, se trece la:

Pasul 3 În lista traseelor curente înlocuim traseele T_1 şi T_2 cu traseul rezultat din concatenarea acestora. Actualizăm distanţa totală de parcurs:

$$D \leftarrow D - s_{ij}$$

Se revine la pasul 1.

Observaţie: Pentru oprirea algoritmului nu este necesară examinarea tuturor reducerilor! Este evident că procedura este considerată încheiată în momentul în care nu mai sunt posibile concatenări admisibile!

7.3 Ilustrări numerice

O mare companie de desfacere a produselor de larg consum a deschis un lanţ de zece magazine în oraşul X şi împrejurimi. Aprovizionarea magazinelor se face dintr-un depozit central, o dată la trei zile. Comenzile sunt centralizate la depozit; mărfurile cerute sunt sortate şi aranjate în **paleţi**. Astfel, pentru simplitate, se poate admite că cererea unui magazin este exprimată printr-un număr întreg de paleţi. Pentru transportul paleţilor de la depozit la magazine compania dispune de un parc propriu de camioane, fiecare cu capacitatea de 20 de paleţi.

În tabelul 7.1 sunt date cererile (în paleți) ale celor zece magazine pentru următoarea aprovizionare.

Localizarea depozitului, notat cu 0 şi a magazinelor 1,2,...,10 este indicată în figura 7.5 în care latura patratului caroiajului măsoară 10 km. Distanțele dintre puncte sunt măsurate în linie dreaptă şi date în tabelul 7.2 (se precizează că distanțele reale nu diferă semnificativ de aceste valori!)

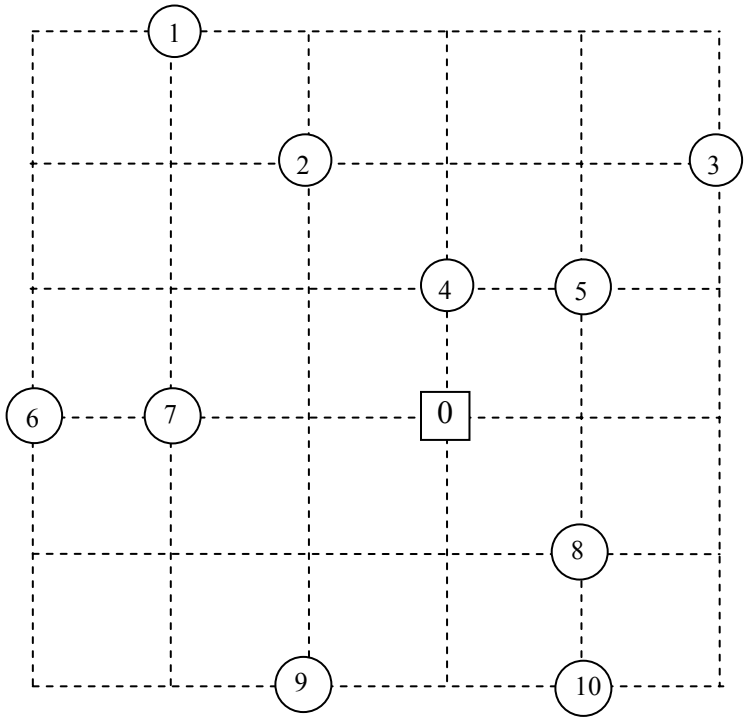


Figura 7.5

Tabelul 7.1

Magazin	1	2	3	4	5	6	7	8	9	10	Total
Cerere (paleți)	8	4	5	4	7	9	7	5	6	3	58

Tabelul 7.2

	1	2	3	4	5	6	7	8	9	10
0	36	23	29	10	15	30	20	15	23	23
1	*	15	42	29	36	32	30	50	51	59
2		*	30	15	23	29	23	36	40	45
3			*	23	15	54	45	32	50	42
4				*	10	32	23	23	32	32
5					*	42	32	20	36	30
6						*	10	42	29	45
7							*	32	23	36
8								*	23	10
9									*	20

Conducerea companiei doreşte stabilirea unor trasee de aprovizionare a căror lungime totală să fie cât mai mică.

Observaţie: 1) Este clar că traseele de aprovizionare depind atât de poziţiile relative ale magazinelor faţă de depozit cât şi de cererile acestora. În consecinţă,la o altă distribuţie a cererilor, traseele se vor schimba...

2) Cererea totală este de 58 de paleţi astfel că vor fi necesare cel puţin $\left\lceil \frac{58}{20} \right\rceil = 3$ vehicule de transport.

7.3.1 Aplicarea euristicii Cluster First - Route Second

Pentru gruparea şi încărcarea cererilor în camioane folosim o euristică de rezolvare suboptimală a problemei „de împachetare" **Bin Packing** cunoscută sub numele de **BFD** ≡ **B**est **F**it **D**ecreasing:

- Mai întâi ordonăm cererile **descrescător**, după volum:

Tabelul 7.3

Magazin	6	1	5	7	9	3	8	2	4	10	Total
Cerere (paleţi)	9	8	7	7	6	5	5	4	4	3	58

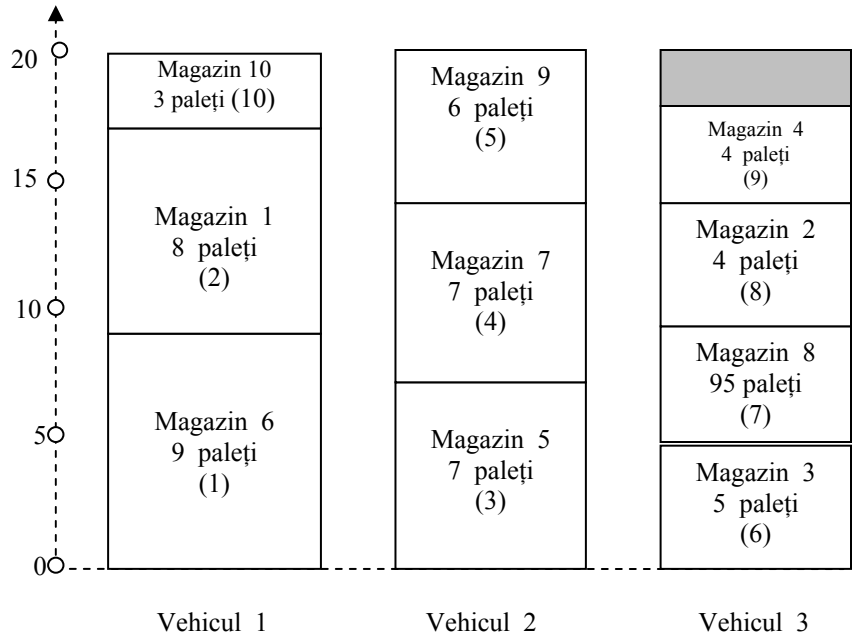
- Presupunem disponibile un număr suficient de camioane – cel puţin trei! – şi le considerăm „trase la rampa depozitului" într-un şir de aşteptare.

- Comenzile vor fi încărcate **pe rând** în ordinea în care apar în tabelul 7.3 În momentul în care i-a sosit rândul, o comandă va fi urcată în primul camion – parţial încărcat – care mai dispune de spaţiu disponibil; dacă nu există spaţiu, comanda va fi pusă în primul camion gol din şirul de aşteptare.

Rezultatul aplicării acestei proceduri este vizualizat schematic în figura 7.6

În acord cu instrucţiunile generale ale euristicii Cluster First Route Second:

Vehiculul 1 va transporta comenzile magazinelor 6,1 şi 10 $\rightarrow$ Traseul T_1;
Vehiculul 2 va transporta comenzile magazinelor 5,7 şi 9 $\rightarrow$ Traseul T_2;
Vehiculul 3 va transporta comenzile magazinelor 3,8,2 şi 4 $\rightarrow$ Traseul T_3.



Atenție: Numerele din paranteze indică **ordinea** încărcării comenzilor în vehicule!

Figura 7.6

Cu o procedură specifică problemei comisvoiajorului, pentru fiecare vehicul în parte vom stabili cele mai scurte trasee de vizitare ale magazinelor repartizate. Problema comisvoiajorului fiind în general extrem de grea, ne vom mulțumi cu determinarea unor trasee „acceptabile" ca lungime folosind pentru aceasta o euristică cât mai performantă. În cazul de față s-a utilizat euristica „mergi la cel mai apropiat vecin"; rezultatele sunt vizualizate în figura 7.7

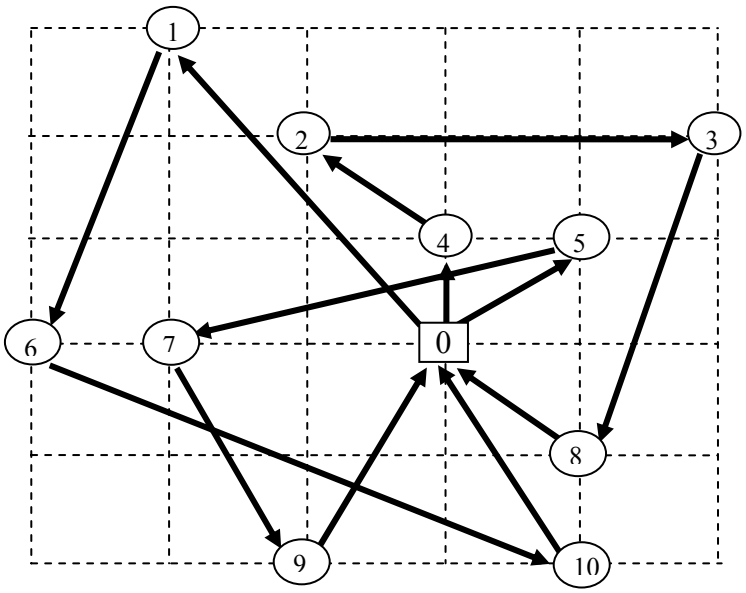


Figura 7.7

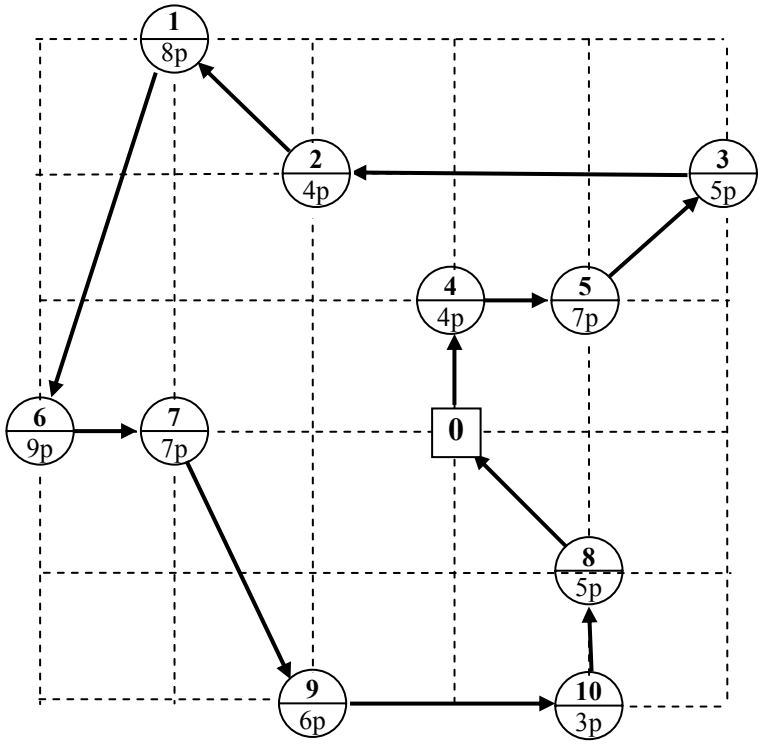
În tabelul 7.4 sunt evidențiate lungimile celor trei trasee:

Tabelul 7.4

Traseu	T_1	T_2	T_3	Total
Lungime (km)	136	93	102	331

7.3.2 Aplicarea euristicii Route First - Cluster Second

Mai întâi, vom determina un traseu de vizitare a tuturor clienților cu plecarea şi întoarcerea în depozitul 0 şi cu lungimea cât mai mică. Cu euristica „mergi la cel mai apropiat vecin" urmată de euristica de ajustare locală s-a obținut traseul din figura 7.8 cu o lungime de 190 km. **Încărcarea comenzilor în camioane se va face în raport cu ordinea de vizitare a celor 10 magazine!** - vezi figura 7.9



Notă: pentru fiecare magazin s-a reamintit cererea de **paleți**!

Figura 7.8

Magazine: $0 \to 4 \to 5 \to 3 \to 2 \to 1 \to 6 \to 7 \to 9 \to 10 \to 8 \to 0$

Cereri: $\underbrace{4p \quad 7p \quad 5p \quad 4p}_{} \quad \underbrace{8p \quad 9p}_{} \quad \underbrace{7p \quad 6p \quad 3p}_{} \quad \underbrace{5p}_{}$ (paleți)

Încărcare: camion 1 camion 2 camion 3 camion 4

Figura 7.9

Vor fi necesare patru mijloace de transport; traseele acestora sunt indicate în figura 7.10
Lungimile celor patru trasee sunt date în tabelul 7.5:

Tabelul 7.5

Traseu	T_1	T_2	T_3	T_4	Total
Lungime (km)	88	98	86	30	302

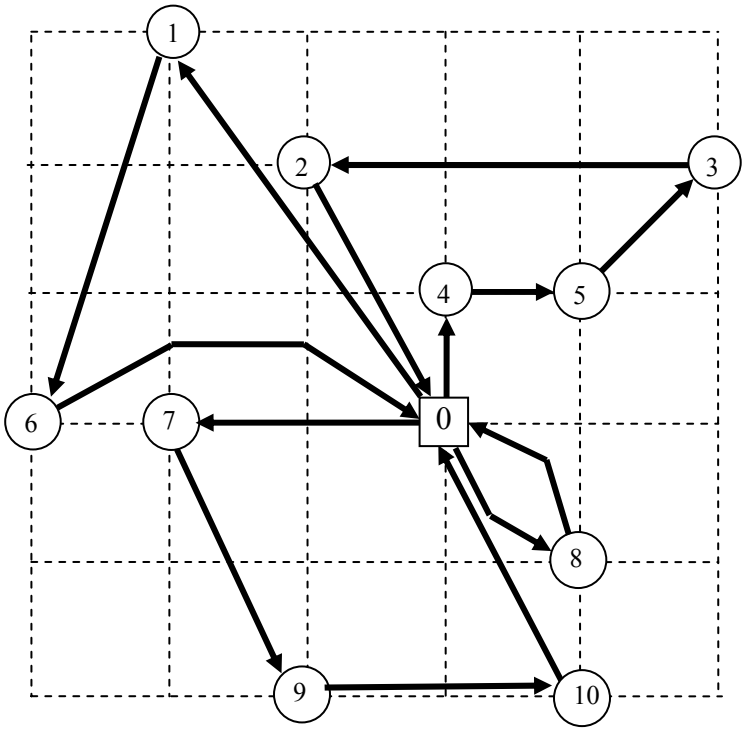


Figura 7.10

7.3.3 Aplicarea euristicii Clarke - Wright

În tabelul 7.6 au fost calculate **reducerile** $s_{ij} = d_i + d_j - d_{ij}$ pentru $1 \le i < j \le 10$:

Tabelul 7.6

	2	3	4	5	6	7	8	9	10
1	44	23	17	15	34	26	1	8	0
2	*	22	18	15	24	20	2	6	1
3		*	16	29	5	4	12	2	10
4			*	15	8	7	2	1	1
5				*	3	3	10	2	8
6					*	40	3	24	8
7						*	3	20	7
8							*	15	28
9								*	26

Aceste reduceri vor fi analizate în ordine **descrescătoare**.

În tabelul 7.7 sunt indicate traseele inițiale: sigla (i) reprezintă traseul $0 \to i \to 0$ cu lungimea $2d_i$.

Tabelul 7.7

Traseu	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	Total
Lungime (km)	72	46	58	20	30	60	40	30	46	46	448 km
Încărcare (paleți)	8	4	5	4	7	9	7	5	6	3	58 paleți

Iterația 1 Cea mai mare reducere $s_{12} = 44$ implică concatenarea traseelor (1) şi (2) în traseul $(1,2): 0 \to 1 \to 2 \to 0$ cu lungimea 72 + 46 – 44 = 78 km şi încărcarea 8 + 4 = 12 paleți – vezi tabelul 7.8

Tabelul 7.8

Traseu	(1,2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	Total
Lungime (km)	74	58	20	30	60	40	30	46	46	404=448-44
Încărcare (paleți)	12	5	4	7	9	7	5	6	3	58

Iterația 2 Se foloseşte reducerea $s_{67} = 40$ şi se concatenează traseele (6) şi (7) - vezi tabelul 7.9

Tabelul 7.9

Traseu	(1,2)	(3)	(4)	(5)	(6,7)	(8)	(9)	(10)	Total
Lungime (km)	74	58	20	30	60	30	46	46	364=404-40
Încărcare (paleți)	12	5	4	7	16	5	6	3	58

Iterația 3 Reducerea următoare $s_{16} = 34$ nu poate fi folosită deoarece concatenarea traseelor (1,2) şi (6,7) deşi posibilă nu este admisibilă ducând la o încărcare 12 + 16 = 28 superioară capacității de transport a unui camion. Putem folosi însă reducerea $s_{35} = 29$ - vezi tabelul 7.10

Tabelul 7.10

Traseu	(1,2)	(3,5)	(4)	(6,7)	(8)	(9)	(10)	Total
Lungime (km)	74	59	20	60	30	46	46	335=364-29
Încărcare (paleți)	12	12	4	16	5	6	3	58

Iterația 4 Reducerea $s_{8,10} = 28$ conduce la traseele din tabelul 7.11

Tabelul 7.11

Traseu	(1,2)	(3,5)	(4)	(6,7)	(8,10)	(9)	Total
Lungime (km)	74	59	20	60	48	46	307=335-28
Încărcare (paleți)	12	12	4	16	8	6	58

Iterația 5 Următoarea reducere $s_{17} = 26$ nu poate fi folosită din lipsă de capacități de transport. În schimb, se poate folosi reducerea $s_{9,10} = 26$.

Pentru concatenare, traseele (8,10) şi (9) vor fi aranjate astfel:

mai întâi (9): $0 \to 9 \to 0$ şi apoi (8,10): $0 \to 10 \to 8 \to 0$ astfel că rezultatul concatenării este traseul $0 \to 9 \to 10 \to 8 \to 0$ cu lungimea 46 + 48 – 26 = 68 km şi încărcarea 6 + 8 = 14 paleți - vezi tabelul 7.12

Tabelul 7.12

Traseu	(1,2)	(3,5)	(4)	(6,7)	(9,10,8)	Total
Lungime (km)	74	59	20	60	68	281=307-26
Încărcare (paleți)	12	12	4	16	14	58

Iterația 6 Abia reducerea $s_{24} = 18$ mai poate fi folosită – vezi tabelul 7.13

Tabelul 7.13

Traseu	(1,2,4)	(3,5)	(6,7)	(9,10,8)	Total
Lungime (km)	76	59	60	68	263=281-18
Încărcare (paleți)	16	12	16	14	58

Iterația 7 Din acest moment nici o altă concatenare nu mai este posibilă din lipsă de capacitate de transport.

Au rezultat patru trasee cu o lungime totală de 263 km, reprezentate în figura 7.11

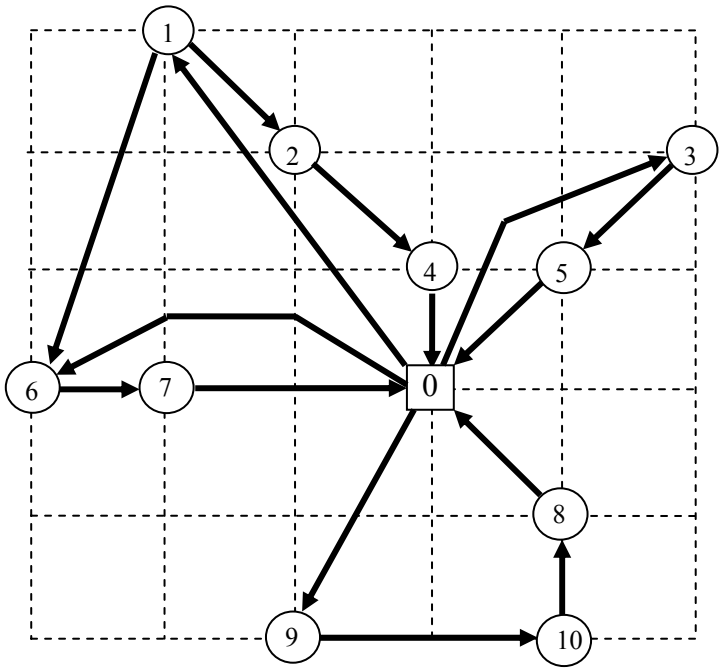


Figura 7.11

Atenție: deşi s-a obținut cel mai scurt parcurs în comparație cu celelalte două soluții nu avem nici un motiv să credem că aceasta ar fi şi soluția optimă a problemei!

Comentariu final Pentru situația dată s-au obținut trei soluții acceptabile, fiecare cu avantajele şi dezavantajele sale.

De exemplu, prima soluție utilizează numai trei vehicule față de patru câte folosesc celelalte două deşi distanța totală de parcurs este mai mare. Acest fapt s-ar putea să conteze în cazul în care vehiculele sunt închiriate.

Pe de altă parte, a treia soluție are cel mai scurt parcurs, în schimb, încărcarea vehiculelor oscilează între 60% şi 80%.

Nu este exclus ca pentru adoptarea unei hotărâri finale, factorul de decizie să solicite şi să țină seama şi de alte informații! În acest context, soluțiile construite trebuie privite doar ca scenarii de referință (recomandări).

Probleme propuse

1. Firma de construcții "Zidarii veseli" are şantiere de lucru în nouă localități P_1, P_2,…, P_9 relativ apropiate. Aprovizionarea cu ciment se face periodic – pe bază de comandă - de la un depozit central P_0. Pozițiile punctelor P_0 ,…, P_9 sunt indicate în figura 7.12 iar distanțele dintre localități – măsurate în linie dreaptă în km – sunt date în tabelul 7.14 (latura pătratului caroiajului măsoară 10 km iar distanțele reale dintre puncte nu diferă semnificativ de cele date).

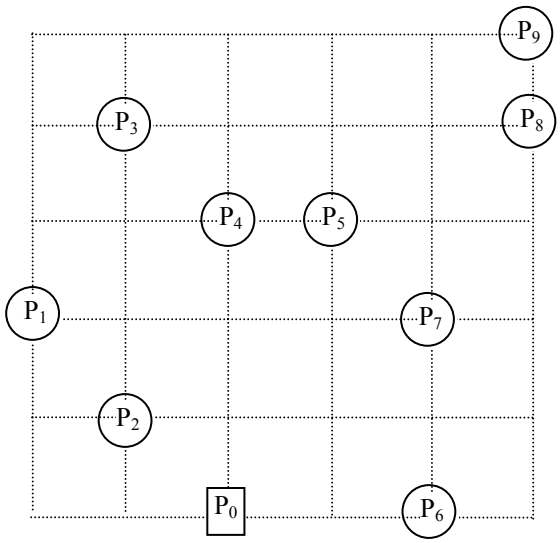


Figura 7.12

Tabelul 7.14

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
P_0	28	14	41	30	32	20	28	50	58
P_1	*	14	22	22	32	45	40	54	58
P_2		*	30	22	28	32	32	50	57
P_3			*	14	22	50	36	40	41
P_4				*	10	45	22	32	36
P_5					*	32	14	22	28
P_6						*	20	41	51
P_7							*	22	32
P_8								*	10

În tabelul 7.15 sunt date cantitățile de ciment – în saci – solicitate de şantiere pentru următoarea perioadă de lucru.

Tabelul 7.15

Şantier	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	Total
Cerere (saci)	25	14	30	40	55	32	54	20	30	300

Pentru transport firma dispune de trei camioane, fiecare cu capacitatea de 100 saci. Conducerea firmei este interesată în elaborarea unui program de transport care să specifice traseele de deplasare,

cantitățile de ciment transportate pe fiecare traseu, lungimile traseelor şi distanța totală de parcurs precum şi modul în care vor fi folosite cele trei camioane.

Utilizați euristicile prezentate şi ilustrate în text; analizați soluțiile în raport cu cerințele formulate de conducerea firmei. Comparați rezultatele şi faceți o recomandare finală.

2. Firma de panificație "Cuptorul de aur" produce un sortiment de baghete foarte apreciat a cărui desfacere este asigurată prin cele 12 magazine proprii. Fabrica producătoare şi centrele de desfacere sunt localizate în punctele P_0 , P_1 ,…, P_{12} indicate în figura 7.13. Caroiajul reprezintă de fapt şi rețeaua stradală astfel că deplasările între puncte se face numai pe liniile orizontale şi verticale ale acestuia. Latura patratului caroiajului măsoară 2 hm = 200 m.

Pentru ziua următoare, cererile magazinelor sunt date în tabelul 7.16

Tabelul 7.16

Magazin	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	Total
Cerere (bucăți)	550	400	200	275	225	175	350	75	400	450	125	375	3600

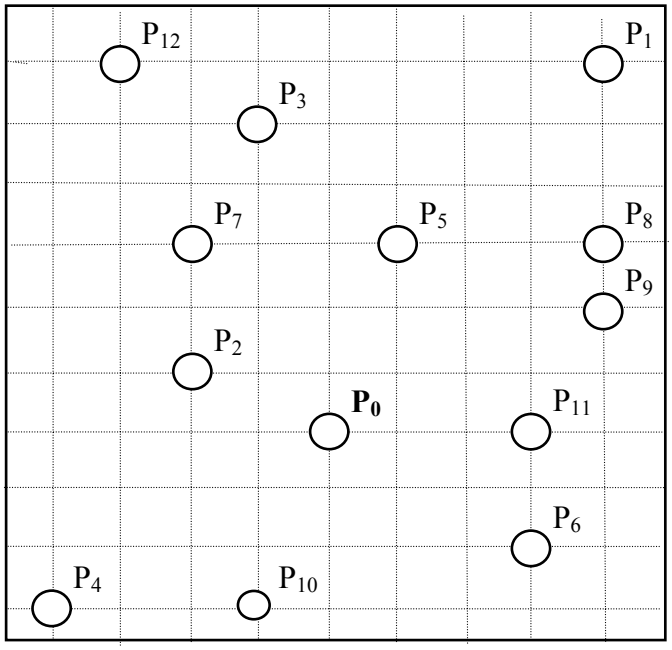


Figura 7.13

Transportul se face cu vehicule speciale cu capacitatea de 1000 bucăți fiecare.Există suficiente maşini pentru ca fiecare să nu facă mai mult de un traseu.

i) Să se determine un program de transport „bun" indicând traseele de urmat, cantitățile de transportat, lungimile traseelor şi distanța totală de parcurs (utilizați toate euristicile explicate – rezultatele vor fi diferite...)
ii) Să presupunem că sunt disponibile doar patru vehicule fiecare putând transporta 900 bucăți. Cum ar trebui ele folosite pentru ca aprovizionarea să se termine cât mai repede?

Indicație: În cazul de față, **depărtarea** dintre două puncte nu se mai măsoară „în linie dreaptă” ci folosind așa numita **distanță Manhattan** a cărei definiție este dată mai jos.

Fie A şi B două puncte dintr-un plan raportat la un sistem ortogonal de axe având coordonatele (x_A, y_A) respectiv (x_B, y_B). Distanța Manhattan dintre A şi B este valoarea:

$$dist_M(x,y) = |x_B - x_A| + |y_B - y_A|$$

Reamintim că **distanța euclidiană** „uzuală” dintre A şi B este dată de binecunoscuta formulă:

$$dist_E(x,y) = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

În general, $dist_E(x,y) \le dist_M(x,y)$ cu egalitate numai când A şi B sunt situate pe o dreaptă orizontală sau pe una verticală. Pentru interpretarea geometrică a celor două distanțe vezi figura 7.14

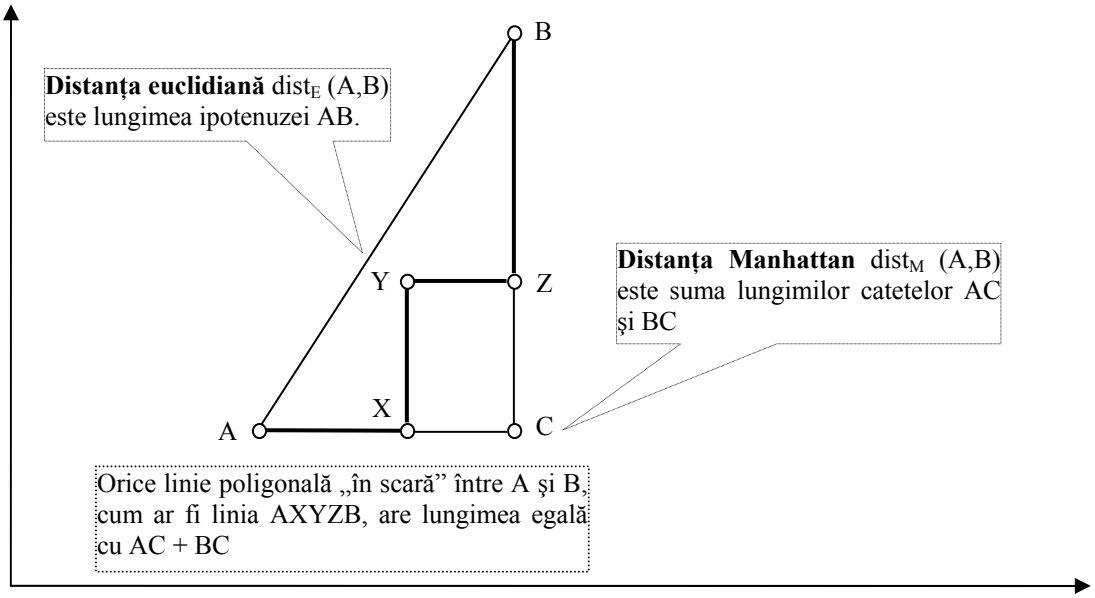


Figura 7.14

Revenind la situația din figura 7.13 distanța Manhattan dintre P_0 şi P_2 este de 1 + 2 = 3 unități de lungime adică $3 \times 200 = 600$ metri în timp ce distanța euclidiană este de $\sqrt{1^2 + 2^2} = \sqrt{5} \approx 2{,}236$ unități de lungime adică 447,2 metri. Analog, distanța dintre P_1 şi P_4 mergând numai pe orizontalele şi verticalele caroiajului este de $(8 + 9) \times 200 = 1400$ metri.

Cu aceste precizări aveți de construit tabelul distanțelor (Manhattan) dintre puncte şi tabelul reducerilor ca în ilustrările numerice date.

3. Folosiți **metoda reducerilor** pentru a găsi un traseu cât mai scurt de vizitare a punctelor P_1, P_2,…, P_{12} cu plecarea şi întoarcerea în P_0.

Unitatea de învățare 8

INTRODUCERE ÎN OPTIMIZAREA COMBINATORIALĂ
Problema poştaşului chinez (Chinese Postman Problem ≡ CPP)

Cuprins

8.1 Formularea problemei. Aplicații
8.2 Grafuri euleriene
8.3 Modelarea problemei poştaşului chinez
8.4 Soluția CPP în grafuri neorientate
8.3.1 Graful este eulerian
8.3.2 Graful nu este eulerian

Probleme propuse

Obiectivele unității de învățare 8

Reamintim că în problema comisvoiajorului se cerea determinarea unui **circuit** (traseu) care să treacă, o singură dată, prin **toate nodurile** unui graf complet şi să aibe valoarea (lungime, cost, durată) cea mai mică.
O problemă de "**rutare**" similară are în vedere muchiile: într-un graf "ponderat" şi eventual (parțial) orientat: să se determine un **circuit** prin care să se traverseze **toate muchiile** grafului şi care să aibe valoarea minimă (fireşte, cu respectarea sensului permis pe muchiile orientate!). Aceasta este în esență **problema poştaşului chinez** (**C**hinese **P**ostman **P**roblem, abreviat **CPP**)
Interesantă este complexitatea rezolvării problemei. Dacă graful este neorientat sau din contră, total orientat, CPP este o problemă "uşoară" asemănătoare problemei drumului de valoare minimă sau a arborelui de cost minim.În schimb, dacă orientarea vizează numai o parte a muchiilor grafului atunci CPP devine o problemă "grea", de calibrul problemei comisvoiajorului!!

Obiectivele principale sunt:

- formalizarea problemei în limbaj de grafuri;
- rezolvarea problemei în cazul grafurilor neorientate.

8.1 Formularea problemei. Aplicații

Următoarea problemă a fost propusă în 1962 de către matematicianul chinez MEI – KO – KWAN:

Un poştaş ridică de la oficiu corespondența de distribuit, străbate străzile din zona repartizată lui şi împarte corespondența destinatarilor după care se întoarce la locul de plecare. Cum trebuie să se deplaseze poştaşul pentru ca distanşa totală parcursă să fie minimă?

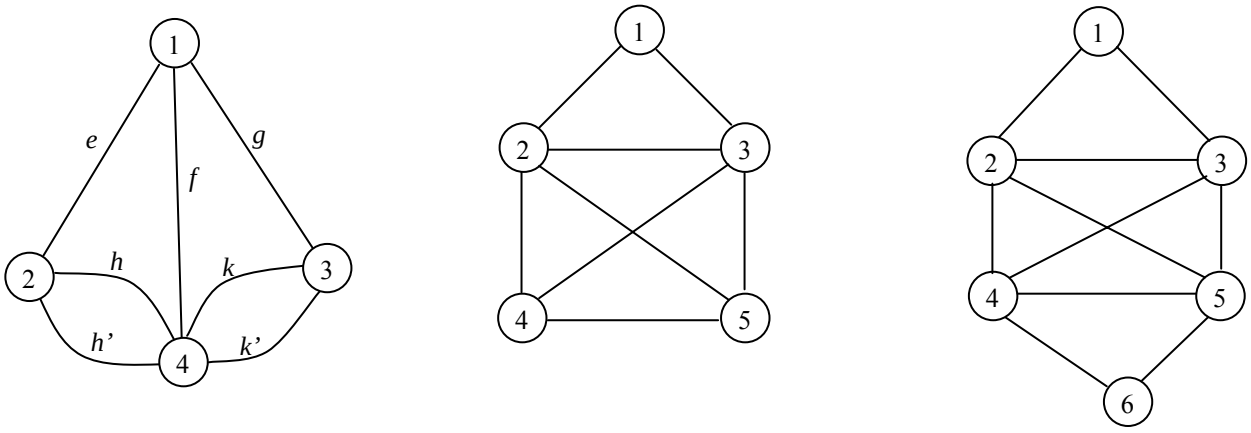
Chestiunea devine una din cele mai cunoscute şi studiate probleme de optimizare combinatorială în urma cercetărilor marelui combinatorist JACK EDMONDS care dă şi prima rezolvare eficientă în 1965.

Colectarea gunoiului menajer, deszăpezirea străzilor şi răspândirea de materiale antiderapante sau stabilirea traseelor patrulelor de poliție sunt numai câteva din numeroasele aplicații practice ale problemei poştaşului chinez.

Deoarece o rețea stradală se reprezintă de obicei printr-un graf, CPP va fi modelată în termenii teoriei grafurilor. Sunt necesare câteva pregătiri (în completarea secțiunii 4.1 din unitatea de învățare 4)

8.2 Grafuri euleriene

Fixăm un **graf** G = (V,E) în care V este mulțimea **nodurilor** şi E este mulțimea **muchiilor**. Pentru simplitate, vom presupune că G **nu este orientat**.
Un **ciclu eulerian** în graful G este un **ciclu** care conține **toate muchiile** din G exact **o singură dată**.



a) b) c)

Figura 8.1

Reamintim că graful G se zice **conex** dacă oricare două noduri diferite sunt extremitățile unui lanț de muchii (cel puțin).

Este clar că **dacă un graf are un ciclu eulerian el este conex**. **Reciproca nu este în general adevărată**; de exemplu grafurile din figurile 8.1a) şi b), deşi sunt conexe, nu au cicluri euleriene (vom vedea de ce!). În schimb, graful din figura 8.1c) are ciclul eulerian.

$$1 — 2 — 5 — 4 — 3 — 2 — 4 — 6 — 5 — 3 — 1$$

care nu este şi singurul; succesiunea

$$1 — 2 — 3 — 5 — 4 — 6 — 5 — 2 — 4 — 3 — 1$$

este un alt ciclu eulerian în acelaşi graf.

Un graf se zice eulerian dacă posedă un ciclu eulerian.

Reamintind că **gradul** unui nod dintr-un graf este **numărul muchiilor** având o extremitate în acel nod, avem următoarea caracterizare a grafurilor euleriene:

Teorema 1 (EULER, HIERHOLZER) Un graf conex este eulerian dacă şi numai dacă toate nodurile sale au grad par.

Un ciclu al unui graf se zice **elementar** dacă toate nodurile prin care trece sunt diferite. Două cicluri se vor numi **disjuncte** dacă nu au muchii comune (pot avea însă noduri comune!)

Următoarea caracterizare alternativă a grafurilor euleriene ne va fi utilă.

Teorema 2 (VEBLEN) Un graf conex este eulerian dacă şi numai dacă este o reuniune de cicluri elementare disjuncte.

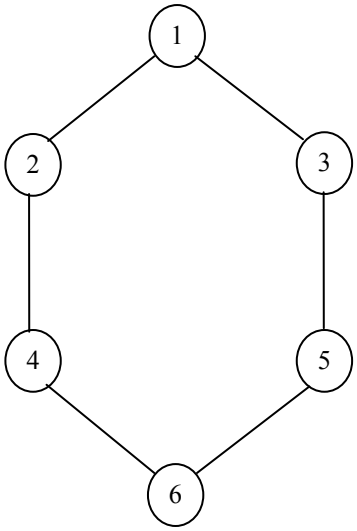
Grafurile conexe din figurile 8.1a) şi b) au noduri de grad impar şi în virtutea teoremei 1 nu sunt euleriene. Graful din figura 8.1c) este eulerian pentru că toate nodurile sale au gradul par. În baza teoremei 2 acest graf se poate descompune în mai multe cicluri elementare disjuncte. Descompunerea nu este unică: figura 8.2 evidențiază două descompuneri distincte ale grafului amintit.

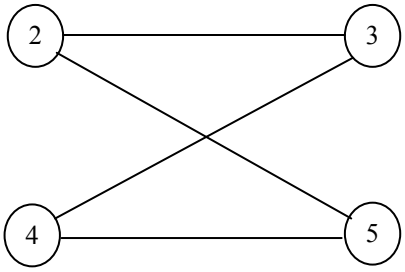
8.3 Modelarea problemei poştaşului chinez

Un **traseu poştal** în graful G este un **ciclu** care conține **toate muchiile** lui G, **cel puțin o dată**.

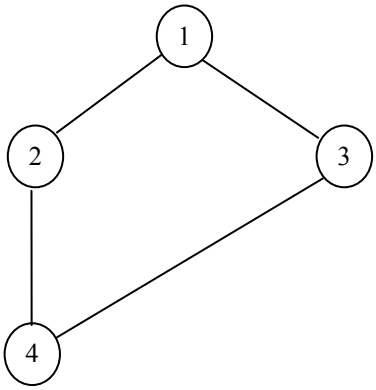
Este evident că:

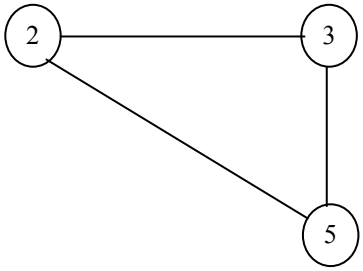
Un ciclu eulerian este un traseu poştal. Orice graf conex posedă cel puțin un traseu poştal.

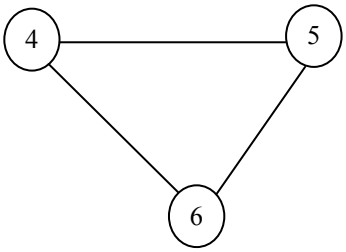




I) Graful din figura 1c) a fost descompus în două cicluri elementare disjucte







II) O descompunere alternativă în trei cicluri elementare disjucte

Figura 8.2

De exemplu, în graful din figura 8.1a) ciclul

$$1 \overset{e}{\text{——}} 2 \overset{h}{\text{——}} 4 \overset{k}{\text{——}} 3 \overset{g}{\text{——}} 1 \overset{f}{\text{——}} 4 \overset{k'}{\text{——}} 3 \overset{k}{\text{——}} 4 \overset{h'}{\text{——}} 2 \overset{e}{\text{——}} 1$$

este un traseu poştal în care muchiile *k* şi *e* apar de câte două ori. În graful din figura 8.1b) ciclul

$$1 \text{——} 2 \text{——} 4 \text{——} 3 \text{——} 2 \text{——} 5 \text{——} 4 \text{——} 5 \text{——} 3 \text{——} 1$$

este un traseu poştal în care se repetă muchia $\{4,5\}$.

Numărul muchiilor unui traseu poştal este cel puțin egal cu numărul total de muchii din graf şi este egal cu acesta numai dacă traseul este un ciclu eulerian.

Să presupunem în continuare că fiecărei muchii $e \in E$ din G i s-a asociat o **valoare (pondere)** $l(e) \geq 0$ care în aplicații poate însemna o distanță, un timp sau un cost. Ca de obicei, valoarea unui lanț de muchii va fi dată de suma valorilor muchiilor alcătuitoare.Cu aceste pregătiri, problema poştaşului chinez capătă următoarea descriere formală:

I. Fiind dat graful conex G ale cărui muchii au fost ponderate cu valori nenegative, să se determine în G un traseu poştal cu valoarea minimă.

Două generalizări sunt evidente şi aplicativ interesante:

II. Fiind dat graful conex şi total orientat G cu arcele nenegativ ponderate, să se determine în G un traseu poştal de valoare minimă a cărui parcurgere să respecte oorientările tuturor muchiilor.

Bineînțeles că în acest caz se va opera cu **„conceptele orientate”** corespunzătoare unui lanț respectiv unui ciclu şi anume **drum** respectiv c**ircuit**.

III. Se dă un graf conex G în care există şi muchii neorientate ce pot fi parcurse în orice sens şi muchii orientate (arce) care nu pot fi traversate decât într-un singur sens, prestabilit.Presupunând că toate muchiile şi arcele au fost ponderate cu valori nenegative, să se determine în G un traseu poştal de valoare minimă, a cărui parcurgere să fie compatibilă cu orientările arcelor.

În timp ce CPP în grafuri **neorientate** sau **total orientate** s-a dovedit a fi o problemă de optimizare combinatorială **uşoară**, aceeaşi chestiune în grafuri **mixte** este o problemă **grea** de „calibrul” problemei comisvoiajorului!

8.4 Soluția CPP în grafuri neorientate

Fixăm un graf **conex, neorientat** G = (V,E) cu ponderi nenegative asociate muchiilor. Sunt două situații de studiat după cum G este sau nu este **graf eulerian**. Proprietatea unui graf de a fi eulerian se verifică uşor calculând **gradele** tuturor nodurilor: dacă toate aceste grade sunt numere **pare** graful este eulerian (teorema 1)

8.4.1 Graful este eulerian

În acest caz un traseu poştal de valoare minimă este dat de un ciclu eulerian. Practic nu avem ce optimiza întrucât toate ciclurile euleriene au aceeaşi valoare, egală cu suma valorilor muchiilor din graf! Problema se reduce la **generarea algoritmică** a unui asemenea ciclu eulerian. Următoarea construcție are la bază posibilitatea descompunerii unui graf eulerian într-o reuniune de cicluri elemetare disjuncte (teorema 2).

- Fixăm un nod s_1 din care va începe şi în care se va sfârşi ciclul eulerian căutat;
- Plecăm din s_1 şi căutăm să parcurgem cât mai multe muchii din G având grijă ca fiecare muchie să fie traversată o singură dată.Deoarece toate nodurile au grad par, mai devreme sau mai târziu vom reveni în s_1, obținând un ciclu C_1. Dacă C_1 conține toate muchiile grafului G, acesta va fi ciclul eulerian căutat. În caz contrar:
- Eliminăm din G toate muchiile ciclului C_1 apoi alegem şi fixăm un nod s_2 prin care trece C_1 şi care este extremitate pentru cel puțin o muchie neştearsă;
- Acum plecăm din s_2 şi căutăm să traversăm cât mai multe dintre muchiile rămase, o singură dată. La revenirea în s_2 obținem un al doilea ciclu C_2 , disjunct de C_1;
- Procesul continuă fie dintr-un nod intermediar al lui C_2 fie dintr-unul al ciclului C_1 obținându-se un nou ciclu C_3. Dacă este cazul, procesul se reia dintr-un nod prin care trece C_3 sau C_2 sau chiar C_1 rezultând ciclul C_4 ş.a.m.d.
- În final rezultă o descompunere a grafului G într-o reuniune de cicluri disjuncte C_1 , C_2 ,..., C_s ciclul C_i având drept nod inițial un nod intermediar al unuia dintre ciclurile anterior generate C_1 ,..., C_{i-1} În general, descompunerea nu este unică;
- Se trece la **concatenarea** ciclurilor C_1 , C_2 ,..., C_s într-un **ciclu eulerian**. Pentru simplitate să presupunem că sunt numai două cicluri C_1 şi C_2:
  - ciclul C_1 începe şi sfârşeşte într-un nod s_1;
  - ciclul C_2 începe şi sfârşeşte într-un nod s_2 care este nod intermediar al ciclului C_1 (atenție, s_2 poate să coincidă cu s_1!)

Plecăm din s_1 traversând muchiile ciclului C_1 într-un sens fixat. O dată ajunşi în s_2 vom parcurge – într-un sens fixat – toate muchiile ciclului C_2. La revenirea în s_2, continuăm deplasarea pe muchiile netraversate din ciclul C_1 până când ajungem în nodul s_1 de unde am plecat.
Schema descrisă este evident generalizabilă la mai mult de două cicluri!

Exemplu ilustrativ Un poştaş are de străbătut străzile vizualizate în graful din figura 8.3 (nodurile reprezintă extremitățile străzilor) În ce ordine va trebui el să parcurgă străzile pentru ca distanța totală parcursă să fie minimă? Punctul de plecare şi întoarcere este, să zicem, nodul a.
Soluție: Deoarece graful este conex şi toate nodurile au gradul par, el este eulerian, astfel că traseul de lungime minimă este un ciclu eulerian. Ca urmare distanțele nu au nici o importanță, distanța totală (minimă) de parcurs, fiind de fapt, suma lungimilor tuturor străzilor!
O descompunere posibilă a grafului într-o reuniune de cicluri elementare disjuncte este dată în figura 8.4; săgețile de pe muchii

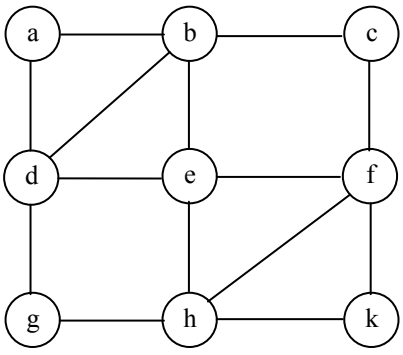


Figura 8.3

indică sensul de deplasare:

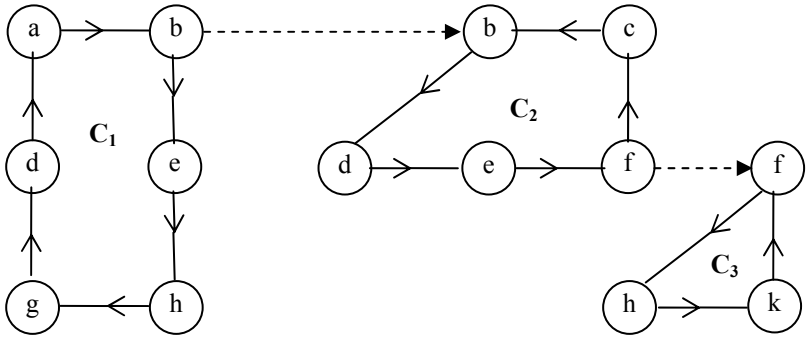


Figura 8.4

Concatenarea ciclurilor rezultate conduce la **ciclul eulerian**:

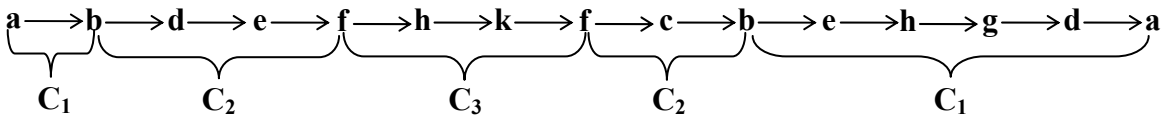


Figura 8.5

8.4.2 Graful nu este eulerian

Aceasta înseamnă că în G există noduri de grad impar! Se ştie că **numărul nodurilor de grad impar este întotdeauna par.**

În acest caz, orice traseu poştal va trece în mod sigur **de mai multe ori** prin anumite muchii.

În continuare, vom considera numai trasee poştale T în care fiecare muchie este traversată **cel mult de două ori** deoarece se poate arăta uşor că traseul T* de valoare minimă are această proprietate. Valoarea unui asemenea traseu va fi atunci egală cu:

$$l(T) = l(E) + l'$$

unde $l(E)$ este suma valorilor tuturor muchiilor din G iar l' este suma valorilor muchiilor traversate încă o dată. Deoarece graful G este presupus a nu fi eulerian, cu siguranţă $l' > 0$.În concluzie, problema poştaşului chinez în grafuri neorientate se reduce la identificarea acelor muchii ce vor fi traversate de două ori pentru care valoarea excedentară l' este minimă. Vom studia mai întâi o situaţie concretă.

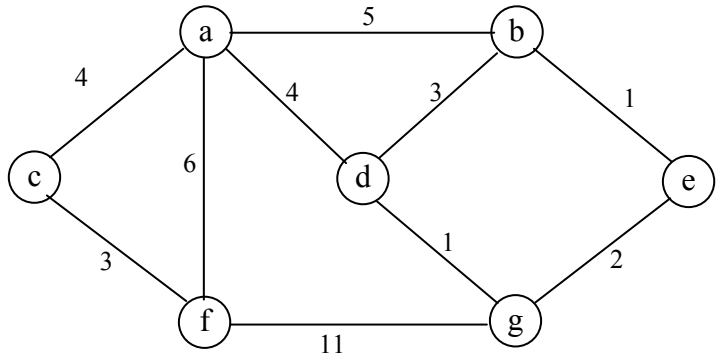


Figura 8.6

Exemplu ilustrativ Graful din figura 8.6 vizualizează o rețea de străzi. În fiecare noapte un echipaj de poliție trebuie să patruleze pe toate străzile veghind la respectarea liniştii şi a ordinii publice. Din motive evidente se doreşte determinarea traseului de lungime minimă. Valorile numerice înscrise pe muchii reprezintă distanțe (în sute de metri). Secția de poliție – locul de plecare şi întoarcere al patrulei – este localizată în nodul a.

În primul rând, graful nu este eulerian deoarece nodurile b, d, f şi g au gradul 3 (impar!). Ca urmare, în orice traseu poştal vor exista muchii traversate de mai multe ori! De exemplu, în traseul

$$\mathbf{T: a \longrightarrow d \longrightarrow b \longrightarrow d \longrightarrow g \longrightarrow f \longrightarrow a \longrightarrow b \longrightarrow e \longrightarrow g \longrightarrow f \longrightarrow c \longrightarrow a}$$

muchiile {b , d}şi {f , g}sunt utilizate de două ori. Lungimea traseului este:

$$l(T) = l(E) + l' = 40 + (3 + 11) = 54 \text{ sute metri.}$$

În schimb, traseul

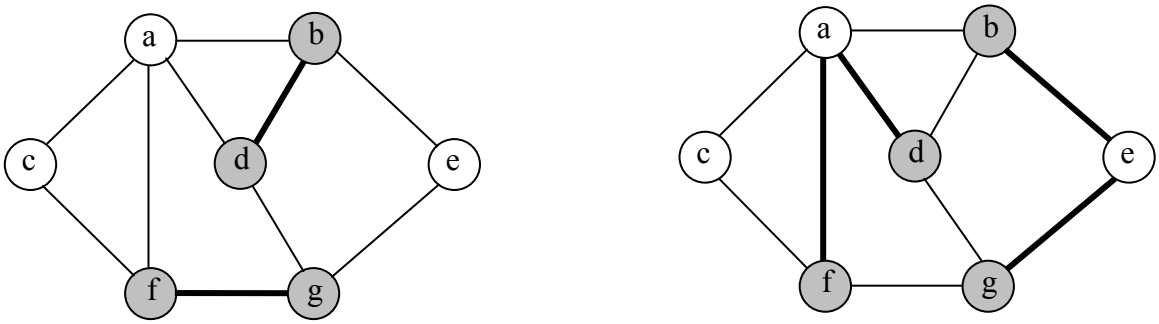
$$\mathbf{T':a \longrightarrow f \longrightarrow a \longrightarrow d \longrightarrow g \longrightarrow e \longrightarrow b \longrightarrow d \longrightarrow a \longrightarrow c \longrightarrow f \longrightarrow g \longrightarrow e \longrightarrow b \longrightarrow a}$$

„dublează" muchiile {a , f}, {a , d}, {b , e}, {g , e}şi ca urmare are lungimea:

$$l(T') = 40 + (6 + 4 + 1 + 2) = 53 \text{ sute metri.}$$

Examinând cele două trasee se constată că muchiile „dublate" formează **lanțuri** având ca extremități **perechi** de noduri de grad **impar**!

A se vedea figura 8.7a) şi b) în care muchiile traversate de două ori în T respectiv T' au fost îngroşate iar nodurile de grad impar au fost haşurate.



a) b)

Figura 8.7

Observaţia este valabilă în general şi în consecinţă, relativ la traseul optim T*, sunt adevărate următoarele afirmaţii:

- nodurile de grad **impar** sunt **cuplate** prin lanţuri de muchii ce vor fi traversate **de două ori**;
- fiecare **lanţ de cuplare** are lungimea **cea mai mică**;
- **cuplarea** nodurilor de grad impar este astfel făcută încât suma lungimilor lanţurilor de cuplare să fie **minimă**.

În exemplul nostru, sunt posibile următoarele cuplări ale nodurilor de grad impar b , d , f şi g:

b cu d şi f cu g ; b cu f şi d cu g ; b cu g şi d cu f

Pentru a stabili cuplajul optim avem nevoie de distanţele cele mai mici dintre nodurile b, d, f şi g ca şi de lanţurile de muchii pe care ele se realizează! Ele sunt calculate în tabelul 8.1 În tabelul 8.2 sunt analizate cele trei posibilităţi de cuplare.

Cuplajul cu cea mai mică pondere este „b cu f şi d cu g". În consecinţă, în traseul poştal optim, muchiile lanţului b — a — f care cuplează nodurile b şi f vor fi traversate de două ori. Tot de două ori va fi traversată şi muchia d — g care cuplează nodurile d şi g.

Tabelul 8.1

	d	f	g
b	3 b — d	11 b — a — f	3 b — e — g
d	*	10 d — a — f	1 d — g
f		*	11 f — g

Tabelul 8.2

Posibilităţi de cuplare	Suma lungimilor lanţurilor de cuplare
b cu d şi f cu g	3 + 11 = 14
b cu f şi d cu g	11 + 1 = 12
b cu g şi d cu f	3 + 10 = 13

Pentru găsirea traseului poştal optim construim un graf G*, derivat din graful original G (din figura 8.6) prin dublarea muchiilor celor două lanțuri de cuplare – vezi figura 8.8.

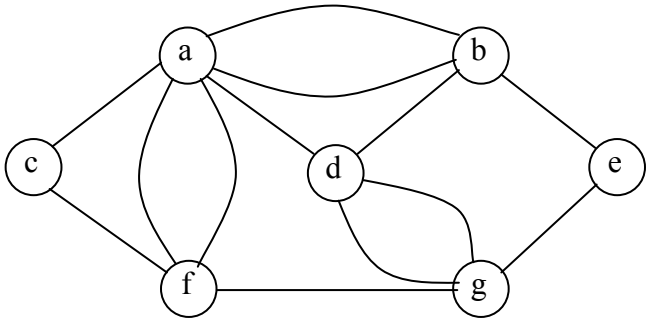


Figura 8.8

Prin construcție G* este un graf eulerian şi o modalitate de parcurgere a muchiilor sale reprezintă – în graful original G – un traseu poştal T* cu cea mai mică valoare:

$$l(T^*) = 40 + 12 = 52 \text{ sute metri.}$$

Putem lua, de exemplu:

T*: a ⟶ f ⟶ g ⟶ d ⟶ b ⟶ a ⟶ c ⟶ f ⟶ a ⟶ d ⟶ g ⟶ e ⟶ b ⟶ a

Cu observația că operațiile întreprinse în acest caz particular sunt valabile şi în cazul general avem următorul **algoritm de rezolvare a problemei poştaşului chinez într-un graf (conex) neorientat G care nu este eulerian:**

Etapa 1 Se identifică nodurile din G de grad impar (acestea sunt în număr par).

Etapa 2 Pentru oricare două noduri diferite *x*, *y* de grad impar se identifică un lanț de muchii de valoare **minimă** cu extremitățile *x*, *y*. Operația se realizează, de exemplu, cu algoritmul lui Dijkstra.

Etapa 3 Se construieşte un **graf complet** K ale cărui noduri corespund nodurilor de grad impar din G. Fiecare muchie $\{x , y\}$ din K se ponderează cu valoarea minimă a lanțului de muchii din G care uneşte *x* cu *y*.

Etapa 4 Se cuplează nodurile din K două câte două astfel încât suma ponderilor muchiilor de cuplare să fie **minimă**. Operația se realizează cu un algoritm specific de rezolvare a problemei cuplajului (maxim) de pondere minimă.

Etapa 5 Fiecare muchie $\{x , y\}$din K ce face parte din cuplajul determinat în etapa precedentă corspunde unui lanț de muchii din G cu extremitățile *x*, *y* pe care îl numim **lanț de cuplare**.Construim un nou graf G* care derivă din G prin **dublarea** muchiilor lanțurilor de cuplare dintre nodurile de grad impar. Prin construcție, G* este un graf eulerian.

Etapa 6 Se identifică în G* un **ciclu eulerian** T*. În raport cu graful original G ciclul T* este un **traseu poştal de valoare minimă**.

Probleme propuse

1. Pentru fiecare dintre grafurile din figura 8.9

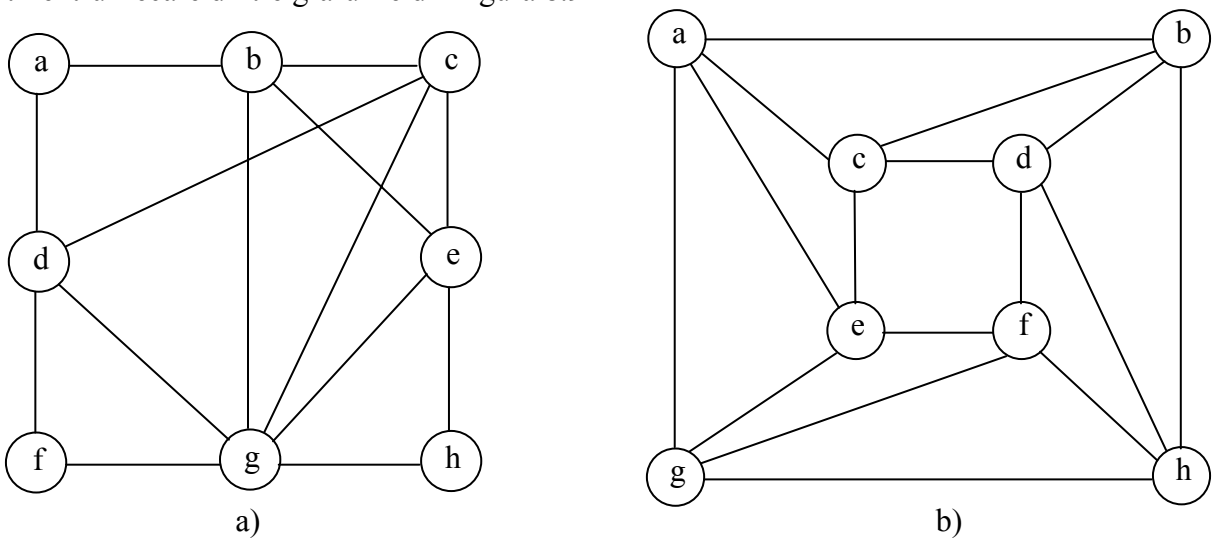


Figura 8.9

i) Completaţi tabelul:

Nod	a	b	c	d	e	f	g	h
Grad								

:

ii) Este posibilă traversarea tuturor muchiilor grafului, fiecare o singură dată? Cum?

2. Cu ce este egală suma gradelor nodurilor unui graf? Aplicaţie: un graf are 10 noduri şi suma gradelor acestora este 16. Arătaţi că graful nu este conex!

3. Într-un graf conex şi neorientat G ale cărui muchii au fost ponderate cu valori nenegative se consideră un traseu poştal T* de valoare minimă. Probaţi că orice muchie din G apare în T* cel mult de două ori.

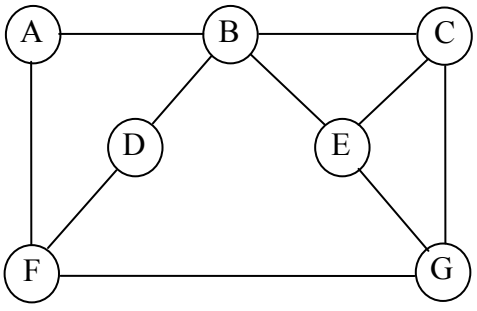


Figura 8.10

4. Graful din figura 8.10 reprezintă stilizat o reţea de zece trasee turistice care leagă cabanele A,B,...,G. Traseele sunt lungi şi destul de obositoare astfel că o persoană plecată dimineaţa de la o cabană oarecare pe un traseu oarecare trebuie să înnopteze – pentru odihnă şi refacere – la cabana

de la celălalt capăt al traseului parcurs. Un turist intenționează să parcurgă toate aceste trasee – câte unul pe zi – cu plecarea şi întoarcerea în cabana A. În câte zile se poate face această excursie? (este vorba de durata minimă...) Ce trasee vor trebui parcurse de mai multe ori?

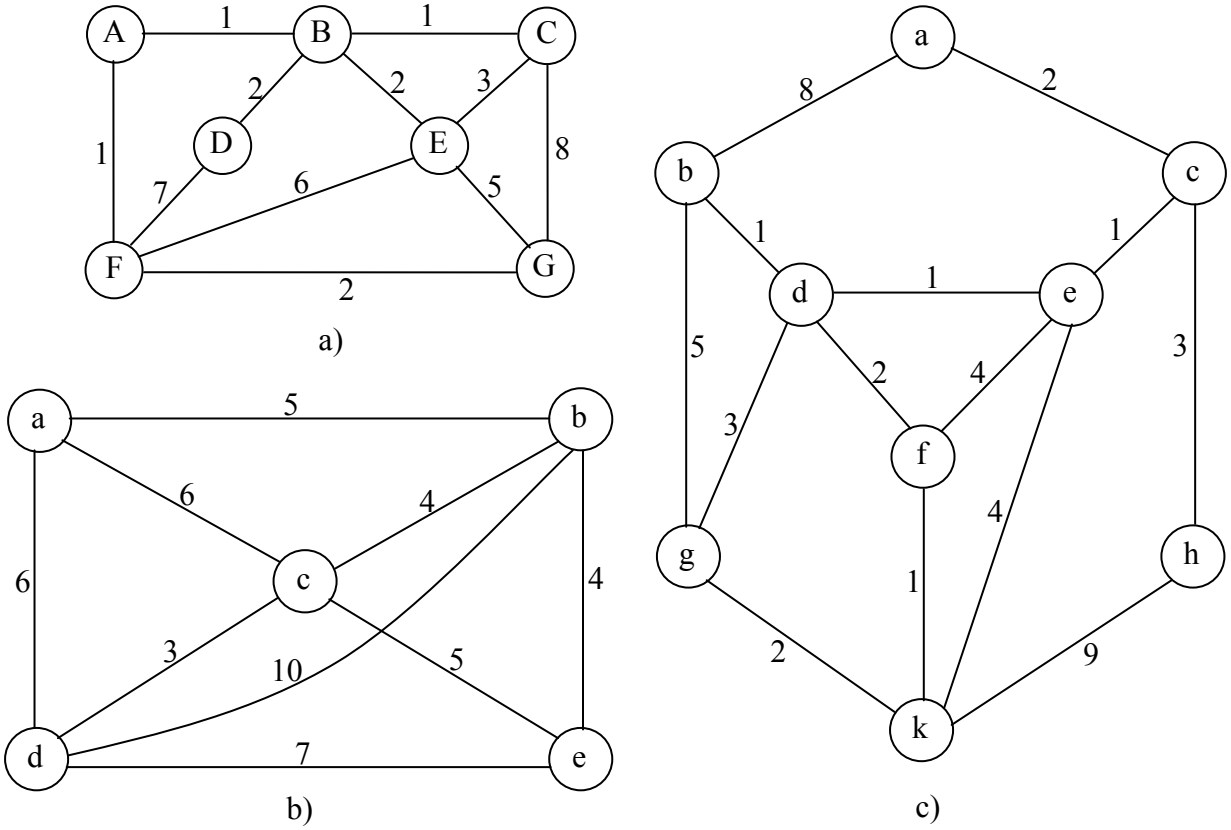


Figura 8.11

5. Pentru fiecare graf din figura 8.11 determinați un traseu poştal de lungime minimă.

Indicație: identificați nodurile de grad impar.Dacă sunt mai mult de două, cuplațile în toate modurile posibile (ca în exemplul din text) şi determinați cuplajul cu cea mai mică pondere. Distanțele minime dintre nodurile de grad impar vor fi evaluate prin inspecție directă.

6. În nordul capitalei s-a construit un nou cartier. Vezi figura 8.12 în care muchiile reprezintă străzile iar nodurile au semnificația de intersecții. Valorile numerice înscrise pe muchii reprezintă distanțe. Comunitatea locală a contractat cu o firmă de salubritate colectarea deşeurilor menajere în următoarele condiții: periodic, un singur vehicul are voie să intre în cartier prin punctul *x* şi trebuie să părăsească zona prin punctul *y* după ce a parcurs toate străzile cartierului. Din motive evidente, operația trebuie să se termine cât mai repede. Viteza vehiculului fiind aproximativ constantă, care este traseul de lungime minimă?

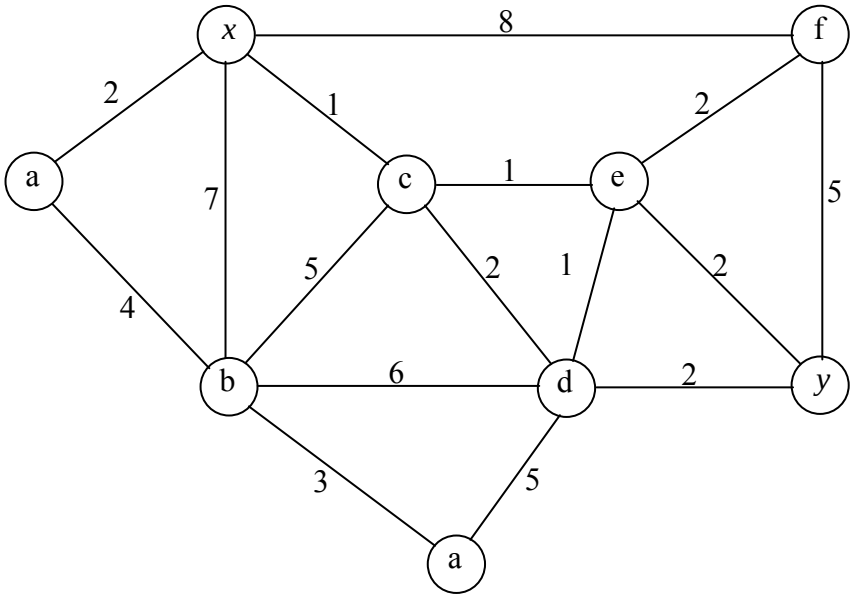


Figura 8.12

Indicație: ne reducem la CPP în felul următor:

- Introducem în graf un nod I şi două noi muchii {I , *x*} şi {I , *y*}
- Ponderăm muchiile introduce cu o valoare foarte mare M aşa încât ele să nu poată fi dublate!
- În graful extins determinăm un traseu poştal de lungime minimă care începe şi sfârşeşte în I datorită condiției precedente traseul va avea forma:

$$I — x — \ldots \ldots — y — I$$

- Ignorând muchiile $I — x$ şi $y — I$ obținem traseul dorit.

Unitatea de învățare 9

INTRODUCERE ÎN OPTIMIZAREA COMBINATORIALĂ

Problema echilibrării liniilor de asamblare
(Assembly Line Balancing Problem ≡ ALBP)

Cuprins

9.1 Problema echilibrării liniilor de asamblare
9.2 Reprezentarea mulțimii operațiilor
9.3 Modelarea ALBP
9.4 Reducerea ALBP la o problemă de drum minim
9.5 Algoritmi de rezolvare suboptimală a ALBP
9.6 Ilustrări numerice

Probleme propuse

Obiectivele unității de învățare 9

Organizarea producției pe linii de asamblare a fost inițiată de HENRY FORD la începutul secolului al XX-lea şi a avut o influență enormă asupra creşterii productivității muncii dar şi asupra personalului muncitor obligat la o strictă specializare cu toate consecințele care decurg de aici.

Descrierea riguroasă a conceptului, formularea unei probleme de optimizare şi modelarea acesteia constituie un prim obiectiv al acestei unități de învățare.În al doilea rând se va studia posibilitatea rezolvării suboptimale a problemei ştiut fiind că echilibrarea liniilor de asamblare face parte din clasa problemelor grele de optimizare combinatorială.

9.1 Problema echilibrării liniei de asamblare

Ideea pe care se bazează conceptul de linie de asamblare este simplă: orice produs finit se poate descompune în ansamble, acestea în subansamble care la rândul lor se desfac în sub-subansamble ş.a.m.d. până când se ajunge la nivelul materialelor primare. Dacă se inversează această descompunere, se pot defini activitățile care permit transformarea materialelor primare în repere simple, a acestora în subansamble ş.a.m.d. până se obține produsul finit. Activitățile sus amintite se vor numi în continuare **operații**.

O linie de asamblare pentru un produs este în esență o succesiune de posturi de lucru în care se execută grupat, operațiile specifice produsului respectiv

Este evident.că în timp ce unele operații pot fi făcute în aproape orice post al liniei – ca de exemplu fixarea unei etichete pe capacul unui motor – marea majoritate a operațiilor trebuie executate într-o anumită **ordine** (de exemplu, o gaură trebuie mai întâi străpunsă şi apoi filetată). Ca urmare, **gruparea operațiilor pe posturi de lucru** trebuie astfel făcută încât, pentru fiecare dintre ele, toate operațiile premergătoare să fie executate fie în cadrul aceluiaşi post fie în posturi situate **în amonte** pe linia de asamblare.

Dacă Q este **productivitatea** liniei de asamblare, exprimată în unități de produs pe unitatea de timp, inversul $c = 1/Q$ reprezintă intervalul de timp între două produse finisate succesive.Este clar că toate operațiile care se execută în cadrul unuia sau altuia dintre posturile de lucru trebuie terminate în acest interval de timp numit şi **ciclul** liniei de asamblare.

De exemplu, dacă productivitatea unei linii de fabricat prăjitoare de pâine este de 8 bucăți pe oră, urmează că intervalul de timp între două produse succesive este de $60/8 = 7{,}5$ minute şi în consecință la fiecare post de lucru suma duratelor aferente postului nu trebuie să depăşească ciclul de 7,5 minute.

O posibilă formulare a **problemei echilibrării liniilor de asamblare** (**A**ssembly **L**ine **B**alancing **P**roblem – abreviat **ALBP**) este următoarea:

Fiind date:

- **Mulțimea** I a **operațiilor** necesare fabricării unui produs împreună cu **relațiile de precedență** dintre operații;
- **Duratele** $d(x)$, $x \in I$ ale operațiilor;
- **Ciclul** de timp c exprimat în unități de timp pe unitatea de produs (se presupune că $c \geq d(x)$, $(\forall)x \in I$);

Se cere:
Gruparea operațiilor într-un număr **minim** de posturi de lucru astfel încât:

- La fiecare post timpul necesar executării tuturor operațiilor aferente acestuia să nu depăşească ciclul c;

- Pentru fiecare operație executată la un anumit post, toate operațiile premergătoare să fie executate fie în cadrul aceluiaşi post fie în posturi de lucru situate în amonte pe linia de asamblare.

9.2 Reprezentarea mulțimii operațiilor

În mulțimea I a operațiilor, **relația binară**:

"operația x **precede** operația y"

notată $x \prec y$ are proprietățile:

- $\overline{x \prec x}$ ($\equiv$ nici o operație nu se precede pe ea însăşi);
- $x \prec y$ şi $y \prec z \Rightarrow x \prec z$ ($\equiv$ tranzitivitate)

şi, prin urmare, este o **relație de ordine** (strictă) pe I. Deoarece pot exista operații **necomparabile (independente),** relația $\prec$ este în general o relație de ordine **parțială**.

Vom spune că "operația x **precede direct** operația y"şi notăm $x \overset{d}{\prec} y$ dacă:

$$x \prec y \text{ şi nu există } z \in I \text{ astfel încât } x \prec z \prec y$$

Altfel spus, x precede direct y dacă y se poate executa **imediat** după terminarea operației x.
Precedențele directe determină complet toate precedențele posibile dintre operațiile ce compun mulțimea I, în sensul că:

$$x \prec y \Leftrightarrow \text{ există "şirul de precedențe directe" : } x = x_0 \overset{d}{\prec} x_1 \overset{d}{\prec} \cdots \overset{d}{\prec} x_{p-1} \overset{d}{\prec} x_p = y$$

Mulțimea parțial ordonată I poate fi vizualizată printr-un **graf orientat** în care:

- **nodurile** se identifică cu operațiile din I;
- **arcele** au forma (x,y) cu $x \overset{d}{\prec} y$ (arcele pun deci în evidență precedențele directe dintre operații)

Exemplul 1 Considerăm un produs ce poate fi realizat prin nouă operații. Precedențele directe dintre operații şi duratele acestora sunt date în tabelul 9.1

Tabelul 9.1

Operația	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Durata (minute)	5	3	4	6	5	3	4	5	3
Operații direct **succesoare**	**2 ; 3**	**4 ; 5**	**5 ; 7**	**8**	**9**	**8 ; 9**	**9**	**-**	**-**

În figura 9.1 a fost vizualizat **graful operațiilor.**

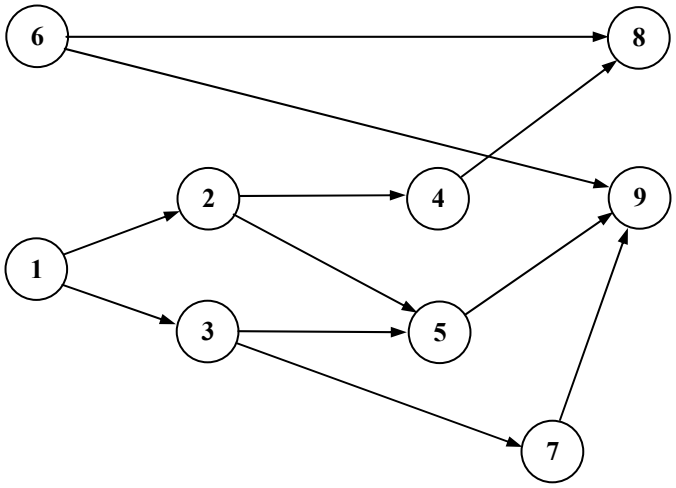


Figura 9.1

9.3 Modelarea ALBP

Cu notațiile şi pregătirile din secțiunile precedente, problema echilibrării liniei de asamblare se formalizează astfel:

Să se determine **şirul de submulțimi nevide** ale mulțimii I:

$$(S_1 , S_2 , \cdots , S_r)$$

cu proprietățile:

1) $\bigcup_{i=1}^{r} S_i = I$;

2) $S_i \cap S_j = \varnothing$ pentru $i \neq j$;

3) $d(S_i) \underset{def}{=} \sum_{x \in S_i} d(x) \leq c$;

4) $x \prec y$ şi $x \in S_i$, $y \in S_j \quad \Rightarrow \quad i \leq j$.

astfel încât să se **minimizeze** expresia:

5) $T = \sum_{i=1}^{r} [c - d(S_i)]$

Observații:

- Posturile de lucru se identifică cu submulțimile S_1 , S_2 , … , S_r ;

- Condițiile 1) şi 2) arată că fiecare operație se execută în cadrul unui singur post;

- Condiția 3) exprimă cerința compatibilității grupării operațiilor pe posturi cu relațiile de precedență dintre operații: dacă o operație y se execută la un post S_j atunci orice operație x care o precede trebuie executată fie în acelaşi post S_j fie într-un post S_i cu $i < j$, altfel spus, într-un post anterior în şirul $(S_1, S_2, \cdots, S_r)$;

- La prima vedere s-ar părea că obiectivul optimizării a fost schimbat! În locul minimizării numărului de posturi de lucru – adică a lui r – se cere minimizarea sumei **timpilor morți** înregistrați la fiecare post, altfel spus se urmăreşte **echilibrarea** posturilor în sensul ocupării cât mai bune a timpului reprezentat de ciclu.

În realitate cele două obiective sunt echivalente pentru că:

$$T = \sum_{i=1}^{r}[c - d(S_i)] = r \cdot c - \sum_{i=1}^{r} d(S_i) = r \cdot c - d(I)$$

unde $d(I) = \sum_{x \in I} d(x) \equiv$ durata tuturor operațiilor din I.

Este clar acum că minimizarea lui T înseamnă minimizarea lui r deoarece c şi $d(I)$ sunt constante.

9.4 Reducerea ALBP la o problemă de drum minim

În terminologia şi notațiile introduse în secțiunile precedente, o mulțime $A \subseteq I$ se numeşte **ideal** al mulțimii parțial ordonate I dacă

$$x \prec y \text{ şi } y \in A \quad \Rightarrow \quad x \in A$$

Fie $\mathcal{A}$ mulțimea idealelor lui I. Evident $\varnothing \in \mathcal{A}$, $I \in \mathcal{A}$.

Exemplul 2 În mulțimea parțial ordonată $I = \{1,2,\cdots,8,9\}$ din exemplul 1, submulțimea $A = \{1,2,3,4,7\}$ este un ideal nevid, însă mulțimea $A' = \{1,4,5,7\}$ nu este ideal deoarece, de exemplu $3 \prec 5$, $5 \in A'$ dar $3 \notin A'$.

Pentru fiecare ideal $A \in \mathcal{A}$ punem $d(A) = \sum_{x \in A} d(x)$; pentru idealul vid vom lua $d(\varnothing) = 0$.

Construim un **graf orientat** $\mathcal{G}$ ale cărui noduri se identifică cu idealele mulțimii I, arcele având forma (A,B) unde

$$A, B \in \mathcal{A} \ ; \ A \subset B \ ; \ d(B) - d(A) \le c$$

Să considerăm acum un **drum** în graful $\mathcal{G}$ de la nodul $\varnothing$ la nodul I:

$$\varnothing = A_0 \subset A_1 \subset A_2 \subset \cdots \subset A_{r-1} \subset A_r = I$$

Fie

$$S_i = A_i \setminus A_{i-1} \quad i = 1, \cdots, r$$

(deci S_i se compune din elementele din A_i care nu sunt în A_{i-1}). Este clar că:

$$\bigcup_{i=1}^{r} S_i = I \text{ şi } S_i \cap S_j = \varnothing \quad \text{pentru } i \neq j$$

Mai departe:

$$d(S_i) = d(A_i) - d(A_{i-1}) \leq c$$

Fie acum $x \prec y$ două operaţii din I cu $x \in S_i$ şi $y \in S_j$. Din $y \in S_j$ şi $x \prec y$ rezultă $x \in S_j$ deoarece A_j este un ideal. Dacă prin absurd am avea $i > j$ atunci $i - 1 \geq j$ de unde ar rezulta incluziunea $A_j \subseteq A_{i-1}$. Atunci $x \in A_{i-1}$ ceeace contrazice ipoteza că $x \in S_i = A_i \setminus A_{i-1}$; în concluzie $i \leq j$.

Cele de mai sus au arătat că şirul de submulţimi nevide din I

$$(S_1 = A_1 \, , S_2 = A_2 \setminus A_1 \, , \cdots , S_r = A_r \setminus A_{r-1})$$

satisface condiţiile 1 – 4 ale modelului matematic al problemei echilibrării liniei de asamblare.

Asociind fiecare parte $S_i = A_i \setminus A_{i-1}$ cu arcul (A_{i-1}, A_i) rezultă că orice grupare a operaţiilor din I pe posturi de lucru, satisfăcând condiţiile 1 – 4, se identifică cu un drum în graful $\mathcal{G}$ de la nodul ∅ la nodul I, numărul posturilor fiind egal cu numărul arcelor componente.

Astfel, problema echilibrării liniei este echivalentă cu problema determinării drumului cu cele mai puţine arce de la nodul ∅ la nodul I în graful $\mathcal{G}$. Ultima chestiune este un caz particular al problemei de determinare a drumului de valoare minimă între două noduri ale unui graf în care toate arcele au valoarea 1!

Exemplul 3 Un produs este realizat prin cinci operaţii A, B, C, D, E. Duratele operaţiilor şi pecedenţele directe între ele sunt date în tabelul 9.2

Tabelul 9.2

Operaţia	A	B	C	D	E
Durata (minute)	5	2	4	8	9
Operaţii direct precedente	-	-	A,B	B	C

Graful operaţiilor este vizualizat în figura 9.2. Idealele mulţimii parţial ordonate $I = \{A, B, C, D, E\}$ sunt listate în tabelul 9.3.

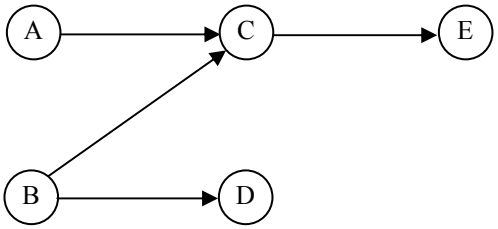


Figura 9.2

Tabelul 9.3

Nr.crt	Idealul A	d(A)
1	∅	0
2	{A }	5
3	{B }	2
4	{A,B }	7
5	{B,D }	10
6	{A,B,C }	11
7	{A,B,D }	15
8	{A,B,C,D }	19
9	{A,B,C,E }	20
10	{A,B,C,D,E }	28

Fixăm un ciclu c = 12 minute / produs ceeace corespunde unei productivități $Q = 1/c = 60/12 = 5$ produse / oră. În graful $\mathcal{G}$ ale cărui noduri sunt cele 10 ideale ale mulțimii I un arc (A,B) corespunde unei perechi de ideale A ⊂ B astfel încât $d(B) - d(A) \le 12$. Graful $\mathcal{G}$ este vizualizat în figura 9.3.

Există trei drumuri de la nodul ∅ la nodul I cu acelaşi număr minim de arce:

$$\varnothing \longrightarrow \{ A,B \} \longrightarrow \{ A,B,C,D \} \longrightarrow I$$
$$\varnothing \longrightarrow \{ A,B,C \} \longrightarrow \{ A,B,C,D \} \longrightarrow I$$
$$\varnothing \longrightarrow \{ A,B,C \} \longrightarrow \{ A,B,C,E \} \longrightarrow I$$

cărora le corespund trei modalități de grupare a operațiilor pe posturi de lucru.

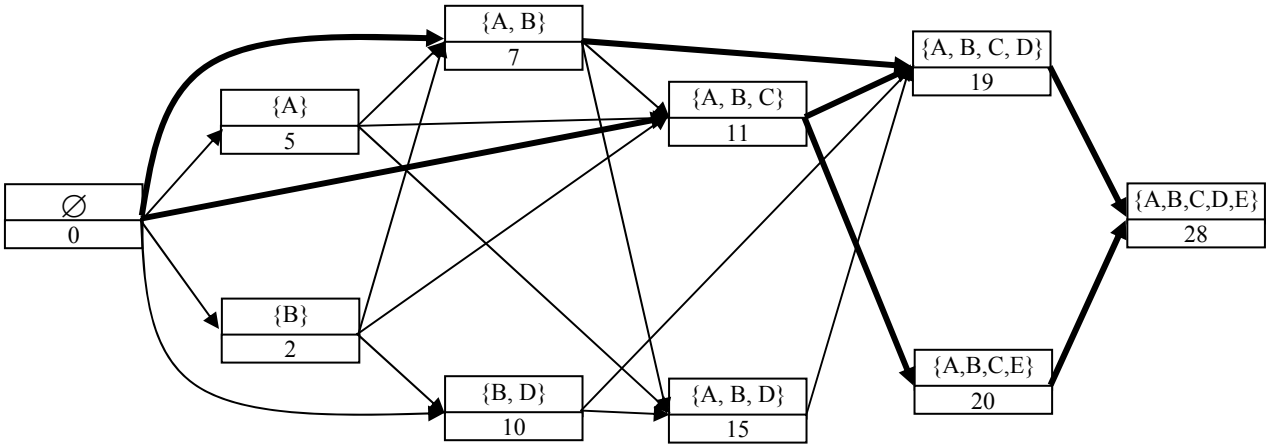


Figura 9.3

1) S_1 = { A,B } S_2 = { C,D } S_3 = { E }

Însumând duratele operațiilor din fiecare post găsim valorile 7, 12, 9 minute, care se încadrează în ciclul c = 12 minute.

2) S_1 = { A,B,C } S_2 = { D } S_3 = { E }

Suma duratelor pe fiecare post este acum de 11 , 8 , 9 minute.

3) S_1 = { A,B,C } S_2 = { E } S_3 = { D }

identică cu soluția precedentă, dacă se face abstracție de ordinea de executare a operațiilor D şi E.

Dacă ciclul era c = 15 minute (corespunzător unei productivități $Q = 60/15 = 4$ produse / oră, graful $\mathcal{G}$ "se îmbogățea" cu arcele $\varnothing \longrightarrow \{A,B,D\}$ şi $\{A,B,D\} \longrightarrow \{A,B,C,D,E\}$ iar gruparea optimă consta în două posturi de lucru:

S_1 = { A,B,D } S_2 = { C,E }

cu timpii de ocupare de 15 respectiv 13 minute.

Observație: Reducerea problemei echilibrării liniei de asamblare la o problemă de drum minim nu înseamnă câtuşi de puțin că prima este o problemă de optimizare „uşoară". Dificultatea majoră rezidă în numărul foarte mare – chiar astronomic – de ideale pe care le poate avea mulțimea parțial ordonată I din exemplele practice. Mai mult, acest număr are tendința de a creşte „exponențial" pe măsura creşterii numărului de operații.

În fapt, este dovedit că problema echilibrării liniei de asamblare este o problemă de optimizare „grea", de calibrul problemei comisvoiajorului sau a problemei stabilirii traseelor sau a croirii...

9.5 Algoritmi de rezolvare suboptimală a ALBP

Menţinem cadrul notaţional introdus în secţiunile precedente.Metodele euristice care vor fi prezentate în continuare au la bază ordonarea prealabilă a operaţiilor cu ajutorul unor **ponderi poziţionale**, compatibile cu relaţiile de precedenţă dintre operaţii. La fiecare iteraţie este **deschis** un singur post de lucru în care sunt incluse una sau mai multe operaţii încă nerepartizate şi cu cea mai "ridicată" pondere. După "umplerea" postului, acesta se **închide** şi se deschide un nou post în caz că mai sunt operaţii de repartizat. Închiderea unui post depinde de duratele operaţiilor şi de mărimea ciclului, date din start. Numărul posturilor este cel puţin egal cu:

$$\left\lceil \frac{\text{suma duratelor tuturor operatiilo r}}{\text{durata ciclului}} \right\rceil$$

Euristica A

Pentru fiecare operaţie $x \in I$ definim:**ponderea poziţională normală** ca fiind valoarea

$$w(x) = d(x) + \sum_{y \in I | x \prec y} d(y)$$

$w(x)$ este deci suma duratelor operaţiei x şi a tuturor operaţiilor pe care x le precede direct sau indirect. Rezultă imediat că dacă operaţia x precede (direct sau indirect) operaţia y atunci $w(x) > w(y)$.

Înscriem toate operaţiile într-o **listă** $\mathcal{L}$ (zisă a operaţiilor nerepartizate) în **ordinea descrescătoare a ponderilor poziţionale normale**.La fiecare iteraţie $r = 1, 2, \ldots$**se deschide** un nou post de lucru S_r şi se iniţializează **capacitatea reziduală** a acestuia:

$$\bar{c} \leftarrow c \equiv \text{ciclul de timp } \textbf{dat}$$

Pe parcurs, capacitatea reziduală va arăta diferenţa dintre ciclul de timp şi suma duratelor operaţiilor incluse în postul deschis.În momentul **închiderii** postului, capacitatea reziduală va indica **timpul de inactivitate** aferent postului închis (**timpul mort**).

După deschiderea postului S_r sunt cercetate **pe rând** operaţiile din **lista curentă** $\mathcal{L}$ **a operaţiilor nerepartizate.** Fie x operaţia ce urmează a fi cercetată. Dacă:

i) $d(x) \le \bar{c}$;

ii) toate operaţiile premergătoare lui x au fost repartizate, fie în S_r fie în alte posturi create în iteraţiile anterioare şi déjà închise, altfel spus aceste operaţii nu mai apar în lista $\mathcal{L}$, operaţia x se introduce în postul S_r după care:

- operația x se şterge din lista $\mathcal{L}$;
- se actualizează capacitatea reziduală a postului S_r : $\bar{c} \leftarrow \bar{c} - d(x)$;
- se trece la examinarea următoarei operații nerepartizate din $\mathcal{L}$ – dacă mai există!

Postul curent S_r se închide dacă:

- nu mai există operații de repartizat: $\mathcal{L} = \varnothing$;

sau dacă:

- operațiile rămase în lista $\mathcal{L}$ nu satisfac una sau alta dintre condițiile i) şi ii).

Dacă mai sunt operații nerepartizate se va deschide un nou post S_{r+1}, se reinițializează capacitatea reziduală $\bar{c} \leftarrow c$ şi se repetă instrucțiunile date mai sus.

Exemplul 4 Reluăm problema de echilibrare dată în exemplul 1.Folosim euristica descrisă pentru a grupa operațiile pe (cât mai puține) posturi de lucru, relativ la un ciclu c = 12 minute - ceeace ar corespunde la o productivitate $Q = 60/12 = 5$ produse pe oră. Suma duratelor operațiilor este 38 min, astfel că vor fi necesare cel puțin $\left\lceil \frac{38}{12} \right\rceil = 4$ posturi.

Soluție Calculăm ponderile poziționale ale celor nouă operații; de mare folos va fi graful operațiilor din figura 9.1 Astfel, operația 2 precede direct operațiile 4 şi 5 şi indirect operațiile 8 şi 9. În consecință, $w(20 = d(2) + d(4) + d(5) + d(8) + d(9) = 3 + 6 + 5 + 5 + 3 = 22$. În schimb, operația 8 este o operație finală aşa că $w(8) = d(8) = 5$

Ponderile poziționale ale operațiilor problemei sunt date în tabelul 9.4

Tabelul 9.4

Operația	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Pondere poziționlă	35	22	16	11	8	11	7	5	3

Listăm operațiile în ordinea descrescătoare a ponderilor poziționale – vezi tabelul 9.5

Tabelul 9.5

Operația	**1**	**2**	**3**	**4**	**6**	**5**	**7**	**8**	**9**
Ponderea poziționlă	35	22	16	11	11	8	7	5	3
Durata	5	3	4	6	3	5	4	5	3
Operații direct precedente	-	**1**	**1**	**2**	-	**2,3**	**3**	**4,6**	**5,6,7**

Start Lista operațiilor nerepartizate: $\mathcal{L} = \{1,2,3,4,6,5,7,8,9\}$

Iterația 1 Deschidem postul S_1 cu capacitatea reziduală $\bar{c} = 12$ min

Includem în S_1:

Operația **1** $\Rightarrow$ $\bar{c} = 12 - 5 = 7$ min

Operația **2** $\Rightarrow$ $\bar{c} = 7 - 3 = 4$ min

Operația **3** $\Rightarrow$ $\bar{c} = 4 - 4 = 0$

Operațiile 1,2,3 sunt şterse din lista operațiilor nerepartizate $\Rightarrow \mathcal{L} = \{4,6,5,7,8,9\}$ Închidem postul S_1.

Iterația 2 Deschidem postul S_2 cu capacitatea reziduală $\bar{c} = 12$ min

Includem în S_2:

Operația **4** $\Rightarrow$ $\bar{c} = 12 - 6 = 6$ min

Operația **6** $\Rightarrow$ $\bar{c} = 6 - 3 = 3$ min

Operațiile 5, 7, 8 care urmează în lista $\mathcal{L}$ nu pot fi incluse în S_2 din lipsă de timp. Operația 9 ar avea loc dar nu poate fi inclusă, deoarece operațiile 5 şi 7 ,premergătoare ei, nu au fost repartizate. Suntem nevoiți să închidem postul S_2 cu un timp de inactivitate de 3 min. Actualizăm $\mathcal{L} = \{5,7,8,9\}$

Iterația 3 Deschidem un nou post S_3 cu capacitatea reziduală $\bar{c} = 12$ min

Includem în S_3:

Operația **5** $\Rightarrow$ $\bar{c} = 12 - 5 = 7$ min

Operația **7** $\Rightarrow$ $\bar{c} = 7 - 4 = 3$ min

Operația 8 nu "încape". În schimb, poate fi inclusă

Operația **9** $\Rightarrow$ $\bar{c} = 3 - 3 = 0$

deoarece operațiile premergătoare sunt déjà repartizate, operațiile 5 şi 7 chiar în S_3 iar operația 6 în postul S_2. Închidem S_3 şi actualizăm: $\mathcal{L} = \{8\}$.

Iterația 4 A rămas o singură operație nerepartizată pentru care se creează un post separat $S_4 = \{\mathbf{8}\}$.

Stop

A rezultat gruparea:

$$S_1 = \{1,2,3\} \; ; \; S_2 = \{4,6\} \; ; \; S_3 = \{5,7,9\} \; ; \; S_4 = \{8\}$$

vizualizată în figura 9.4 Soluția găsită este chiar optimă întrucât ştiam din start că sunt necesare cel puțin patru posturi!

$$\text{Ponderea timpilor morți} = \frac{\text{suma timpilor morti la fiecare post de lucru}}{\text{ciclu} \times \text{nr. posturilor}} \cdot 100 =$$

$$= \frac{0+3+0+7}{12 \times 4} \cdot 100 = 20{,}8\%$$

Atenție: în cadrul unui post, operațiile **independente** pot fi executate în orice ordine! Este cazul operațiilor 2 şi 3 din S_1 sau 4 şi 6 din S_2 sau 5 şi 7 din S_3.

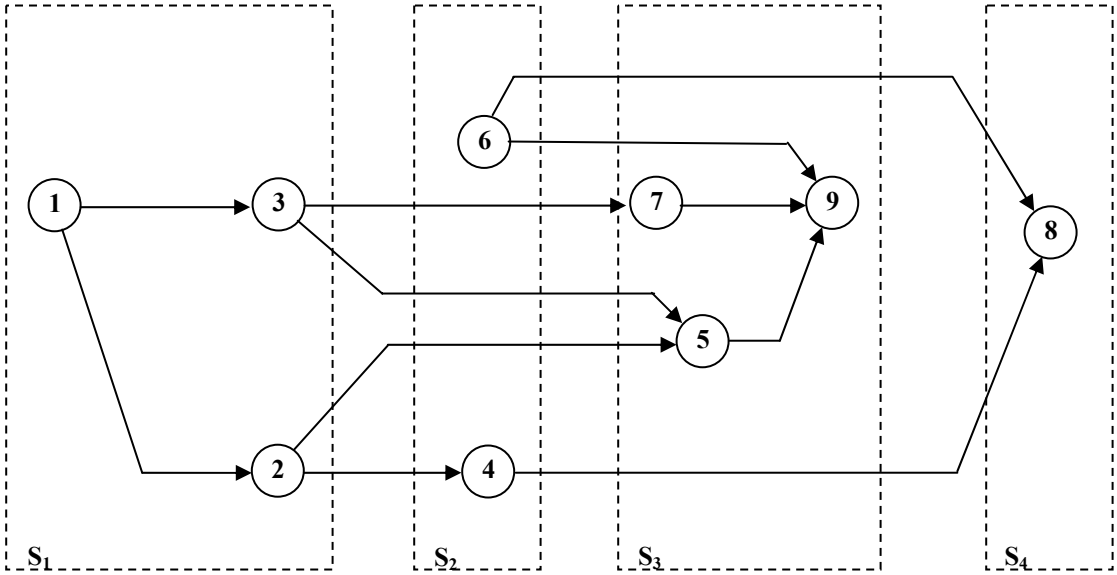


Figura 9.4

Euristica B

Pentru fiecare operație $x \in I$ definim:

- **Ponderea poziționalăinversă** ca fiind valoarea:

$$\theta(x) = d(x) + \sum_{y \in I \mid y \prec x} d(y)$$

$\theta(x)$ este suma duratelor operației x şi a tuturor operațiilor care preced – direct sau indirect – operația x.

Se vede uşor că dacă operația x precede - direct sau indirect - operația y atunci $\theta(x) < \theta(y)$.

Lista $\mathcal{L}$ a operațiilor nerepartizate se inițializează cu toate operațiile, aşezate în ordinea **crescătoare** a ponderilor poziționale inverse.

Pentru a nu ne "pierde" în amănunte notaționale, inevitabile într-un expozeu general, euristica B – de altfel foarte simplă – va fi explicată pe un exemplu numeric (suntem siguri că cititorul atent şi conştiincios se va descurca în orice altă situație asemănătoare…)

Exemplul 5 Realizarea unui produs P implică executarea a 12 operații. Duratele operațiilor şi precedențele directe dintre ele sunt date în tabelul 9.6

Tabelul 9.6

Operația	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Durata (ore)	0,2	0,4	0,7	0,1	0,3	0,11	0,32	0,6	0,27	0,38	0,5	0,12
Operații direct **precedente**	**-**	**-**	**1**	**1,2**	**2**	**3**	**3**	**3,4**	**6,7,8**	**5,8**	**9,10**	**11**

Pentru evaluarea ponderilor este necesar **graful precedențelor directe** (figura 9.5)

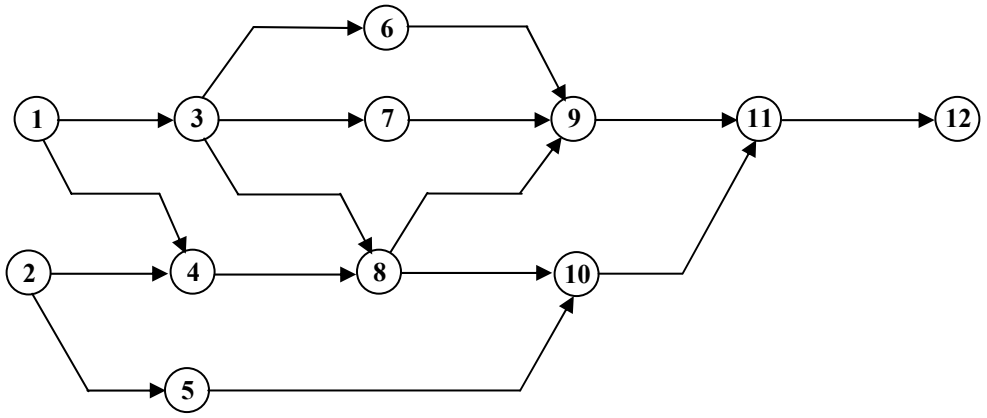


Figura 9.5

Ponderile poziționale inverse sunt indicate în tabelul 9.7:

Tabelul 9.7

Operația x	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
$\theta(x)$	0,2	0,4	0,9	0,6	0,7	1,01	1,22	2	2,1	2,68	3,88	4

Ordonarea crescătoare a ponderilor poziționale inverse conduce la lista din tabelul 9.8:

Tabelul 9.8

Operația x	**1**	**2**	**4**	**5**	**3**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
$\theta(x)$	0,2	0,4	0,6	0,7	0,9	1,01	1,22	2	2,1	2,68	3,88	4

Fixăm un ciclu $c = 1$; vor fi necesare este de cel puțin patru posturi de lucru.

Iterația 1 Deschidem postul S_1. În acest post nu pot intra operațiile 6,7,8,9,10,11 sau 12. Într-adevăr, includerea oricăreia dintre ele în S_1 implică automat includerea operațiilor care preced respectiva operație; indicatorul $\theta(x)$ arată că acest lucru este imposibil, depășindu-se durata ciclului!

Aşadar, pentru a fi incluse în postul S_1 candidează operaţiile 1,2,3,4,5. Subgraful precedenţelor directe între aceste operaţii este dat în figura 9.6 **Selectarea operaţiilor care vor face parte din S_1 va fi realizată cu un program bivalent descris în continuare.** Asociem operaţiilor candidate variabilele:

$$x_i = \begin{cases} 1 \text{ daca operatia } i \text{ este inclusa in postul } S_1 \\ 0 \text{ in caz contrar} \end{cases} \quad i = 1,...,5$$

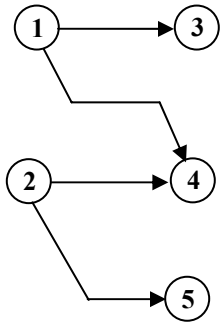


Figura 9.6

Rezultă restricţiile:

$$\begin{cases} 0,2x_1 + 0,4x_2 + 0,7x_3 + 0,1x_4 + 0,3x_5 \le 1 & (1) \\ x_1 \ge x_3 \\ x_1 \ge x_4 & (2) \\ x_2 \ge x_4 \\ x_2 \ge x_5 \end{cases}$$

Restricţia (1) arată că suma duratelor operaţiilor care vor face parte din postul S_1 nu trebuie să depăşească durata ciclului (=1). Restricţiile (2) formalizează cerinţa ca, o dată cu o anumită operaţie, în postul S_1 să fie incluse şi operaţiile care o preced!

Este evident faptul că sistemul (1) – (2) are destule soluţii $\bar{x} = (\bar{x}_j)$ cu componente 0,1.Dintre acestea o vom alege pe aceea care „încarcă cel mai bine" postul în sensul maximizării sumei duratelor operaţiilor incluse! În consecinţă ataşăm sistemului (1) – (2) funcţia obiectiv:

$$f(x) = 0,2x_1 + 0,4x_2 + 0,7x_3 + 0,1x_4 + 0,3x_5 \quad \to \max \qquad (3)$$

Programul bivalent (1) – (3) are soluţia optimă:

$$x_1 = 1 \quad x_2 = 1 \quad x_3 = 0 \quad x_4 = 1 \quad x_5 = 1 \qquad f(x) = 1$$

Postul S_1 se va compune din operaţiile **1**,**2**,**4** şi **5.**Din graful precedenţelor directe eliminăm operaţiile repartizate; Graful din figura 9.5 se reduce la cel din figura 9.7 şi pe baza lui actualizăm ponderile poziţionale inverse ale operaţiilor rămase – vezi tabelul 9.9

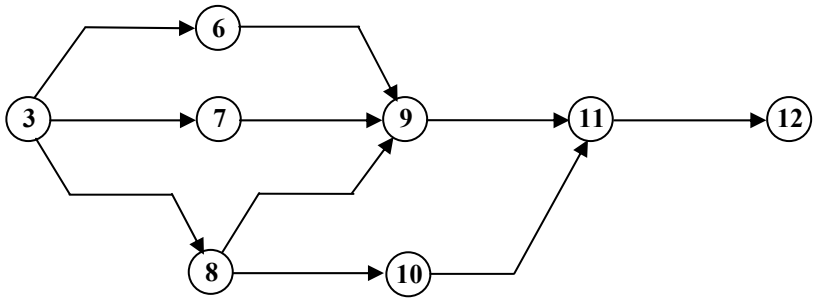


Figura 9.7

Tabelul 9.9

Operația x	**3**	**6**	**7**	**8**	**10**	**9**	**11**	**12**
$\theta(x)$	0,7	0,81	1,02	1,3	1,68	2	2,88	3

Iterația 2 Fără a mai scrie modelul bivalent analog se vede direct că cea mai bună încărcare pentru al doilea post de lucru este

$$S_2 = \{\mathbf{3}\ ,\ \mathbf{6}\}$$

care ocupă 0.81 din durata ciclului c = 1. Eliminând operațiile 3 şi 6 precedențele directe dintre operațiile rămase sunt relevate de graful din figura 9.8

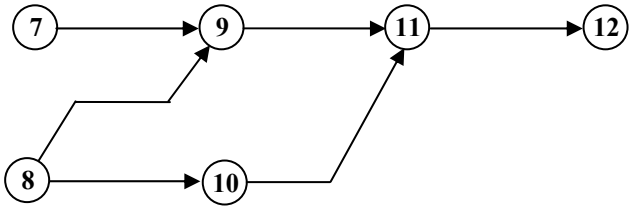


Figura 9.8

În tabelul 9.10 sunt actualizate ponderile poziționale inverse ale operațiilor rămase.

Tabelul 9.10

Operația x	**7**	**8**	**10**	**9**	**11**	**12**
$\theta(x)$	0,32	0,6	0,98	1,19	2,07	2,19

În continuare se obțin încă trei posturi: S_3 = {**7** , **8**} , S_4 = {**9** , **10**} , S_5 = {**11** , **12**} care utilizează 0,92 0,65 respectiv 0,62 din durata ciclului dat.

Din punctul de vedere al încărcării posturilor, soluția găsită nu este satisfăcătoare: în timp ce posturile 1 şi 3 utilizează aproape în întregime timpul unui ciclu, la posturile 4 şi 5 mai bine de o treime din acest timp nu este folosit! Ponderea timpilor morți este de 20%

O încărcare mai uniformă se obține mărind durata ciclului (creşterea trebuie să fie destul de mică pentru a nu afecta"grav" productivitatea!) Astfel pentru un ciclu cu durata c = 1,02 rezultă patru posturi cu o pondere a timpilor de inactivitate de numai 2%!! – vezi tabelul 9.11

Tabelul 9.11

Postul de lucru	S_1	S_2	S_3	S_4
Operațiile repartizate	**1,3,6**	**2,5,7**	**4,8,9**	**10,11,12**
Timp de inactivitate	0,01	0	0,05	0,02

Această soluție este chiar optimă (de ce ?)

9.6 Ilustrări numerice

1. Firma CRAI ION este pe cale de a instala o nouă linie de fabricație pentru ascuțitori de creioane. Dvs. vi se cere să echilibrați linia cunoscând duratele operațiilor şi relațiile de precedență directă:

Tabelul 9.12

Operația	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**
Durata (secunde)	12	24	18	78	6	48	18	72
Operații imediat **succesoare**	**b**	**d**	**d**	**g**	**f**	**g**	**h**	**-**

i) Desenați graful precedențelor directe dintre operații;
ii) Repartizați operațiile pe posturi de lucru în ordinea descrescătoare a ponderilor poziționale (normale). Se va lua ciclul de timp cel mai mic posibil.

Calculați ponderea timpilor morți şi productivitatea liniei corespunzătoare ciclului ales.

iii) Redefinim ponderea poziționlă normală a unei operații ca fiind numărul operațiilor care o succed direct sau indirect. Cum s-ar grupa operațiile pe posturi de lucru la acelaşi ciclu de timp?
iv) Care este cel mai mic ciclu de timp care ar permite organizarea a numai două posturi de lucru? Faceți repartizarea operațiilor şi determinați productivitatea liniei precum şi ponderea timpilor morți.

Soluție i) Graful precedențelor directe este dat în figura 9.9; au fost puse în evidență şi duratele operațiilor pentru facilitarea calculului ponderilor poziționale de orice tip!

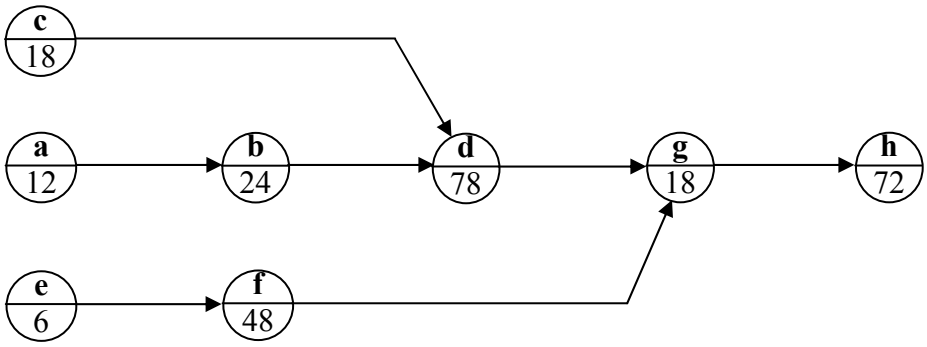


Figura 9.9

ii) Ponderile poziționale normale sunt afişate în tabelul 9.13.

Tabelul 9.13

Operația x	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**
Ponderea pozițională normală w(x)	204	192	186	168	144	138	90	72

Deoarece ponderile calculate sunt descrescătoare, repartizarea operațiilor se va face în ordinea dată în enunț.
Cel mai mic ciclu posibil este durata celei mai lungi operații: c = 78 sec. Suma duratelor tuturor operațiilor este de 276 sec astfel că, pentru ciclul ales vor fi necesare cel puțin $\left\lceil \frac{276}{78} \right\rceil = 4$ posturi.

Aplicăm euristica A.(recomandăm cititorului să revadă exemplul 4 din secțiunea 11.5)

Start Deschidem postul S_1. Inițializăm capacitatea reziduală $\bar{c} = c = 78$ sec.

Iterația 1

- în postul deschis, includem, pe rând, operațiile:
  - **a** $\rightarrow$ $\bar{c} = 78 - 12 = 66$ sec;
  - **b** $\rightarrow$ $\bar{c} = 66 - 24 = 42$ sec;
  - **c** $\rightarrow$ $\bar{c} = 42 - 18 = 24$ sec.
- următoarea operație din listă,**d**, are o durată ce depăşeşte capacitatea reziduală curentă.
- putem include în S_1 operația **e** $\rightarrow$ $\bar{c} = 24 - 6 = 18$ sec.
- nici o altă operație nu mai poate fi inclusă în S_1; singura care ar avea „loc" este **g** dar predecesoarea ei **d** nu a fost încă repartizată.
- închidem postul S_1.
- deschidem postul S_2 cu capacitatea reziduală de start $\bar{c} = 78$ sec.

Iterația 2

- includem în S_2 operația **d** cu observația că toate operațiile care o preced au fost deja repartizate $\rightarrow$ capacitatea reziduală devine $\overline{c} = 78 - 78 = 0$.
- postul S_2 este „plin" şi ca urmare se închide.
- deschidem postul S_3 şi reactualizăm $\overline{c} = 78$ sec.

Iterația 3

- în S_3 includem operațiile:

f (operația precedentă **e** este repartizată în S_1) $\rightarrow \overline{c} = 78 - 48 = 30$ sec;

g (în acest moment toate operațiile care o preced – direct sau indirect – sunt repartizate) $\rightarrow \overline{c} = 30 - 18 = 12$ sec.

- închidem postul S_3 deoarece în acest post nu mai „începe" nimic.
- deschidem postul S_4, cu $\overline{c} = 78$ sec, întrucât lista operațiilor nerepartizate este încă nevidă.

Iterația 4

- includem în S_4 operația rămasă **h** $\rightarrow \overline{c} = 78 - 72 = 6$ sec.
- închidem postul S_4 şi ne oprim deoarece toate operațiile sunt repartizate.

A rezultat următoarea grupare a operațiilor pe posturi de lucru (tabelul 9.14):

Tabelul 9.14

Post de lucru	S_1	S_2	S_3	S_4
Operații incluse	**a , b , c , e**	**d**	**f , g**	**h**
Timp ocupat	60	78	66	72
Timp mort	18	0	12	6

Soluția găsită este chiar optimă pentru că numărul posturilor de lucru nu poate fi mai mic de patru. Ponderea timpilor morți:

$$\frac{\text{suma timpilor de inactivitate la nivel de post}}{\text{nr. posturilor} \times \text{ciclul de timp}} \cdot 100 = \frac{18 + 0 + 12 + 6}{4 \cdot 78} \cdot 100 = 11{,}54\%$$

Productivitatea liniei, corespunzătoare ciclului ales şi la o zi de lucru de 7ore = 420min = 25200sec este de $\frac{25200}{78} \cong 320$ produse finite.

Repartizarea operațiilor pe posturi este vizualizată în figura 9.10

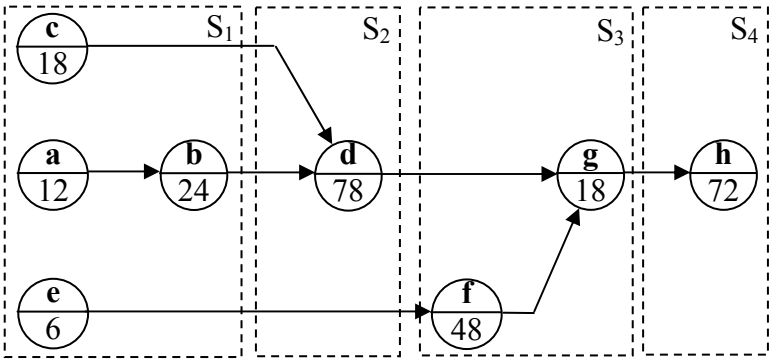


Figura 9.10

iii) Pentru fiecare operație determinăm numărul de operații succesoare; pentru aceasta folosim din nou graful precedențelor directe din figura 9.9 Rezultatele apar în tabelul 9.15

Tabelul 9.15

Operația	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**
Numărul operațiilor **succesoare**	4	3	3	2	3	2	1	-

Ordinea în care operațiile vor fi repartizate este: **a, b, c, e, d, f, g, h.** Se obține aceeaşi grupare pe posturi.

iv) Suma duratelor operațiilor este de 276 sec. Pentru a avea numai două posturi de lucru ar trebui ca ciclul de timp să fie $c = \frac{276}{2} = 138$ sec.Operațiile s-ar grupa astfel (tabelul 9.16):

Tabelul 9.16

Post de lucru	S_1	S_2
Operații incluse	**a , b , c , d , e**	**f , g , h**
Timp ocupat	138	138
Timp mort	0	0

Productivitatea liniei – la nivel de zi de lucru $\frac{25200}{138} \cong 180$ produse finite.

2. Un produs de larg consum este realizat în serie pe o linie de fabicație. Duratele operațiilor (în minute) şi precedențele dintre ele sunt date în tabelul 9.17

Tabelul 9.17

Operația	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**k**
Durata (minute)	5	3	4	5	4	5	1	4	6
Operații direct **precedente**	**-**	**a**	**a**	**b,c**	**d**	**d**	**d**	**e**	**f,g**

i) Desenați graful precedențelor directe dintre operații.
ii) Calculați ponderea poziționlă inversă a operațiilor şi grupați operațiile pe posturi de lucru (cu algoritmul B) considerând un ciclu de fabricație c = 13 minute. Calculați ponderea timpilor morți şi productivitatea liniei la un schimb de opt ore.
iii) Reluați problema echilibrării liniei grupând operațiile în funcție de ponderea poziționlă normală (acelaşi ciclu). Comparați rezultatele.

Soluție i) Graful precedențelor directe este dat în figura 9.11

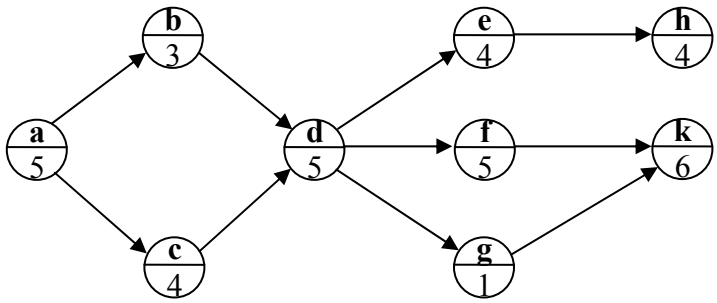


Figura 9.11

Duratele celor nouă operații însumează 37 minute astfel că la un ciclu de 13 minute sunt necesare cel puțin $\left\lceil \frac{37}{13} \right\rceil = 3$ posturi de lucru.

ii) Ponderile poziționale inverse, în ordine crescătoare, sunt date în tabelul 9.18 Reamintim (vezi şi exemplul 5 din secțiunea 11.5) că ponderea poziționlă inversă a unei operații x rezultă din însumarea duratelor operației x şi a tuturor operațiilor care o preced.

Tabelul 9.18

Operația x	**a**	**b**	**c**	**d**	**g**	**e**	**f**	**h**	**k**
Ponderea poziționlă inversă $\theta(x)$	5	8	9	17	18	21	22	25	29

Iterația 1 Este uşor de văzut că primul post S_1 este format din operațiile **a** , **b** şi **c** ale căror durate „ocupă" 12 minute din cele 13 ale ciclului. Eliminăm operațiile **a, b, c** şi recalculăm ponderile poziționale inverse ale operațiilor rămase – vezi tabelul 9.19

Tabelul 9.19

Operația x	**d**	**g**	**e**	**f**	**h**	**k**
Ponderea poziţională inversă $\theta(x)$	5	6	9	10	13	17

Iterația 2 Pentru ocuparea postului S_2 candidează operațiile **d, g, e, f** şi **h** ale căror ponderi inverse nu depăşesc ciclul dat.Soluția nu mai este imediată ca în iterația 1; ea va rezulta din rezolvarea unui program bivalent de maximizare a „timpului ocupat".
Introducem variabilele booleene x_d, x_g, x_e, x_f, x_h cu valoarea 1 dacă operația corespunzătoare este inclusă în postul S_2 şi 0 în caz contrar. Inecuația:

$$5x_d + 1x_g + 4x_e + 5x_f + 4x_h \le 13 \qquad (1)$$

formalizează cerința ca duratele operațiilor incluse în post să nu depăşească – în sumă – ciclul dat. Dacă x şi y desemnează două din cele cinci operații candidate şi x precede y , includerea operației y în S_2 implică obligatoriu includerea operației x în S_2. Formalizarea acestei condiții conduce la relațiile:

$$\begin{aligned} &x_d \ge x_e \ge x_h \\ &x_d \ge x_f \\ &x_d \ge x_g \end{aligned} \qquad (2)$$

Timpul ocupat de operațiile incluse în S_2 este dat de expresia:

$$T = 5x_d + 1x_g + 4x_e + 5x_f + 4x_h \to \max \qquad (3)$$

Programului bivalent (1) – (3) are soluția optimă:

$$x_d = 1\,,\ x_g = 0\,,\ x_e = 1\,,\ x_f = 0\,,\ x_h = 1 \quad T = 13$$

şi ca urmare, postul S_2 are alcătuirea:

$$S_2 = \{\mathbf{d}\,,\ \mathbf{e}\,,\ \mathbf{h}\}$$

Iterația 3 Este evident faptul că operațiile rămase „încap" într-un singur post.

$$S_3 = \{\mathbf{f}\,,\ \mathbf{g}\,,\ \mathbf{k}\}$$

A rezultat următoarea grupare a operațiilor pe posturi de lucru (tabelul 9.20):

Tabelul 9.20

Post de lucru	S_1	S_2	S_3
Operații incluse	**a , b , c**	**d,e,h**	**f , g,k**
Timp ocupat	12	13	12
Timp mort	1	0	1

cu o pondere a timpilor de inactivitate de numai $\frac{1+0+1}{3\cdot 13}\cdot 100 = 5{,}13\%$. Productivitatea liniei, corespunzătoare ciclului dat ar fi de $\frac{8\cdot 60}{13} \cong 37$ produse finite pe schimb. Soluția găsită este optimă!

iii) Lăsăm în seama cititorului aplicarea euristicii A. Va obține o soluție mai proastă cu patru posturi de lucru!

Probleme propuse

Se propun trei probleme cu următorul enunț comun.
Se proiectează linia de fabricație a unui produs. Sunt stabilite operațiile de prelucrare, duratele şi precedențele dintre operații. Obiectivul este echilibrarea liniei adică gruparea operațiilor pe cât mai puține posturi de lucru în funcție de un ciclu de timp prestabilit.

i) Desenați graful precedențelor directe;
ii) Determinați numărul minim de posturi necesare, raportând suma duratelor operațiilor la durata ciclului;
iii) Grupați operațiile folosind, după dorință, una sau alta dintre euristicile A, B. Calculați ponderea timpilor morți la nivel de post. Dacă este cazul studiați problema modificării ciclului de fabricație propus în scopul obținerii unor rezultate mai bune atât în ceeace priveşte numărul posturilor cât şi încărcarea acestora.

1.

Tabelul 9.21

Operația	**a**	**b**	**c**	**d**	**e**	**f**	**g**
Durata (minute)	1	5	4	3	5	5	6
Operații direct precedente	**-**	**a**	**b**	**a**	**b**	**e**	**d**

Durata ciclului c = 10 min.

2.

Tabelul 9.22

Operația	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**
Durata (minute)	11	17	9	5	8	12	10	3
Operații direct precedente	**-**	**a**	**b**	**b**	**c**	**c,d**	**e**	**f**

Durata ciclului c = 28 min; c = 27 min ; c = 25 min.

3.

Tabelul 9.23

Operația	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**k**	**l**	**m**
Durata (minute)	6	2	5	7	1	2	3	6	5	5	4
Operații direct precedente	**-**	**a**	**a**	**a**	**a**	**b**	**c,d,e**	**f**	**g**	**h**	**k,l**

Durata ciclului c = 12 min

Unitatea de învățare 10

INTRODUCERE ÎN OPTIMIZAREA COMBINATORIALĂ
Introducere în studiul problemei ordonanțării (Scheduling Problem)

Cuprins

10.1 O clasificare a problemelor de ordonanțare
10.2 Probleme de ordonanțare pe un utilaj
10.3 Ordonanțarea în flux (Flow Shop)
10.3.1 Ordonanțarea în flux pe două utilaje
10.3.2 Ordonanțarea în flux pe $m \geq 3$ utilaje
10.4 Ordonanțarea pe mai multe utilaje, cazul general (Job Shop)
10.4.1 Modelarea problemei
10.4.2 O rezolvare suboptimală
10.5 Ilustrări numerice

Probleme propuse

Obiectivul unității de învățare 10

În cadrul unui proces tehnologic, asupra unor obiecte denumite generic **repere** se execută o serie de **operații** în vederea obținerii unor produse (acestea pot fi produse finite sau semifabricate mai complexe care devin repere în alte procese tehnologice)

Operațiile se execută pe **utilaje** specializate şi au o **durată** bine stabilită. În principiu, un utilaj poate executa operații aferente mai multor repere dar, la un moment dat, nu poate face decât o singură operație.

În acest cadru, **problema ordonanțării** înseamnă stabilirea **ordinii** şi a **termenelor** la care se vor executa diferitele operații pe fiecare utilaj în parte, astfel încât un anumit criteriu de performanță să fie îndeplinit.

Deja anunțată în unitatea de învățare 4, problema ordonanțării este una din cele mai grele probleme de optimizare combinatorială

În această secțiune sunt descrise câteva probleme reprezentative de ordonanțare împreună cu unele metode de rezolvare – exactă sau suboptimală.

10.1 O clasificare a problemelor de ordonanțare

Extrem de diverse, problemele de ordonanțare se clasifică după o mulțime de factori dintre care amintim:

- După **numărul utilajelor**, avem probleme de ordonanțare pe un utilaj sau pe mai multe;

- După **ordinea în care se execută operațiile** unui reper avem:

  - Probleme în care ordinea de execuție a operațiilor este neesențială (**open shop**)
  - Probleme în care ordinea de parcurgere a utilajelor este aceeaşi pentru toate reperele (**flow shop**)
  - Probleme în care fiecare reper parcurge utilajele într-o anumită ordine (**job shop**)

- În problemele reale de ordonanțare nu se lucrează cu repere indidualizate ci cu **loturi** de repere **identice**. Uneori se întâmplă ca după executarea unei operații pe o parte a unui lot de repere, utilajul pe care se face operația respectivă să fie eliberat şi pregătit în vederea executării unei operații pe un alt lot de repere, restul reperelor din lotul anterior urmând a fi prelucrat mai târziu. Deoarece în modelare un lot de repere identice este asimilat unui singur reper (bineînțeles, cu durata de execuție a unei operații egală cu suma duratelor respectivei operații pe fiecare reper individual în parte…) situația descrisă este echivalentă cu **întreruperea** unei operații şi reluarea ei la un moment ulterior.
Ca urmare, avem probleme în care este permisă întreruperea operațiilor dar şi probleme în care o operație, o dată începută, este continuată până la terminarea ei.

- De regulă, fiecare reper (lot de repere identice) are un **termen de predare** (livrare) dată la care toate operațiile aferente reperului trebuie să fie terminate! Un program de ordonanțare se va considera **admisibil** daă toate reperele vor fi terminate cel mai târziu la termenele de predare.
Adesea, pentru a fi siguri că termenele de predare vor fi respectate, planificatorii fixează pentru fiecare reper în parte un termen "intern" de predare care devansează mai mult sau mai puțin termenele de predare propriu zise. Aceste termene se numesc **termene impuse** şi un program de ordonanțare va fi cu atât mai bun cu cât **întârzierile** față de termenele impuse vor fi mai mici.

- De obicei, după terminarea unei operații, utilajul prelucrător trebuie reglat (pregătit) pentru executarea unei operații aferente altui reper. Timpul de pregătire depinde atât de operația executată cât şi de viitoarea operație. Dacă aceşti timpi au valori apreciabile, ei trebuie luați în considerare.

- Un factor important este **criteriul de apreciere** a diferitelor programe de ordonanțare. Funcție de situația concretă şi de factorii amintiți mai sus (dar şi de alții…), într-o problemă de ordonanțare se poate urmări:

  - Minimizarea termenului de terminare a tuturor reperelor;
  - Minimizarea sumei întârzierilor față de termenele impuse;
  - Minimizarea celei mai mari întârzieri față de termenele impuse;
  - Minimizarea numărului reperelor întârziate față de termenele impuse;
  - Minimizarea sumei timpilor de pregătire etc.

Este clar că din combinarea factorilor mai sus amintiți şi a criteriilor de performanță adecvate rezultă o mare varietate de probleme de ordonanțare. Cercetările în domeniu – unele de ordin exclusiv "bibliografic" – au condus la identificarea a peste 3000 de probleme diferite!

În marea lor majoritate, problemele de ordonanțare sunt dificile în sensul că soluția optimă este foarte greu de găsit dacă nu chiar imposibil de găsit în timp rezonabil chiar şi pentru probleme de gabarit moderat. Aceste este şi motivul pentru care, în asemenea situații, se încearcă o rezolvare "aproximativă" (suboptimală) bazată pe metode euristice.

10.2 Probleme de ordonanțare pe un utilaj

În general, ordonanțarea pe un singur utilaj este o problemă **uşoară.** Iată câteva cazuri, fără pretenția epuizării subiectului.

Exemplul 1. Reperele 1,2,…,*n* se prelucrează pe acelaşi utilaj. Pentru reperul *j* se cunosc:

- durata p_j a operației de prelucrare;
- termenul **impus** d_j.

În ce ordine vor fi lansate reperele în execuție pentru ca întârzierea cea mai mare față de termenele impuse să fie minimă?
Este un caz **simplu**, care se rezolvă cu următoarea regulă:

Are prioritate la programare reperul cu cel mai mic termen impus

Pentru ilustrare se consideră datele din tabelul 10.1

Tabelul 10.1

Reper	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Durata p_j	5	8	3	10	9	4	8
Termen impus d_j	36	18	49	26	32	40	30

Notă: termenele impuse sunt stabilite în raport cu un moment de referință "0" cu semnificația de începere a execuției lotului de repere. Dacă unitatea de timp este **ziua** atunci reperul 3 ar trebui terminat la 49 de zile de la startul execuției lotului. **Ar fi de dorit** ca terminarea acestui reper să nu depăşească acest termen, sau, dacă nu se poate, **întârzierea să fie cât mai mică**, deoarece după executare mai sunt necesare operații de verificare şi testare înainte de predarea lui ca produs finit.

Soluția optimă – afişată în tabelul 10.2 - corespunde **ordonării crescătoare** a termenelor impuse:

Tabelul 10.2

Termen impus d_j	18	26	30	32	36	40	49
Reper	**2**	**4**	**7**	**5**	**1**	**6**	**3**
Durata p_j	8	10	8	9	5	4	3
Termen terminare	8	18	26	35	40	44	47
Întârzieri	-	-	-	3	4	4	-

Reperele vor fi procesate în ordinea 2,4,7,5,1,6,3 iar întârzierea maximă va fi de 4 unități de timp.

Observație: Rezolvarea prin **enumerare completă** ar fi necesitat inspectarea a 5040 soluții, corespunzătoare celor 7! = 5040 posibilități de ordonare a reperelor!

Exemplul 2. Reperele 1,2,…,*n* se prelucrează pe acelaşi utilaj. Pentru reperul *j* se cunosc:

- durata p_j a operației de prelucrare;
- termenul **de predare** $\overline{d}_j$.

În ce ordine vor fi lansate reperele în execuție astfel încât **suma** termenelor de terminare să fie minimă? Şi aceasta este o problemă de optimizare combinatorială **uşoară**, rezolvabilă prin următorul algoritm.

Start Se inițializează lista S a reperelor de programat: $S = \{1,2,\cdots,n\}$

Pasul 1 Se determină mulțimea:

$$C = \left\{ k \in S \;\middle|\; \overline{d}_k \geq \sum_{j \in S} p_j \right\}$$

Dacă C = ∅ **Stop** : problema nu are soluție în sensul că există cel puțin un reper ce nu poate fi terminat la termenul de predare stabilit. Dacă C ≠ ∅ se trece la:

Pasul 2 Se determină reperul $k^* \in C$ cu cea mai mare durată de prelucrare: $p_{k^*} = \max\{p_j \;,\; j \in C\}$ Se programează reperul k^* **la sfârşit** dar **înaintea** reperelor déjà programate la sfârşit. Se **actualizează** lista reperelor de programat:

$$S \leftarrow S \setminus \{k^*\}$$

Pasul 3 Dacă S = ∅ **Stop**: au fost programate toate reperele în raport cu criteriul de performanță stabilit. În caz contrar, se revine la pasul 1 în cadrul unei noi iterații.

Notă: Ca şi în exemplul precedent, termenele de predare $\bar{d}_j$ se raportează la un moment de referință 0 care semnifică începerea procesului de prelucrare a reperelor. Spre deosebire de termenele impuse, termenele de predare nu pot fi depăşite şi de aceea este posibil ca problema să nu aibe soluție.

Pentru ilustrare se consideră datele din tabelul 10.3

Tabelul 10.3

Reper	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Durata p_j	10	7	4	8	5	2	9	5
Termen de predare $\bar{d}_j$	35	50	61	59	15	55	19	28

Start: S = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 }

Iterația 1 $\sum_{j\in S} p_j = 50 \Rightarrow$ C = { 2 , 3 , 4 , 6 }

Duratele prelucrării reperelor din C sunt 7, 4, 8 respectiv 2 unități de timp, astfel că:

$$p_{k^*} = \max\{p_2 = 7, p_3 = 4, p_4 = 8, p_6 = 2\} = p_4 \Rightarrow k^* = 4$$

Se programează reperul 4 la sfârşitul listei de lansare în execuție: | | | | | | | | **4** |
Actualizăm: S = { 1 , 2 , 3 , 5 , 6 , 7 , 8 }

Iterația 2 $\sum_{j\in S} p_j = 50 - 8 = 42 \Rightarrow$ C = { 2 , 3 , 6 } $\Rightarrow$ $k^* = 2 \Rightarrow$ | | | | | | | **2** | **4** |

Actualizăm: S = { 1 , 3 , 5 , 6 , 7 , 8 }

Iterația 3 $\sum_{j\in S} p_j = 42 - 7 = 35 \Rightarrow$ C = { 1 , 3 , 6 } $\Rightarrow$ $k^* = 1 \Rightarrow$ | | | | | | **1** | **2** | **4** |

Actualizăm: S = { 3 , 5 , 6 , 7 , 8 }

Iterația 4 $\sum_{j\in S} p_j = 35 - 10 = 25 \Rightarrow$ C = { 3 , 6 , 8 } $\Rightarrow$ $k^* = 8 \Rightarrow$ | | | | | **8** | **1** | **2** | **4** |

Actualizăm: S = { 3 , 5 , 6 , 7 ,}

Iterația 5 $\sum_{j\in S} p_j = 25 - 5 = 20 \Rightarrow$ C = { 3 , 6} $\Rightarrow$ $k^* = 3 \Rightarrow$ | | | **3** | **8** | **1** | **2** | **4** |

Actualizăm: S = {5 , 6 , 7 }

Iterația 6 $\sum_{j\in S} p_j = 20 - 4 = 16 \Rightarrow$ C = {6 , 7 } $\Rightarrow$ $k^* = 7 \Rightarrow$ | | **7** | **3** | **8** | **1** | **2** | **4** |

Actualizăm: S = {5 , 6 }

Iterația 7 $\sum_{j\in S} p_j = 16-9 = 7 \Rightarrow$ C = {5 , 6 } $\Rightarrow \quad k^* = 5 \Rightarrow$

6	**5**	**7**	**3**	**8**	**1**	**2**	**4**

Rezultatele programării sunt date în tabelul 10.4; comparând momentele de terminare ale operațiilor cu termenele de predare se constată că acestea din urmă sunt respectate

Tabelul 10.4

Reper	**6**	**5**	**7**	**3**	**8**	**1**	**2**	**4**
Durata p_j	2	5	9	4	5	10	7	8
Momentul începerii operației	0	2	7	16	20	25	35	42
Momentul terminării operației	**2**	**7**	**16**	**20**	**25**	**35**	**42**	**50**
Termen de predare $\overline{d}_j$	**55**	**15**	**19**	**61**	**28**	**35**	**50**	**59**

10.3 Ordonanțarea în flux

În general, ordonanțarea pe mai multe utilaje este o problemă de optimizare combinatorială foarte grea.

În cazul ordonanțării în flux, reperele parcurg utilajele (în vederea procesării) în aceeaşi ordine.

Să considerăm *m* repere care se prelucrează **în flux** pe utilajele U_1 , U_2 , ... , U_n : întâi pe U_1, apoi pe U_2 ş.a.m.d.,ultima procesare a fiecărui reper având loc pe utilajul U_n. Există deci *m*! moduri diferite de repartizare a celor *m* repere în **pozițiile** 1 , 2 , ... *m* ale **listei de lansare în execuție**.
În continuare, **fixăm o ordine de lansare** în execuție a reperelor. Din acest moment, un reper nu va mai fi identificat prin codul său ci prin **poziția** pe care o ocupă în ordinea fixată: astfel, dacă reperul cu codul *x* ocupă poziția *i* în ordinea fixată,nu ne vom mai referi la el ca fiind „reperul *x*" ci „reperul din poziția *i*". Fie d_{ij} durata procesării reperului din poziția *i* pe utilajul U_j.

În cele ce urmează vom da o procedură algoritmică simplă de calcul a duratei prelucrării tuturor reperelor, bineînțeles funcție de ordinea de lansare în execuție prestabilită.

În raport cu un moment de referință 0 care marchează începutul procesului de prelucrare a reperelor, vom nota cu t_{ij} momentul **terminării** procesării reperului **din poziția** *i* a listei de lansare în execuție, pe utilajul U_j. Este clar că termenul t_{mn} la care ultimul reper din listă îşi termină prelucrarea pe ultimul utilaj va semnifica şi durata întregului proces de prelucrare.

În evaluarea acestor termene, două situații sunt posibile:

- Procesarea reperului din poziția „curentă" *i* pe utilajul U_{j-1} se termină **mai târziu** decât procesarea reperului din poziția anterioară *i* – 1 pe utilajul „curent" U_j – vezi figura 10.1

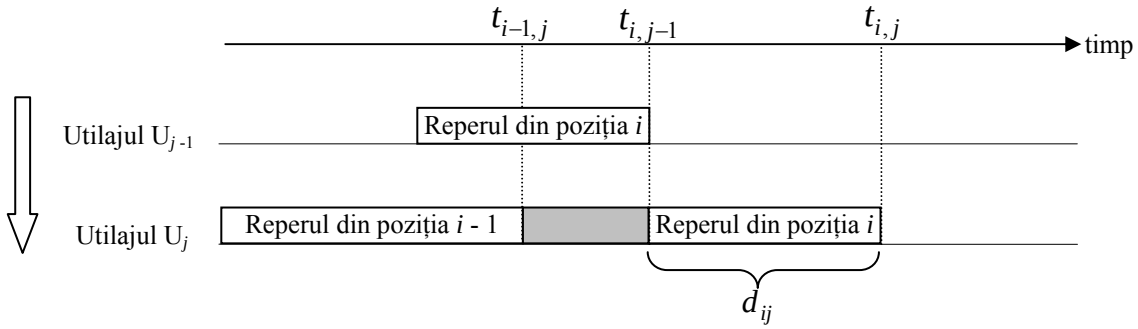


Figura 10.1

În această situație, utilajul U_j va sta **în aşteptare** în intervalul de timp $\lfloor t_{i-1,j}\ ,\ t_{i,j-1} \rfloor$ de unde:

$$t_{ij} = t_{i,j-1} + d_{ij} \qquad (1)$$

- Procesarea reperului din poziția *i* pe utilajul U_{j-1} se termină **mai devreme** decât procesarea reperului din poziția anterioară *i* – 1 pe utilajul U*j* – vezi figura 10. 2

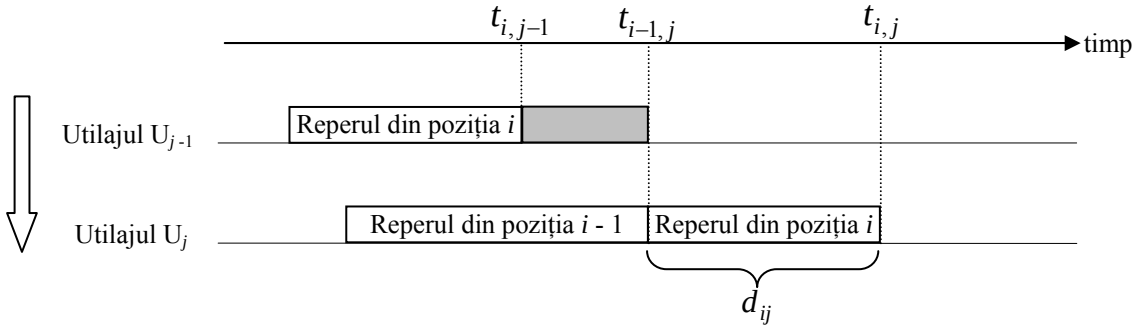


Figura 10.2

De această dată, reperul din poziția *i* va sta **în aşteptare** în intervalul de timp $\lfloor t_{i,j-1}\ ,\ t_{i-1,j} \rfloor$ şi de aici:

$$t_{ij} = t_{i-1,j} + d_{ij} \qquad (2)$$

Relațiile (1) şi (2) se reunesc în formula compactă:

$$t_{ij} = \max\{t_{i-1,j}\ ;\ t_{i,j-1}\} + d_{ij} \qquad (3)$$

în care $t_{0j} = t_{i0} = 0 \quad i = 1, \cdots, m \quad j = 1, \cdots, n$.

Exemplul 3 Reperele *a, b, c, d* se prelucrează **în flux** pe utilajele U_1 , U_2 , U_3 , U_4 , U_5. Duratele de prelucrare (în minute) sunt date în tabelul 10.5

Tabelul 10.5

Reper	Durata de prelucrare (minute)				
	U_1	U_2	U_3	U_4	U_5
a	3	4	3	7	2
b	3	7	2	8	5
c	1	2	4	3	7
d	4	3	7	2	8

Vom determina, pe baza relației (3), durata procesului de prelucrare a întregului lot în cazul în care reperele se lansează în execuție în ordinea:

i) *a , b , c , d* ii) *b , d , c , a*

Calculele sunt sistematizate în tabelele 10.6i) şi 10.6ii). Cele două săgeți arată sensul de efectuare a calculelor: pe prima linie, apoi pe a doua ş.a.m.d. iar pe fiecare linie de la stânga la dreapta.

Tabelul 10.6 i)

Reper	Poziția în lista de lansare	Termene t_{ij}				
		U_1	U_2	U_3	U_4	U_5
a	1	3	7	10	17	19
b	2	6	14	16	25	30
c	3	7	16	20	28	37
d	4	11	19	27	30	**45**

Tabelul 10.6 ii)

Reper	Poziția în lista de lansare	Termene t_{ij}				
		U_1	U_2	U_3	U_4	U_5
b	1	3	10	12	20	25
d	2	7	13	20	22	33
c	3	8	15	24	27	40
a	4	11	19	27	34	**42**

În concluzie, dacă reperele sunt lansate în ordinea *a , b , c , d*, întregul proces durează $t_{45} = 45$ minute. Procesarea reperelor în ordinea *b , d , c , a* durează numai $t_{45} = 42$ minute.

În ceea ce priveşte rezolvarea problemei de ordonanțare în flux deosebim două cazuri importante.

- Ordonanțarea în flux pe două utilaje;

- Ordonanțarea în flux pe $m \geq 3$ utilaje.

Separarea celor două cazuri este urmarea complexității lor diferite: în primul caz avem de a face cu o problemă de optimizare uşoară; al doilea caz este tot atât de "dificil" cât este o problemă a comisvoiajorului!

10.3.1 Ordonanțarea în flux pe două utilaje

Reamintim problema:

Un număr de repere sunt prelucrate pe două utilaje U_1 şi U_2. Fiecare reper se prelucrează **mai întâi** pe U_1 şi **apoi** pe U_2, duratele operațiilor fiind cunoscute.În ce ordine vor fi lansate reperele pentru ca timpul necesar terminării lor să fie **minim**?

Următorul algoritm, datorat lui JOHNSON (1954), rezolvă problema într-un număr de paşi egal cu numărul reperelor. Pentru simplitate vom presupune că toate duratele de prelucrare sunt diferite între ele.În cazul în care mai multe operații – fie pe U_1 fie pe U_2 fie pe amândouă – au durate egale vom modifica uşor unele din ele pentru ca toate să fie distincte!

Start: Toate reperele sunt declarate **neprogramate**.

Pasul 1 Se identifică operația **cu cea mai mică** durată, aferentă unui reper **neprogramat**.

Pasul 2 Dacă operația selectată se execută pe **primul** utilaj U_1, reperul respectiv se programează **la început**, dar **după** reperele anterior programate la început.

Dacă operația selectată se execută pe **al doilea** utilaj U_2, reperul în cauză se va programa **la sfârşit** dar **înaintea** reperelor anterior programate la sfârşit.

Se şterge reperul programat din lista reperelor neprogramate şi se revine la pasul 1 în cadrul unei noi iterații.

Exemplul 4 Reperele $R_1,...,R_8$ se procesează în flux pe utilajele U_1 şi U_2. În tabelul 10.7 sunt date duratele operațiilor (ore).

Tabelul 10.7

Reper	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
Durata operației pe U_1	8	4	9	6	2	5	8	3
Durata operației pe U_2	7	2	5	10	3	5	4	6

Să se determine ordinea de lansare în execuție a reperelor care minimizează durata întregului proces de prelucrare.

Soluție. Perturbăm „uşor” unele durate pentru ca toate să fie diferite – vezi tabelul 10.8. După cum se va vedea în final, unele modificări sunt de prisos în timp ce altele conduc la mai multe ordonări optimale – adică cu aceeaşi durată totală minimă! Important este faptul că, prin perturbare, la fiecare iterație a algoritmului, un singur reper este programat! Pentru calculul termenelor de terminare t_{ij} ale diferitelor operații se revine la valorile inițiale ale duratelor.

Tabelul 10.8

Reper	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
Durata operației pe U_1	8	4	9	6	2	5,01	8,01	3,01
Durata operației pe U_2	7	2,01	5	10	3	5,02	4,01	6,01

(modificările s-au făcut „la întâmplare!)

Start: declarăm toate reperele neprogramate.

Iterația 1 Cea mai scurtă durată - 2 ore – o are prelucrarea reperului R_5 pe **primul** utilaj U_1, drept care reperul R_5 este programat **la începutul** listei de lansare:

R_5							

Ştergem reperul R_5 din lista reperelor neprogramate.

Iterația 2 Minimul duratelor operațiilor neprogramate este 2,01; operația respectivă este aferentă reperului R_2 şi se execută pe **al doilea utilaj** U_2. Programăm reperul R_2 **la sfârşitul** listei de lansare:

R_5							R_2

Ştergem reperul R_7 din lista reperelor neprogramate.

Continuând, se găseşte în final următoarea ordine de lansare în execuție:

R_5	R_8	R_6	R_4	R_1	R_3	R_7	R_2

Pentru ordinea rezultată calculăm termenele de terminare t_{ij} ale diferitelor operații, aplicând relația (3) – vezi tabelul 10.9

Tabelul 10.9

Reper	R_5	R_8	R_6	R_4	R_1	R_3	R_7	R_2
Momentele de terminare ale operațiilor pe U_1	2	5	10	16	24	33	41	45

Momentele de terminare ale operațiilor pe U_2	5	11	16	26	33	38	45	**47**

Durata minimă de procesare a tuturor reperelor este de 47 ore.
Notă: cititorul atent a observat desigur că singura perturbare care a contat, a vizat valoarea 5 prezentă de trei ori în tabelul 3. Prin „permutarea" perturbărilor acestei valori (vezi tabelul 10.8) - restul modificărilor rămânând aceleaşi sau chiar schimbându-se!! - se obțin alte două ordonări

R_5	R_8	R_4	R_1	R_3	R_6	R_7	R_2

şi

R_5	R_8	R_4	R_1	R_6	R_3	R_7	R_2

cu aceeaşi durată totală de execuție, conform tabelelor 10.10 şi10.11

Tabelul 10.10

Reper	R_5	R_8	R_4	R_1	R_3	R_6	R_7	R_2
Momentele de terminare ale operațiilor pe U_1	2	5	11	19	28	33	41	45
Momentele de terminare ale operațiilor pe U_2	5	11	21	28	33	38	45	**47**

Tabelul 10.11

Reper	R_5	R_8	R_4	R_1	R_6	R_3	R_7	R_2
Momentele de terminare ale operațiilor pe U_1	2	5	11	19	24	33	41	45
Momentele de terminare ale operațiilor pe U_2	5	11	21	28	33	38	45	**47**

10.3.2 Ordonanțarea în flux pe $m \geq 3$ utilaje

Pentru ordonanțarea în flux pe cel puțin trei utilaje prezentăm o procedură simplă dar foarte performantă, **euristica NEH** (după inițialele autorilor **N**AVAZ , **E**NSCORE şi **H**AM)

Start: Se aranjează reperele într-o **listă** $\mathcal{L}$ în ordinea **descrescătoare** a **sumelor** duratelor de procesare pe **toate** utilajele.

Se iau primele două repere şi se ordonează aşa încât să se minimizeze timpul necesar terminării lor.

Celelalte repere se vor programa în ordinea în care apar în lista $\mathcal{L}$ după schema din:

Pasul iterativ: Presupunem programate primele $k - 1$ repere din lista $\mathcal{L}$. În raport cu aceste repere, deja programate, pentru reperul din poziția k a listei $\mathcal{L}$ există k locuri posibile de **inserție** – vezi figura 10.3.

Pentru fiecare loc de inserție se calculează durata procesării celor k repere, în ordinea rezultată din inserție. Se inserează reperul considerat în locul corespunzător duratei **minime**.

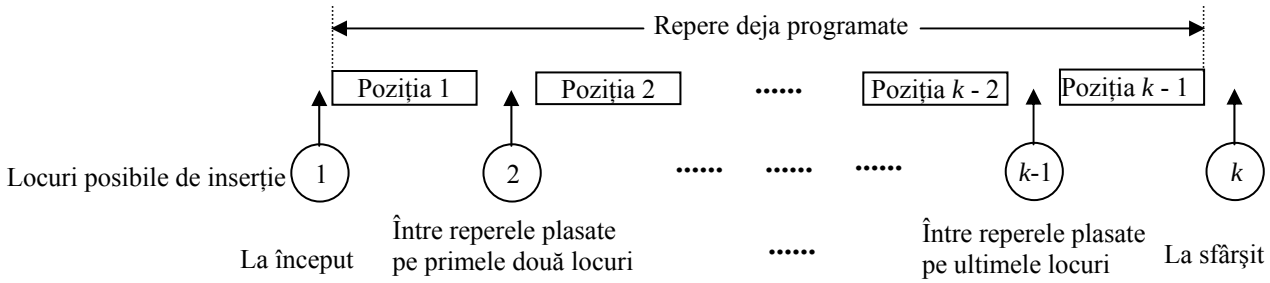


Figura 10.3

Exemplul 5. Aplicăm algoritmul NEH la programarea lansării în execuție a reperelor din exemplul 3. Sumele duratelor de prelucrare pe cele cinci utilaje sunt date în tabelul 10.12:

Tabelul 10.12

Reper	*a*	*b*	*c*	*d*
Durata totală a operațiilor de prelucrare	19	25	17	24

Reperele vor fi analizate în ordinea *b* , *d* , *a* , *c*.
Primele două pot fi lansate în execuție fie în ordinea *b* urmat de *d* fie invers. Calculele termenelor corespunzătoare t_{ij} de terminare a operațiilor – conform relației (3) – sunt date în tabelele 10.13 i) și 10.13 ii).

Tabelul 10.13 i)

Reper	Poziția în lista de lansare	Termene t_{ij}				
		U_1	U_2	U_3	U_4	U_5
b	1	3	10	12	20	25
d	2	7	13	20	22	**33**

Tabelul 10.13 ii)

Reper	Poziția în lista de lansare	Termene t_{ij}				
		U_1	U_2	U_3	U_4	U_5
d	1	4	7	14	16	24
b	2	7	14	16	24	**29**

Se constată că lansarea reperului *d* înaintea lui *b* duce la o durată totală de execuție (a celor două repere!) mai mică: numai 29 minute față de 33 minute cât ar dura procesul corespunzător ordinii inverse. **Algoritmul cere ca această precedență să fie respectată în continuare!**

Iterația 1 Inserarea reperului următor *a* conduce la examinarea următoarelor succesiuni:

(***a*** , *d* , *b*); (*d* , ***a*** , *b*); (*d* , *b* , ***a***)

Termenele t_{ij} corespunzătoare sunt sistematizate în tabelele 10.14 i),ii) şi iii)

Reper	Poziția în lista de lansare	Termene t_{ij}				
		U_1	U_2	U_3	U_4	U_5
a	1	3	7	10	17	19
d	2	7	10	17	19	27
b	3	10	17	19	27	**32**

Tabelul 10.14 i)

Reper	Poziția în lista de lansare	Termene t_{ij}				
		U_1	U_2	U_3	U_4	U_5
d	1	4	7	14	16	24
a	2	7	11	17	24	26
b	3	10	18	20	32	**37**

Tabelul 10.14 ii)

Reper	Poziția în lista de lansare	Termene t_{ij}				
		U_1	U_2	U_3	U_4	U_5
d	1	4	7	14	16	24
b	2	7	14	16	24	29
a	3	10	18	21	31	**33**

Tabelul 10.14 iii)

Cea mai mică durată corespunde ordinii (*a* , *d* , *b*) şi aceasta va fi menținută în continuare.

Iterația 2 Pentru ultimul reper *c* din listă există patru posibilități de inserție:

(***c*** , *a* , *d* , *b*); (*a* , ***c*** , *d* , *b*); (*a* , *d* , ***c*** , *b*) ; (*a* , *d* , *b* , ***c***)

Ele sunt analizate în următoarele patru tabele:

Reper	Poziția în lista de lansare	Termene t_{ij}				
		U_1	U_2	U_3	U_4	U_5
c	1	1	3	7	10	17
a	2	4	8	11	18	20
d	3	8	11	18	20	28
b	4	11	18	20	28	**33**

Tabelul 10.15 i)

Reper	Poziția în lista de lansare	Termene t_{ij}				
		U_1	U_2	U_3	U_4	U_5
a	1	3	7	10	17	19
c	2	4	9	14	20	27
d	3	8	12	21	23	35
b	4	11	19	23	31	**40**

Tabelul 10.15 ii)

Reper	Poziția în lista de lansare	Termene t_{ij}				
		U_1	U_2	U_3	U_4	U_5
a	1	3	7	10	17	19
d	2	7	10	17	19	27
c	3	8	12	21	24	34
b	4	11	19	23	32	**39**

Tabelul 10.15 iii)

Reper	Poziția în lista de lansare	Termene t_{ij}				
		U_1	U_2	U_3	U_4	U_5
a	1	3	7	10	17	19
d	2	7	10	17	19	27
b	3	10	17	19	27	32
c	4	11	19	23	30	**39**

Tabelul 10.15 iv)

În concluzie, algoritmul NEH propune ca ordinea de lansare în execuție a reperelor să fie *c* , *a* , *d* , *b* pentru care durata totală este de 33 minute.

10.4 Ordonanțarea pe mai multe utilaje; cazul general

Reamintim pe scurt problema:

Reperele 1,2,...,*m* se prelucrează pe utilajele U_1, U_2, ... , U_n. fiecare reper parcurge utilajele (sau o parte dintre ele) într-o ordine prestabilită dictată de tehnologia de fabricație. Orice operație de prelucrare a reperului are o durată cunoscută şi este perfect determinată de reperul asupra căruia se execută ca şi de utilajul pe care ea se realizează (în consecință, dacă o operație „fizică" ca de exemplu „darea unei găuri" trebuie făcută la mai multe repere diferite,ea se va multiplica în atâtea „copii" câte repere suferă operația respectivă; este posibil ca aceste copii să aibe timpi de execuție diferiți!) În principiu, un utilaj poate executa operații aferente mai multor repere, dar la un moment dat nu poate procesa decât un singur reper. Nu există restricții în ceeace priveşte ordinea în care vor fi executate operațiile programate pe acelaşi utilaj şi orice operație, o dată începută, nu poate fi întreruptă.

În aceste ipoteze generale, se cere programarea operațiilor pe utilaje astfel încât timpul necesar executării tuturor operațiilor să fie minim.

Se poate întâmpla ca tehnologia de fabricație a unui reper să specifice efectuarea consecutivă a două operații „fizice" pe un acelaşi utilaj. În acest caz simplficăm situația comprimând cele două operații într-una singură cu durata egală cu suma duratelor operațiilor originale. Astfel, putem presupune că două operații consecutive ale aceluiaşi reper se execută pe utilaje diferite (implicând deci o „mutare" a reperului de la un utilaj la altul) În fine, vom presupune că oricare două operații neconsecutive ale aceluiaşi reper se execută pe utilaje diferite (altfel spus, nu este permisă „întoarcerea" unui reper pe un utilaj pe care a mai fost procesat într-o etapă anterioară!).

10.4.1 Formalizarea problemei

- Reperele sunt numerotate 1 , 2 , ... , *m* (ordinea de numarotare este arbitrară)
- Numerotăm cu 1,2,...,*p* operațiile aferente reperului 1 – în ordinea în care ele trebuie excutate. Apoi, numerotăm cu $p + 1$, $p + 2$,..., $p + q$ operațiile aferente reperului 2 ş.a.m.d. Adăugăm două operații fictive cu durata 0 cu semnificația de începere respective terminare a întregului process de prelucrare.

Fie $\mathcal{O} = \{ 0 , 1 , \dots , N \}$ lista tuturor operațiilor inclusiv cele fictive, 0 fiind operația de începere a procesului şi N operația de terminare. Pentru fiecae reper, operația 0 va precede **prima** operație efectuată asupra reperului în timp ce operația N va succede **ultimei** operații "reale" suferită de reperul respectiv.

Structura problemei generale de ordonanțare va fi vizualizată printr-un **graf orientat** $G = (\mathcal{O} , \mathcal{A})$ ale cărui elemente constitutive sunt descrise mai jos:

- Nodurile grafului se identifică cu operațiile din lista $\mathcal{O}$;

- Pentru fiecare pereche de **operații consecutive** *i* , *j* **ale aceluiaşi reper** va exista în G un **arc** (i,j). La aceste arce adăugăm şi:

-arcele de forma (0,*h*) unde *h* este **prima** operație ce se execută asupra unui reper;
-arcele de forma (*k*,N) unde *k* este **ultima** operație ce se execută asupra unui reper.

Arcele mai sus definite se numesc arce **conjunctive** şi totalitatea lor va fi notată cu $\mathcal{C}$. În desen ele vor fi reprezentate prin **săgeți pline.**

- Pentru fiecare pereche de operații distincte i , j **executate pe acelaşi utilaj** includem în G două arce (i,j) şi (j, i) numite **arce disjunctive**. În desen ele vor fi reprezentate prin **săgeți punctate**.Dacă în lista $\mathcal{O}$ avem $i < j$ arcul disjunctiv (i,j) se va numi **arc normal** iar arcul disjuctiv (j, i) va fi **opusul** primului. Totalitatea arcelor disjunctive va fi notată cu $\mathcal{D}$.

- În G nu există alte arce în afara celor descrise mai sus aşa că $\mathcal{A} = \mathcal{C} \cup \mathcal{D}$. Graful $G = (\mathcal{O} , \mathcal{A} = \mathcal{C} \cup \mathcal{D})$ astfel construit se numeşte **graful disjunctiv** asociat problemei de ordonanțare.

- Fiecărui arc $(i, j) \in \mathcal{A}$ îi este asociată o **pondere** $d_{ij} \geq 0$ reprezentând intervalul de timp dintre **începutul** operației i şi **începutul** operației j. De obicei, d_{ij} se identifică cu durata d_i a operației premergătoare i dar poate include, dacă este necesar:
- un **timp de pregătire** al utilajului pe care s-a executat operația i, în vederea executării operației j, aceasta în cazul în care i şi j sunt operații ce se execută **pe acelaşi** utilaj (echivalent, dacă arcul (i, j) este un arc disjunctiv)
- un **timp de mutare** al reperului de la utilajul pe care s-a executat operația i la utilajul pe care se va executa operația j, în cazul în care i , j sunt operații **consecutive** aferente aceluiaşi reper (echivalent, dacă (i,j) este un arc conjunctiv)

În exemplificările noastre vom presupune în mod constant că $d_{ij} = d_i$ aceasta însemnând că timpii menționați mai sus ori sunt nesemnificativi ori sunt incluşi în duratele operațiilor.

Exemplul 6 Considerăm cazul a două repere care se procesează pe trei utilaje. În tabelul 10.16 se dau duratele operațiilor; în diagrama alăturată se precizează ordinea în care reperele parcurg utilajele.

Tabelul 10.16

utilaj / reper	U_1	U_2	U_3
1	4	3	6
2	2	1	5

Reper 1 : $U_1 \rightarrow U_3 \rightarrow U_2$

Reper 2 : $U_3 \rightarrow U_2 \rightarrow U_1$

Notăm cu 1 , 2 , 3 operațiile efectuate asupra reperului 1 pe utilajele U_1 , U_3 respectiv U_2; mai departe notăm cu 4 , 5 şi 6 operațiile aferente reperului 2 şi executate pe utilajele U_3 , U_2 respectiv U_1. Atunci, pe utilajul U_1 vor fi executate operațiile 1 şi 6, pe U_2 se vor executa operațiile 3 şi 5 iar operațiile rămase 2 şi 4 vor fi executate pe U_3. Graful disjunctiv este vizualizat în figura 10.4

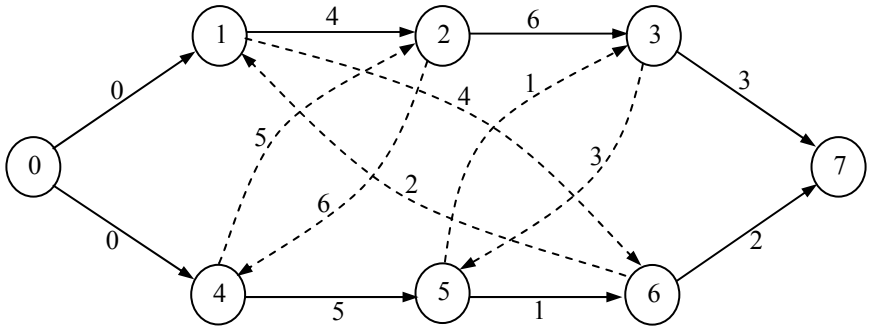


Figura 10.4

Revenind la problema generală de ordonanțare, a programa execuția reperelor pe utilaje înseamnă a stabili, pentru fiecare utilaj, ordinea în care vor fi executate şi termenele la care vor fi executate operațiile repartizate pe acel utilaj.

În termenii grafului disjunctiv asociat, aceasta înseamnă ca din fiecare pereche de arce disjunctive (*i*, *j*) şi (*j*, *i*) să se aleagă unul singur ce va marca ordinea în care vor fi executate cele două operații *i* şi *j*.

Exemplul 7 Din fiecare pereche de arce disjunctive care apar în graful din figura 10.4 selectăm arcul normal. Obținem graful aciclic din figura 10.5 care defineşte o programare a executării tuturor operațiilor: astfel, pe utilajul U_1 se va executa operația 1 de la primul reper şi apoi operația 6 de la al doilea reper etc. În raport cu această programare, timpul necesar executării celor două repere este dat de **durata drumului critic** adică de maximul valorilor drumurilor dintre nodul inițial 0 şi nodul final 7.

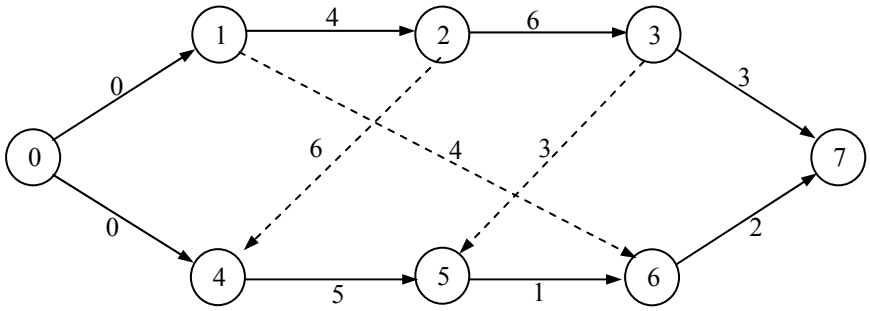


Figura 10.5

Efectuând calculele necesare – vezi figura 10.6, unde s-au folosit notațiile uzuale ale Analizei Drumului Critic, întrebuințate în rețelele coordonatoare „cu activitățile pe arce”!– găsim că, în raport cu programarea aleasă, operațiile celor două repere se termină în 18u.t. (unități de timp) A se vedea şi diagramele din figurile 10.7 şi 10.8

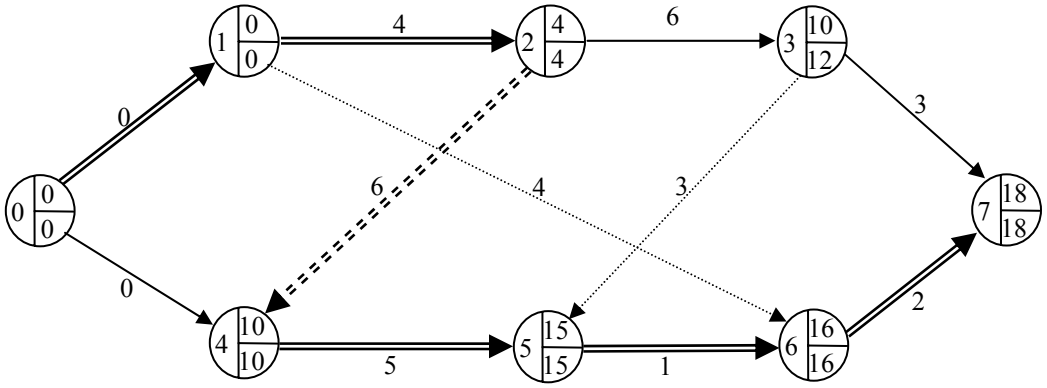


Figura 10.6

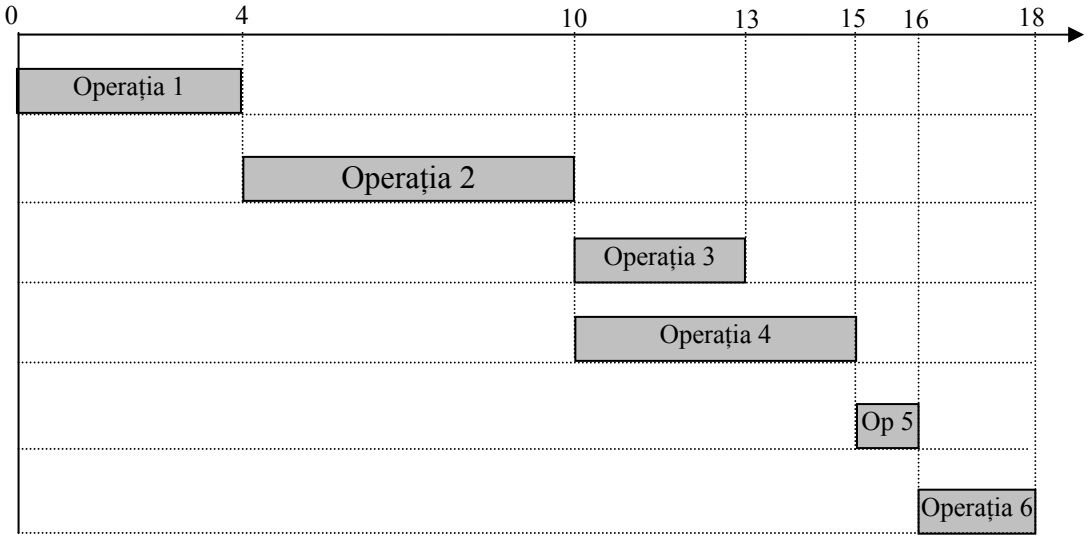


Figura 10.7

Gruparea operațiilor pe utilaje

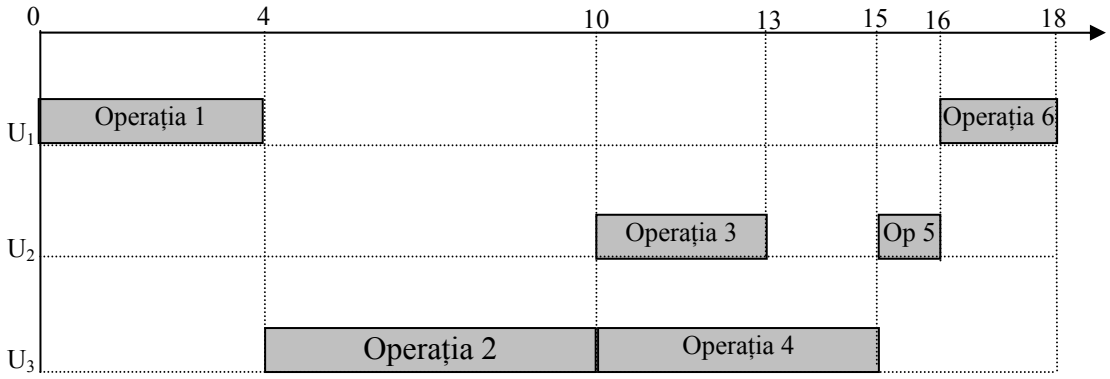


Figura 10.8

Exemplul 8 În graful din figura 10.4 vom face o altă selecție: din perechea de arce disjunctive (1,6) şi (6,1) vom alege arcul normal (1,6) iar din celelalte două perechi vom alege arcele opuse arcelor normale. Se obține o altă programare a execuției tuturor operațiilor cu durata de numai 14 u.t. conform calculelor din figura 10.9. Vezi şi diagrama din figura 10.10.

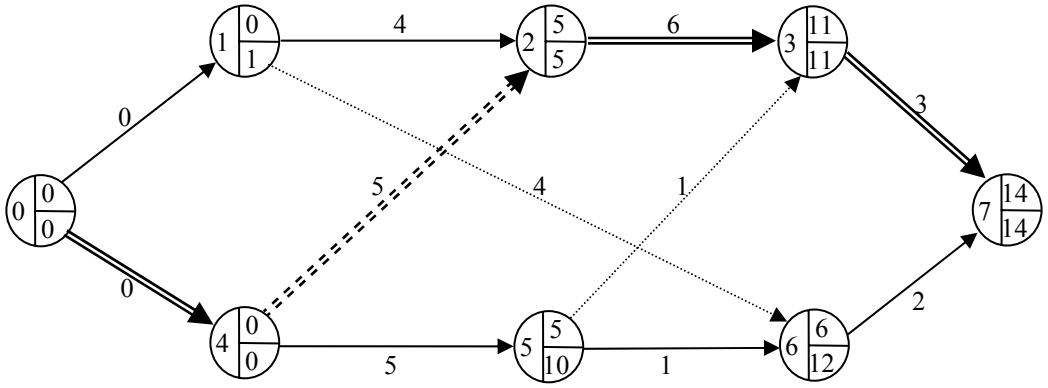


Figura 10.9

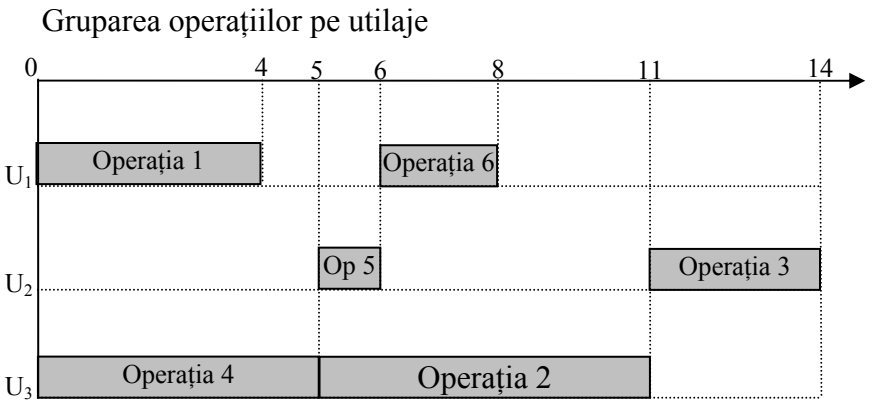


Figura 10.10

Să revenim la cazul general.

Vom numi **selecție** o submulțime $\mathcal{S} \subset \mathcal{D}$ care conține **exact** un arc din fiecare pereche de arce disjunctive din $\mathcal{D}$. Selecția $\mathcal{S}$ se va zice **admisibilă** dacă graful parțial $G(\mathcal{S}) = (\mathcal{O}, \mathcal{C} \cup \mathcal{S})$ este **aciclic** (adică nu are circuite). Evident, dacă p este numărul perechilor de arce disjunctive atunci numărul total de selecții posibile este 2^p dintre care unele pot să nu fie admisibile (≡ creează circuite în $G(\mathcal{S})$!)
Fiecare selecție admisibilă $\mathcal{S}$ corespunde unei programări a execuției operațiilor pe utilaje. Fie $d(\mathcal{S})$ durata minimă a executării tuturor operațiilor conform programării definite de selecția $\mathcal{S}$; $d(\mathcal{S})$ va fi **durata drumului critic** din graful $G(\mathcal{S})$.
Cu aceste pregătiri problema generală a ordonanțării se formalizează astfel:

Să se determine selecția admisibilă $\mathcal{S}^*$ cu proprietatea că $d(\mathcal{S}^*)$ are cea mai mică valoare.

10.4.2 Cazul general – o rezolvare suboptimală

S-a subliniat deja faptul că teoria complexității plasează ordonanțarea generală în categoria problemelor grele de optimizare combinatorială, pentru care nu există proceduri eficiente de rezolvare exactă şi foarte probabil că nu vor exista vreodată.. Mai mult, problemele de ordonanțare s-au dovedit a fi mult mai dificile chiar în comparație cu alte probleme combinatoriale grele. Pentru a ne face o idee de ceeace înseamnă acest lucru, vom aminti că deja au fost rezolvate probleme de tip comisvoiajor cu sute şi chiar mii de oraşe în timp ce soluția optimă a unei probleme de ordonanțare a zece repere pe zece utilaje, propusă în 1963, a fost determinată abia în 1986, rezolvarea durând cinci ore!

Pe de altă parte, ordonanțarea este o problemă frecvent întâlnită în practică şi din acest motiv este firesc să se acorde atenție căutării de metode suboptimale, capabile de a produce soluții acceptabile în timp util.

În continuare vom prezenta ideile generale ale unei euristici vechi, foarte simple şi destul de performante.

- Procedura programează operațiile unei probleme de ordonanțare în ordinea rangurilor lor (**rangul** unei operații, aferente unui anumit reper, este numărul de ordine al operației în şirul de prelucrări ce determină realizarea reperului respectiv. Fiecare reper are o **primă** operație, urmată de **a doua**, **a treia** ș.a.m.d.). Cu alte cuvinte, nici o operație nu va fi programată mai înainte de programarea tuturor operațiilor anterioare ei.
- Programarea unei operații se va face la cel mai mic termen posibil, condiția de programare fiind aceea ca utilajul prelucrător să fie disponibil pe toată durata operației.
- Dacă operațiile aferente mai multor repere sunt de programat pe un acelaşi utilaj şi la acelaşi termen posibil de începere, are prioritate reperul pentru care durata operațiilor încă neprogramate este cea mai mare.
- În principiu, termenele operațiilor programate sunt definitive, adică, o dată stabilite,nu se mai modifică ulterior.

Exemplul 4 Şapte repere J_1 , J_2 ,..., J_7 se procesează pe utilajele M_1 , M_2 , M_3 , M_4. Ordinea în care sunt parcurse utilajele şi duratele operațiilor sunt date în figura 10.11 Se cere programarea operațiilor astfel încât durata execuției întregului lot să fie cât mai mică.

$$J_1 : M_1^{(3)} \rightarrow M_4^{(2)} \rightarrow M_3^{(9)} \rightarrow M_2^{(1)}$$
$$J_2 : M_1^{(8)} \rightarrow M_2^{(7)} \rightarrow M_3^{(6)}$$
$$J_3 : M_1^{(3)} \rightarrow M_3^{(4)} \rightarrow M_2^{(5)} \rightarrow M_4^{(6)}$$
$$J_4 : M_1^{(1)} \rightarrow M_2^{(9)} \rightarrow M_4^{(2)}$$
$$J_5 : M_2^{(8)} \rightarrow M_3^{(3)} \rightarrow M_1^{(5)} \rightarrow M_4^{(7)}$$
$$J_6 : M_2^{(4)} \rightarrow M_1^{(2)} \rightarrow M_4^{(6)} \rightarrow M_3^{(8)}$$
$$J_7 : M_3^{(9)} \rightarrow M_2^{(5)} \rightarrow M_4^{(4)} \rightarrow M_1^{(1)}$$

Valorile înscrise în paranteze reprezintă **duratele** operațiilor !

Figura 10.11

Soluție

- Conform procedurii descrise mai sus începem prin a programa **prima** operație a fiecărui reper. Operațiile de debut ale reperelor J_1 , J_2 , J_3 , J_4 se execută pe utilajul M_1.Duratele totale de prelucrare ale acestor repere sunt 15 , 21 , 18 respectiv 12 u.t. (unități de timp) Pe utilajul M_1 cele patru repere vor fi lansate în ordinea J_2 , J_3 , J_1 , J_4.

Pentru executarea primei operații, reperele J_5 şi J_6 au nevoie de utilajul M_2. Duratele totale de prelucrare sunt de 23u.t. şi respectiv 20u.t. astfel că J_5 va fi programat primul şi imediat după el J_6.

Reperul J_7 nu are concurență la programarea pe utilajul M_3.

În diagrama din figura 10.12 au fost vizualizate cele şapte operații de debut sub forma unor dreptunghiuri a căror dimensiune orizontală indică durata operației. Rangul operației este pus în evidență în colțul din dreapta sus.

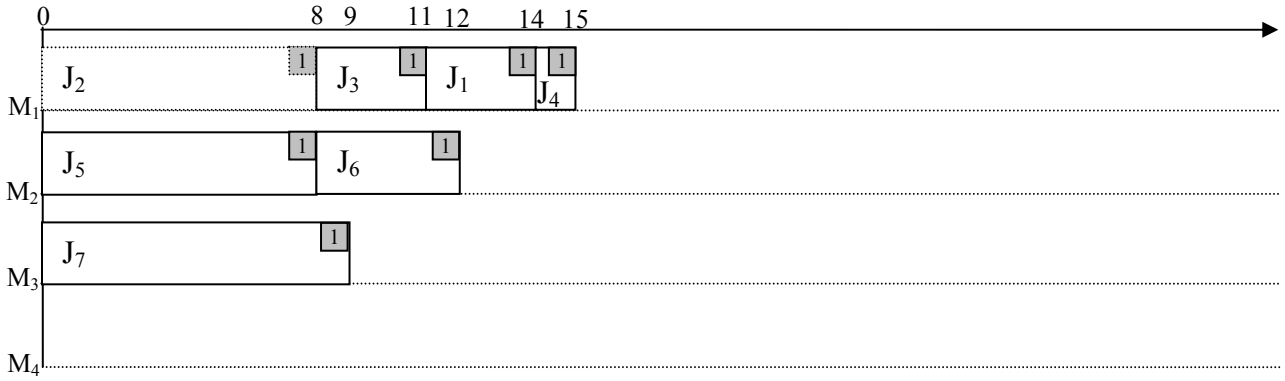


Figura 10.12

Programarea operațiilor de rangul 2 Din figura 10.11 rezultă că avem de programat:

- pe utilajul M_1: a 2-a operație la reperul J_6;
- pe utilajul M_2: a 2-a operație la reperele J_2 , J_4 şi J_7;
- pe utilajul M_3: a 2-a operație la reperele J_3 şi J_5;
- pe utilajul M_4: a 2-a operație la reperul J_1.

Rezolvăm pe rând aceste cerințe:

- a 2-a operație a reperului J_6 se programează pe M_1 la momentul 15 – vezi figura 10.12 care arată că de la momentul 15 utilajul M_1 este disponibil.
- utilajul M_2 este liber de la momentul 12; la acest moment J_2 şi J_7 sunt „în aşteptare" în timp ce pe J_4 nu s-a efectuat încă prima operație! Operațiile neprogramate aferente reperelor J_2 şi J_7 însumează 13u.t. respectiv 10u.t.Va avea prioritate reperul J_2, urmat de J_7 şi la urmă de J_4 care „între timp" a suferit prima operație.
- la momentul 9 când utilajul M_3 se eliberează, J_5 este în aşteptare în timp ce J_3 este în prelucrare. Evident, vom programa mai întâi J_5 şi imediat după terminare pe J_3.
- utilajul M_4 a fost liber până acum: nici o primă operație nu a fost prevăzută pe M_4. În consecință programăm J_1 pe M_4 la momentul în care J_1 termină prima operație, adică la momentul 14.

Noile programări sunt afişate în figura 10.13

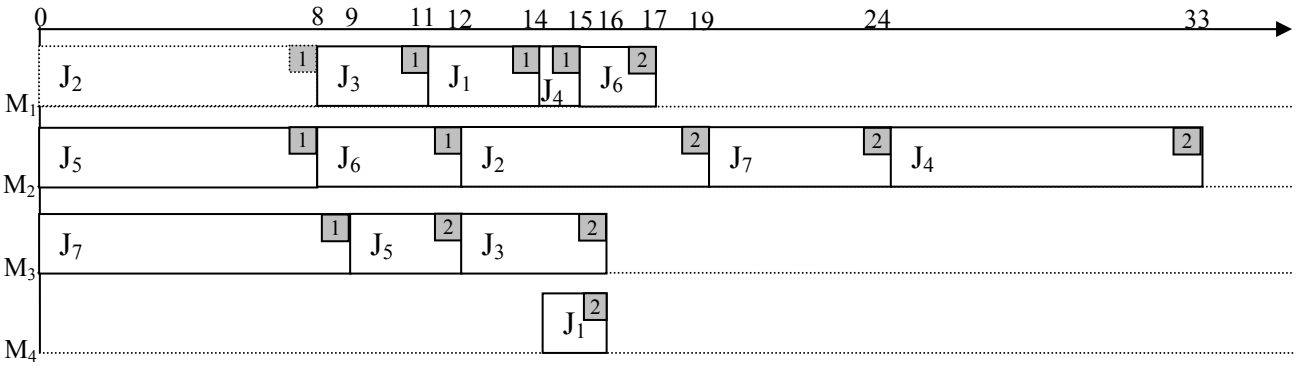


Figura 10.13

- **Programarea operațiilor de rangul 3** Recomandăm cititorului să aibe în față figura 10.13 şi datele de intrare!
  - reperul 5 se programează pe M_1 şi reperul J_3 se programează pe M_2 la momentele în care aceste utilaje devin disponibile.
  - pe utilajul M_3 va intra mai întâi J_1 şi apoi J_2 (de ce?)
  - pe utilajul M_4 intră mai întâi J_6 apoi J_7 şi la urmă J_4 (de ce?)

Inserarea noilor programări în diagrama 6 conduce la figura 10.14

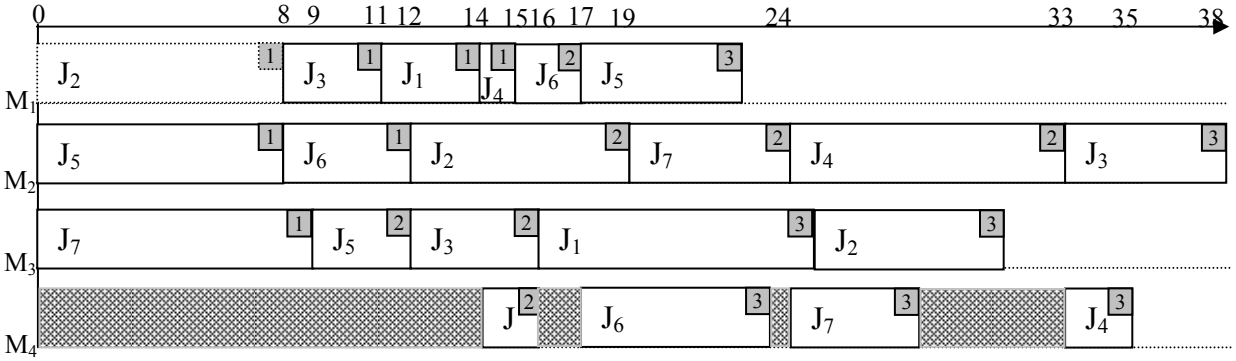


Figura 10.14

- **Programarea operațiilor de rangul 4** Situația din figura 10.14 arată că:
  - utilajul M_1 rămâne în aşteptare până la momentul 28 când reperul J_7 termină a 3- a operație pe M_4.
  - reperele J_1 şi J_6 se programează pe M_2 ,respectiv M_3, imediat ce aceste utilaje sunt disponibile.
  - pe utilajul M_4 intră mai întâi J_5 şi apoi J_3.

Programarea făcută, vizualizată în figura 10.15 (începând de la momentul 16 din motive de spațiu...), asigură realizarea celor şapte repere în 48u.t.

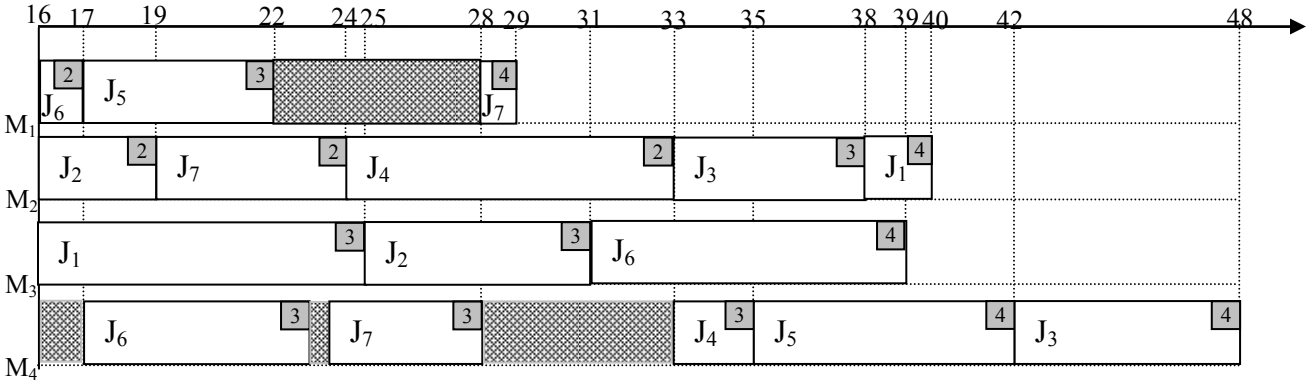


Figura 10.15

Observație: dacă se programează a 4-a operație a reperului J_5 imediat după a 3-a operație a lui J_6 pe utilajul M_4 se obține un timp mai bun! Invităm cititorul să modifice corespunzător figura 10.15 pentru a se convinge că toată treaba se poate termina în 44u.t. **Este acesta cel mai scurt timp?...**

10.5 Ilustrări numerice

1. (o margine inferioară pentru durata totală de execuție) Reperele 1, 2, ... , *m* se prelucrează **în flux** pe două maşini: întâi pe U_1 şi apoi pe U_2. Pentru reperul *i* se cunosc duratele d_{i1} şi d_{i2} de prelucrare pe cele două maşini.Introducem notațiile:

T^* ≡ durata minimă de prelucrare a tuturor reperelor;

$\sum_1 = \sum_{i=1}^{m} d_{i1}$ ≡ suma duratelor operațiilor pe maşina M_1;

$\sum_2 = \sum_{i=1}^{m} d_{i2}$ ≡ suma duratelor operațiilor pe maşina M_2.

Este evident că pe prima maşină nu avem timpi de aşteptare aşa încât:

$$T^* \geq \Sigma_1 + \min_i d_{i2} \qquad (1)$$

cu egalitate dacă, la momentul în care utilajul U_1 şi-a încheiat activitatea, utilajul U_2 este liber să prelucreze ultimul reper.
Pe al doilea utilaj vom avea din start un timp de aşteptare până când primul utilaj termină procesarea primului reper. Ca urmare

$$T^* \geq \Sigma_2 + \min_i d_{i1} \qquad (2)$$

Din (1) şi (2) rezultă următoarea **margine inferioară pentru durata minimă totală**:

$$T^* \geq \max\left\{\Sigma_1 + \min_i d_{i2}\ ;\ \Sigma_2 + \min_i d_{i1}\right\}$$

Următorul exemplu arată că inegalitatea poate fi strictă:

Tabelul 10.17

Reper	*a*	*b*	*c*	
Durata operației pe U_1	5	8	4	$\Sigma_1 = 17$
Durata operației pe U_2	2	7	7	$\Sigma_2 = 16$

Avem: $\Sigma_1 + \min_i d_{i2} = 17 + 2 = 19$; $\Sigma_2 + \min_i d_{i1} = 16 + 4 = 20$ dar T^* =21, conform diagramei din figura 10.16

:

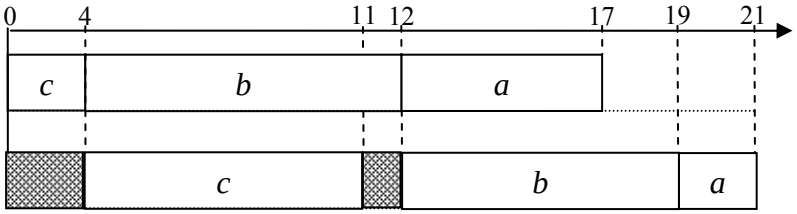


Figura 10.16

2. Nouă repere se prelucrează pe două maşini U_1 şi U_2. În tabelul 10.18 se precizează ordinea de efectuare a operațiilor de prelucrare şi duratele acestora (în paranteze).

Tabelul 10.18

Reper	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Durata operației pe **prima** maşină	$U_1(8)$	$U_1(7)$	$U_1(9)$	$U_1(4)$	$U_2(6)$	$U_2(5)$	$U_1(9)$	$U_2(1)$	$U_2(5)$
Durata operației pe **a doua** maşimă	$U_2(2)$	$U_2(5)$	$U_2(8)$	$U_2(7)$	$U_1(4)$	$U_1(3)$	-	-	-

Astfel:

- reperele 1 , 2 , 3 ,4 se prelucrează în flux, mai întâi pe U_1 şi apoi pe U_2;
- reperele 5 şi 6 se prelucrează şi ele în flux dar mai întâi pe U_2 şi apoi pe U_1;
- reperele 7 , 8 , 9 au nevoie de o singură operație.

Se cere programarea operațiilor astfel încât timpul total de prelucrare să fie minim.

Soluție I) Începem prin a programa operațiile reperelor 1, 2, 3,4 folosind algoritmul lui JOHNSON (secțiunea 10.3.1).Ordinea care minimizează timpul total de prelucrare al acestor repere este 4,3,2,1. Termenele de terminare ale operațiilor – raportate la un moment de referință 0 - sunt date în tabelul 10.19 Desfăşurarea în timp a operațiilor este vizualizată în figura 10.17

Tabelul 10.19

Reper	**4**	**3**	**2**	**1**
Termen de terminare a operației pe U_1	4	13	20	28
Termen de terminare a operației pe U_2	11	21	26	**30**

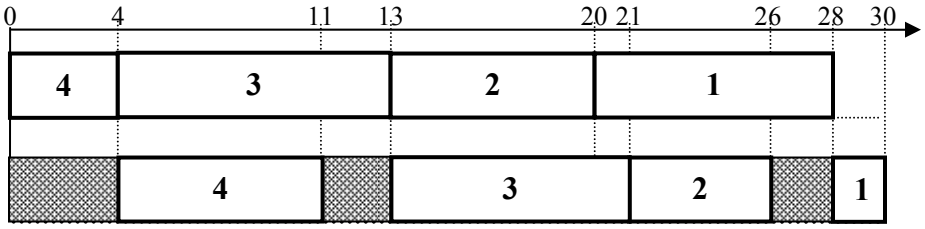


Figura 10.17

Zonele haşurate reprezintă intervale de timp în care a doua maşină nu este folosită (este în aşteptare).

II) Reperele 8 şi 9 se prelucrează numai pe U_2 şi necesită în total 1 + 5 = 6u.t. Observăm că maşina U_2 aşteaptă 4u.t. la început şi, după prelucrarea reperului 4 mai aşteaptă încă 2u.t. Dacă se amână începerea prelucrării reperului 4 pe maşina U_2 cu 2u.t. celelalte termene nu se modifică şi se obţine la început un interval de timp de 4 + 2 = 6u.t. suficient pentru procesarea reperelor 8 şi 9!

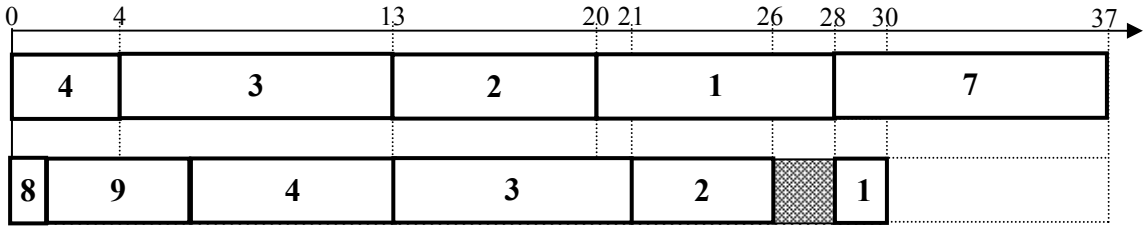


Figura 10.18

În ceeace priveşte reperul 7 acesta va fi programat pe maşina U_1 după ce aceasta a terminat prelucrarea reperelor 1,2,3,4. Noile programări apar în figura 10.18

III) Reperele 5 şi 6 se prelucrează mai întâi pe U_2 şi apoi pe U_1; operaţiile aferente însumează 6 + 5 = 11u.t. pe U_2 şi 4 + 3 = 7u.t. pe U_1. Avem tot interesul în a folosi şi timpul de aşteptare de 2u.t. pe care maşina U_2 îl înregistrează între prelucrările reperelor 2 şi 1! Drept care programăm reperele 5 şi 6 pe maşina U_2 imediat după terminarea reperului 2 – ordinea nu contează! – după care terminăm şi reperul 1(urmăriţi cu atenţie figura 10.18!) În acest fel, ambele maşini nu au nici un **timp mort** şi prin urmare soluţia construită – vizualizată în figura 10.19 - este **optimă**!

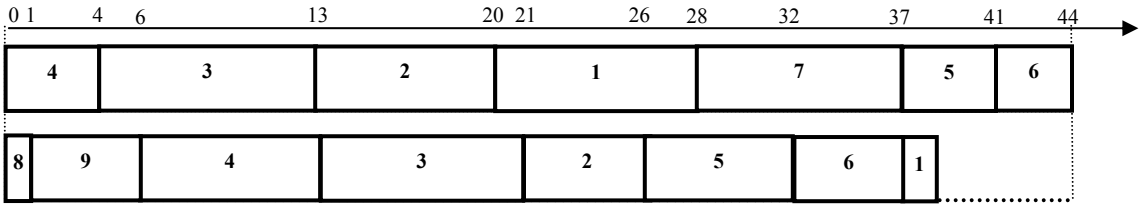


Figura 10.19

3. Avem de programat fabricarea a cinci repere $J_1,...,J_5$. Diferitele operații de prelucrare se fac pe patru utilaje $M_1,...,M_4$. Utilajele implicate în fabricarea fiecărui reper, ordinea de execuție a operațiilor şi duratele acestora – înscrise în paranteze – sunt date mai jos:

$$J_1 : M_1^{(2)} \rightarrow M_4^{(3)}$$
$$J_2 : M_1^{(3)} \rightarrow M_3^{(3)} \rightarrow M_4^{(2)}$$
$$J_3 : M_1^{(1)} \rightarrow M_4^{(3)} \rightarrow M_2^{(2)}$$
$$J_4 : M_1^{(4)} \rightarrow M_2^{(1)} \rightarrow M_3^{(3)}$$
$$J_5 : M_2^{(4)} \rightarrow M_3^{(4)}$$

Soluție. **I) Programăm operațiile de rangul 1.** Pe maşina M_1 „concurează" J_1 , J_2 , J_3 şi J_4. cu duratele totale de prelucrare de 5 , 8 , 6 respectiv 8u.t. Dintre J_2 şi J_4 „alegem" J_4 deoarece prelucrarea pe M_1 durează mai mult. Astfel maşina M_1 va prelucra cele patru repere în ordinea J_4 , J_2 , J_3 , J_1.Programarea primelor operații este vizualizată în diagrama din figura 10.20

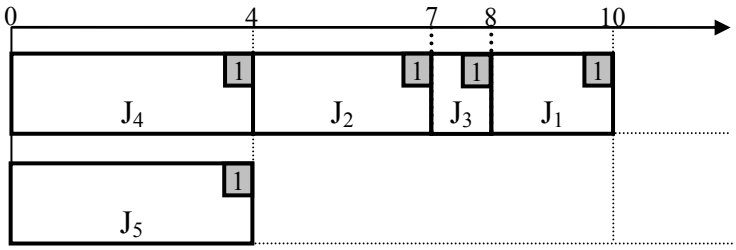


Figura 10.20

II) Programarea operațiilor de rangul 2 ține seama de momentele de eliberare ale utilajelor care au executat prima operație ca şi de momentele de eliberare ale reperelor din prima operație – vezi diagrama din figura 10.21

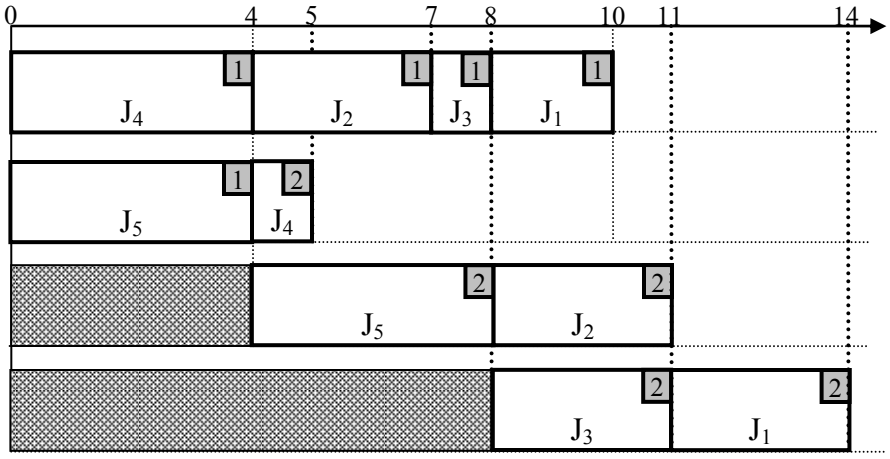


Figura 10.21

III) Programarea operațiilor de rangul 3 vizează numai reperele J_2 , J_3 şi J_4. Aceste operații se execută pe mașini diferite așa încât programarea se va face la cel mai devreme moment în care şi reperul şi utilajul sunt disponibile – vezi diagrama din figura 10.22

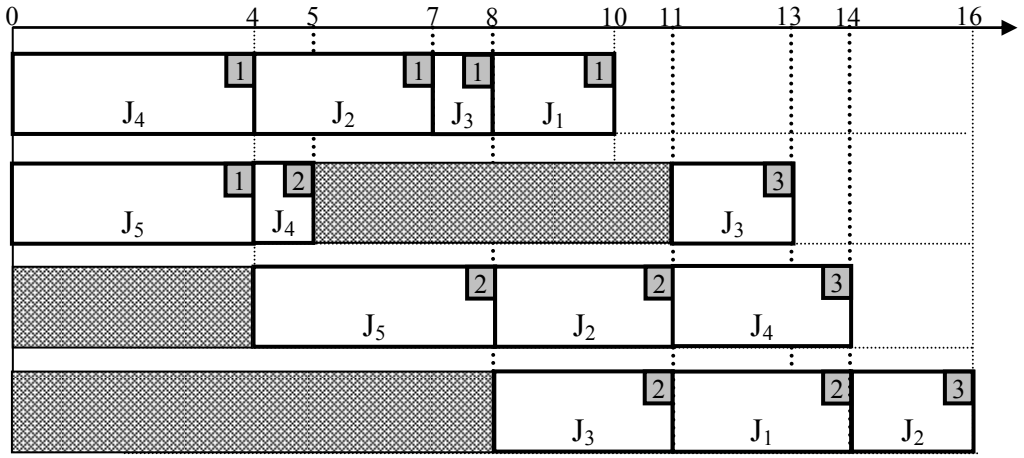


Figura 10.22

Probleme propuse

1. Şapte repere urmează a fi prelucrate pe un utilaj. În tabelul 10.20 se dau duratele operațiilor şi termenele impuse de planificator.În ce ordine ar trebui lansate reperele în execuție pentru ca întârzierea maximă față de termenele impuse să fie cât mai mică?

Tabelul 10.20

Reper j	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Durata p_j	3	4	4	6	9	11	15
Termen impus d_j	22	28	36	21	15	35	31

2. Opt repere urmează a fi prelucrate pe un utilaj. În tabelul 10.21 se dau duratele operațiilor şi termenele de livrare, care trebuie respectate de planificator.Se cere ordinea de lansare în execuție care minimizează suma termenelor de terminare.

Tabelul 10.21

Reper j	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Durata p_j	10	19	14	8	23	17	9	5
Termen de livrare $\overline{d}_j$	25	45	90	100	80	105	50	20

3. Zece repere se prelucrează în flux pe două utilaje U_1 şi U_2: prima operație se execută U_1 iar a doua pe U_2.Duratele operațiilor sunt date în tabelul 10.23 În ce ordine vor fi lansate în execuție cele zece repere pentru ca durata întregului proces de prelucrare să fie minimă?

Tabelul 10.23

Reper	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Durata operației pe U_1	14	22	50	32	36	41	28	22	30	20
Durata operației pe U_2	30	45	32	58	36	14	41	45	42	58

4. Programați executarea în flux a 15 repere pe două utilaje cu datele din tabelul 10.24

Tabelul 10.24

Reper	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Durata operației pe U_1	16	3	9	4	12	7	5	9	8	8	6	7	15	15	12
Durata operației pe U_2	9	18	8	6	7	7	8	6	9	6	10	5	10	12	11

5. Reperele *a* , *b* , *c* , *d* , *e* se prelucrează în flux pe utilajele U_1 , U_2 , U_3 , U_4. În tabelul 10.25 se dau duratele operațiilor. Să se aplice euristica NEH pentru determinarea unui program acceptabil de lansare în execuție a reperelor.

Tabelul 10.25

	U_1	U_2	U_3	U_4
a	10	4	6	12
b	8	11	15	7
c	5	3	14	12
d	12	4	13	8
e	9	14	5	11

6. Să se determine o soluție suboptimală pentru problema ordonanțării în flux a cinci repere *a*, *b*, *c*, *d*, *e* pe trei maşini U_1 , U_2 , U_3. duratele operațiilor sunt date în tabelul 10.26

Tabelul 10.26

	U_1	U_2	U_3
a	5	7	1
b	8	4	6
c	2	3	4
d	6	2	8
e	5	5	4

Un număr de repere se prelucrează pe mai multe utilaje. Fiecare reper parcurge utilajele într-o anumită ordine. Ordinea şi duratele operaţiilor (înscrise în paranteze) sunt date în diagramele care urmează. Pentru fiecare problemă de ordonanţare să se construiască o programare a operaţiilor care să conducă la un timp total de procesare cât mai mic.

7. Trei repere şi patru utilaje:

$$J_1 : M_1^{(9)} \to M_2^{(8)} \to M_3^{(4)}$$
$$J_2 : M_1^{(5)} \to M_2^{(6)} \to M_4^{(3)}$$
$$J_3 : M_3^{(10)} \to M_1^{(4)} \to M_2^{(9)}$$

8. Şase repere şi şase utilaje:

$$J_1 : M_3^{(1)} \to M_1^{(3)} \to M_2^{(6)} \to M_4^{(7)} \to M_6^{(3)} \to M_5^{(6)}$$
$$J_2 : M_2^{(8)} \to M_3^{(5)} \to M_5^{(10)} \to M_6^{(10)} \to M_1^{(10)} \to M_4^{(4)}$$
$$J_3 : M_3^{(5)} \to M_4^{(4)} \to M_6^{(8)} \to M_1^{(9)} \to M_2^{(1)} \to M_5^{(7)}$$
$$J_4 : M_2^{(5)} \to M_1^{(5)} \to M_3^{(5)} \to M_4^{(3)} \to M_5^{(8)} \to M_6^{(9)}$$
$$J_5 : M_3^{(9)} \to M_2^{(3)} \to M_5^{(5)} \to M_6^{(4)} \to M_1^{(3)} \to M_4^{(1)}$$
$$J_6 : M_2^{(3)} \to M_4^{(3)} \to M_6^{(9)} \to M_1^{(10)} \to M_5^{(4)} \to M_3^{(1)}$$

Indicaţie: Singura problemă de alegere este la programarea operaţiilor de rangul 1.Reamintim regula: are prioritate reperul pentru care suma duratelor operaţiilor neprogramate este cea mai mare.

La programarea operaţiilor de rangul 2,3,...se va aplica regula „de bun simţ": nici unui utilaj nu îi este permis să aştepte dacă poate executa o anumită operaţie.

Pentru această problemă este dovedit că timpul total minim este de 55u.t. Cum apreciaţi soluţia găsită de dvs?

9. Patru repere şi patru utilaje:

$$J_1 : M_4^{(83)} \to M_3^{(45)} \to M_1^{(5)} \to M_2^{(70)}$$
$$J_2 : M_4^{(45)} \to M_1^{(24)} \to M_3^{(58)} \to M_2^{(80)}$$
$$J_3 : M_1^{(29)} \to M_2^{(56)} \to M_3^{(29)} \to M_4^{(61)}$$
$$J_4 : M_4^{(74)} \to M_3^{(45)} \to M_2^{(64)} \to M_1^{(43)}$$

10. **Algy**, **Bertie**, **Charles** şi **Digby** sunt membri ai aceluiaşi club. În fiecare sâmbătă dimineaţa ei se întâlnesc la club pentru a citi ziarele şi a face politică. Toţi cred că numai patru ziare merită a fi citite sâmbăta la club si acestea sunt (în ordine alfabetică): Daily Express (abreviat DE), Financial Times (FT) The Guardian (G) şi The Sun (S). De fiecare dată cele patru ziare îi aşteaptă în sala de lectură a clubului dar (din motive de economie, of course...) într-un singur exemplar.

În cea mai bună tradiţie britanică, fiecare din cei patru soseşte la club la o anumită oră, citeşte ziarele într-o anumită ordine acordând fiecăruia un acelaşi timp, indiferent de dată. Astfel Algy soseşte întotdeauna la 8,30 lecturează FT timp de 60' apoi G timp de 30' după care răsfoieşte DE în 2' şi S în alte 5'. Digby apare la club întotdeauna la 9,30 stă 90' pe S după care îşi aruncă ochii pe FT apoi pe G şi la urmă pe DE timp de un minut pentru fiecare. Alte detalii privind momentul sosirii la club, ordinea în care sunt citite ziarele şi timpul de lectură sunt date în tabelul 10.27

Tabelul 10.27

	Momentul sosirii la club	Ordinea şi timpul lecturării
Algy	8,30	FT (60'); G (30)' ; DE (2') ; S (5')
Bertie	8,45	G (75'); DE(3') ; FT (25') ; S (10')
Charles	8,45	DE (5'); G (15'); FT (10') ; S (30')
Digby	9,30	S (90'); FT (1'); G (1') ; DE (1')

Ca nişte gentlemeni care se respectă, în cazul în care ziarul dorit este citit de altcineva, fiecare aşteaptă fără comentarii până când ziarul devine disponibil. Problema celor patru prieteni este de a stabili un program de lecturare a ziarelor astfel încât operația să se termine până la ora 12 când se seveşte prânzul...

11. Într-un atelier sunt patru puncte de lucru: primul punct este dotat cu patru freze, al doilea cu trei maşini de găurit şi filetat, al treilea cu două strunguri şi ultimul cu o maşină de polizat. În atelier sunt procesate cinci tipuri dee repere. Pentru fiecare reper se cunoaşte cererea şi fluxul de prelucrare, adică ordinea în care se execută operațiile specifice, utilajele executante şi duratele operațiilor – vezi tabelul 10.28 Să se construiască un program de producție prin care comenzile să se realizeze în cel mai scurt timp.

Tabelul 10.28

Tipul reperului	Cerere	Flux de prelucrare (utilaj, durata operației în minute)			
		Operația 1	Operația 2	Operația 3	Operația 4
1	8	**L** , 6	**M** , 9	**D** , 9	**G** , 4
2	12	**L** , 8	**M** , 12	**G** , 5	
3	10	**M** , 6	**D** , 6	**G** , 3	
4	6	**M** , 8	**L** , 5	**D** , 10	**M** , 12
5	12	**M** , 7	**D** , 8	**L** , 6	**D** , 9

M ≡ freză ; **D** ≡ maşina de găurit şi filetat ; **L** ≡ strung ; **G** ≡ maşina de polizat