Eleonor Ciurea Laura Ciupală

ALGORITMI

Introducere în algoritmica fluxurilor în reţele

Prefaţa

Fluxurile în reţele reprezintă un domeniu captivant pentru mulţi cercetători, practicieni şi studenţi. Algoritmica fluxurilor în reţele a cunoscut în ultimul timp o dezvoltare puternică. Aplicaţiile ei sunt remarcabile în domeniile: reţele de calculatoare, chimie, inginerie, telecomunicaţii, transport, planificare, sisteme sociale etc. Toate aceste aplicaţii îi conferă o importanţă crescută. Aşa se explică numeroasele lucrări ştiinţifice şi tratate apărute în ultimii treizeci de ani care se ocupă în totalitate sau parţial de problemele fluxurilor în reţele. De asemenea, algoritmica fluxurilor în reţele este în acelaşi timp şi un domeniu foarte dinamic, care a ţinut pasul cu progresul exponenţial cunoscut de dezvoltarea calculatoarelor.

Algoritmica fluxurilor în reţele poate fi considerată ca un subdomeniu al algoritmicii grafurilor. Există numeroase contribuţii româneşti la dezvoltarea diferitelor domenii ale grafurilor. Menţionăm aici doar pe cele ale prof. I. Tomescu (Universitatea Bucureşti), prof. E. Olaru (Universitatea Galaţi), prof. C. Croitoru (Universitatea Iaşi) şi prof. T. Toadere (Universitatea Cluj).

Această carte este prima în limba română care tratează într-un spaţiu tipografic mai extins, principalele probleme de fluxuri în reţele. În primele patru capitole (noţiuni introductive, parcurgeri de grafuri, arbori şi arborescenţe, distanţe şi drumuri minime) sunt prezentate noţiunile şi problemele necesare în celelalte capitole. În următoarele opt capitole sunt prezentate diferite probleme ale fluxurilor în reţele (fluxuri maxime, fluxuri minime, fluxuri de cost minim, fluxuri dinamice, algoritmul simplex pentru fluxuri, fluxuri cu multiplicatori, fluxuri cu mai multe produse). Cititorul interesat de alte aspecte ale problemelor de fluxuri în reţele poate consulta bibliografia indicată în carte, care conţine peste 300 de monografii, articole de sinteză şi articole originale.

Pentru lectura manuscrisului şi observaţiile utile aducem mulţumirile noastre prof. I. Tomescu (Universitatea Bucureşti). De asemenea, dacă numărul erorilor de tehnoredactare, nu este aşa de mare, aceasta se datoreşte doamnei Gabriela Mailat (Universitatea Braşov).

Primele cinci capitole ale cărţii pot constitui material didactic pentru disciplina *Algoritmica grafurilor* predată studenţilor domeniului informatică. Următoa-

rele patru capitole pot fi utilizate pentru un curs ţinut studenţilor unui masterat în informatică. Ultimele trei capitole pot fi recomandate absolvanţilor unui masterat în informatică care doresc să-şi elaboreze lucrarea de dizeraţie în domeniul fluxurilor în reţele. De asemenea, cartea poate fi consultată de cercetători şi practicieni (ingineri, economişti etc.) în domeniu.

Autorii

Cuprins

Capitolul 1

Noţiuni introductive

1.1 Vocabularul de bază în teoria grafurilor

Avertizăm cititorul că terminologia din Teoria Grafurilor nu este complet unitară. Vom păstra, în cea mai mare parte, terminologia consacrată în literatura română de specialitate.

Definiţia 1.1. Se numeşte *graf orientat* un triplet $G = (N, A, g)$ format dintr-o mulţime N de elemente numite *noduri* sau *vârfuri*, dintr-o familie A de elemente numite *arce* şi dintr-o aplicaţie $g : A \to N \times N$ numită *funcţie de incidenţă*, prin care fiecărui element $a \in A$ i se asociază o pereche ordonată $(x, y) \in N \times N$ cu $x \neq y$; dacă eliminăm condiţia $x \neq y$ atunci arcul de forma (x, x) se numeşte *buclă*, iar G se numeşte *graf general orientat*.

În continuare vom presupune că graful orientat G este finit, adică mulţimea nodurilor $N = \{\ldots, x, \ldots\}$ este finită şi familia arcelor $A = (\ldots, a, \ldots)$ este un şir finit. Cardinalul mulţimii N notat $|N| = n$, se numeşte *ordinul grafului* orientat G.

Un graf orientat $G = (N, A, g)$ se reprezintă grafic în modul următor:
i. fiecare nod $x \in N$ se reprezintă printr-un punct sau cerculeţ în plan;
ii. fiecare arc $a \in A,\ a = (x, y)$ se reprezintă printr-o linie care uneşte cele două noduri şi pe care se află o săgeată cu sensul de la x la y.

Exemplul 1.1. Graful din figura 1.1. este de ordinul 8.

Fig.1.1.

Observaţia 1.1. Reprezentarea grafică a unui graf orientat $G = (N, A, g)$ nu este unică. În primul rând nodurile se pot plasa în plan la întâmplare. În al doilea rând nu este obligatoriu ca arcele să fie segmente de dreaptă.

Exemplul 1.2. Cele trei grafuri din figura 1.2. reprezintă acelaşi graf.

Fig.1.2.

Definiţia 1.2. Două grafuri orientate, $G_1 = (N_1, A_1, g_1)$ şi $G_2 = (N_2, A_2, g_2)$ se numesc *izomorfe*, dacă există o bijecţie $\phi : N_1 \to N_2$ cu proprietatea că aplicaţia $\psi : A_1 \to A_2$, definită prin $\psi(x_1, y_1) = (\phi(x_1), \phi(y_1)), (x_1, y_1) \in A_1, (\phi(x_1), \phi(y_1)) = (x_2, y_2) \in A_2$, este o bijecţie.

Exemplul 1.3. Grafurile $G_1 = (N_1, A_1, g_1)$ şi $G_2 = (N_2, A_2, g_2)$ prezentate în figura 1.3 sunt izomorfe.

Fig.1.3.

Dacă definim bijecţia ϕ prin $\phi(1) = 4$, $\phi(2) = 3$, $\phi(3) = 5$, $\phi(4) = 6$, $\phi(5) = 1$, $\phi(6) = 2$, atunci $\psi(1, 2) = (\phi(1), \phi(2)) = (4, 3)$, $\psi(1, 4) = (\phi(1), \phi(4)) = (4, 6)$, $\psi(1, 6) = (\phi(1), \phi(6)) = (4, 2)$ etc. este evident bijecţia $\psi : A_1 \to A_2$.

Definiţia 1.3. Dacă oricare pereche ordonată $(x, y) \in N \times N$ este imaginea a cel mult q, $q > 1$, elemente din A, atunci $G = (N, A, g)$ se numeşte *multigraf orientat*.

Exemplul 1.4. În figura 1.1. avem un multigraf orientat cu $q = 2$.

În paragrafele şi capitolele următoare ne vom ocupa de grafuri orientate cu $q = 1$. În acest caz funcţia g este injectivă şi familia A este o mulţime. Un astfel de graf se numeşte *digraf* şi se notează $G = (N, A)$.

Definiţia 1.4. Se numeşte *graf neorientat* un triplet $G = (N, A, g)$ format dintr-o mulţime N de elemente numite *noduri* sau *vârfuri*, dintr-o familie A de elemente numite *muchii* şi dintr-o aplicaţie $g : A \to \mathcal{P}_2(N)$ numită *funcţie de incidenţă*, prin care fiecărui element $a \in A$ i se asociază o pereche $\{x, y\} \in \mathcal{P}_2(N)$; dacă considerăm $g : A \to \mathcal{P}_{(2)}(N)$, unde: $\mathcal{P}_{(2)}(N) = \mathcal{P}_2(N) \cup \mathcal{P}_1(N)$, atunci aplicaţia g asociază fiecărui element $a \in A$, fie o pereche de noduri $\{x, y\} \in \mathcal{P}_2(N)$ care se notează $[x, y]$ sau $[y, x]$, fie un nod $\{x\} \in \mathcal{P}_1(N)$ care se notează $[x, x]$ şi se numeşte *buclă*, iar G se numeşte *graf general neorientat*.

În continuare vom presupune că graful neorientat G este finit, adică mulţimea nodurilor N este finită şi familia muchiilor A este un şir finit.

Un graf neorientat $G = (N, A, g)$ se reprezintă grafic la fel ca în cazul grafurilor orientate cu deosebirea că o muchie $a = [x, y]$ se reprezintă printr-o linie care uneşte cele două noduri fără săgeata care precizează sensul în cazul arcului. De asemenea, la fel ca în cazul grafurilor orientate se defineţe izomorfismul a două grafuri neorientate.

Definiţia 1.5. Dacă oricare pereche $[x, y] \in \mathcal{P}_2(N)$ este imaginea a cel mult $q, q > 1$, elemente din A atunci $G = (N, A, g)$ se numeţe *multigraf neorientat*.

Exemplul 1.5. Dacă se elimină săgeata de pe fiecare arc al grafului din figura 1.1, atunci fiecare arc (x, y) devine o muchie $[x, y]$ şi graful devine un multigraf neorientat cu $q = 3$.

În paragrafele şi capitolele următoare ne vom ocupa de grafuri neorientate cu $q = 1$. În acest caz funcţia g este injectivă şi familia A este o mulţime. Un graf neorientat cu $q = 1$ se numeşte *graf simplu* şi se notează $G = (N, A)$.

În majoritatea problemelor prezentate în continuare, se presupune că graful este orientat. De aceea, vom prezenta definiţiile pentru grafurile orientate. Definiţiile pentru grafurile neorientate se pot deduce, în cele mai multe cazuri, cu uşurinţă din definiţiile corespunzătoare pentru grafuri orientate. Totuşi, vom face unele precizări referitoare la unele definiţii pentru grafurile neorientate.

Definiţia 1.6. Fie un arc $a = (x, y) \in A$. Nodurile x, y se numesc *extremităţile arcului*, nodul x este *extremitatea iniţială* şi y *extremitatea finală*; nodurile x şi y se numesc *adiacente*.

Evident că, pentru o muchie $a = [x, y] \in A$, nu se pot preciza extremitatea iniţială şi extremitatea finală.

Definiţia 1.7. Fie un arc $a = (x, y) \in A$. Nodul x se numeşte *predecesor* al nodului y şi nodul y se numeşte *succesor* al nodului x. Mulţimea succesorilor nodului x este mulţimea $V^+(x) = \{y|(x, y) \in A\}$ şi mulţimea predecesorilor nodului x este mulţimea $V^-(x) = \{y|(y, x) \in A\}$. Mulţimea $V(x) = V^+(x) \cup V^-(x)$ se numeşte *vecinătatea* nodului x. Dacă $V(x) = \emptyset$, atunci x se numeşte *nod izolat*.

Exemplul 1.6. Pentru 2 - graful din figura 1.1. avem $V^+(4) = \{2, 5\}, V^-(4) = \{2\}, V(4) = \{2, 5\}$.

Definiţia 1.8. Se spune că nodul x este *adiacent* cu submulţimea $N' \subset N$, dacă $x \notin N'$ şi $x \in V(N'), V(N') = \cup\{V(y)|y \in N'\}$.

Observaţia 1.2. Conceptul de digraf se poate defini şi prin perechea $G = (N, \Gamma)$, unde $N = \{\ldots, x, \ldots\}$ este mulţimea nodurilor şi Γ aplicaţia multivocă $\Gamma : N \to \mathcal{P}(N), \Gamma(x) = V^+(x), x \in N$. Cele două definiţii sunt echivalente. Într-adevăr, dacă digraful este dat sub forma $G = (N, A)$, atunci $\Gamma(x) = V^+(x), x \in N$. Reciproc, dacă digraful este dat sub forma $G = (N, \Gamma), \Gamma(x) = V^+(x),\ x \in N$, atunci se determină $E^+(x) = \{(x, y)|y \in V^+(x)\}, x \in N$ şi $A = \cup\{E^+(x)|x \in N\}$. De asemenea, se defineşte $\Gamma^{-1}(x) = V^-(x), x \in N$. Recursiv avem

$$\Gamma^2(x) = \Gamma(\Gamma(x)), \ldots, \Gamma^k(x) = \Gamma\left(\Gamma^{k-1}(x)\right), \ldots$$

şi analog

$$\Gamma^{-2}(x) = \Gamma^{-1}(\Gamma^{-1}(x)), \ldots, \Gamma^{-k}(x) = \Gamma^{-1}\left(\Gamma^{-(k-1)}(x)\right), \ldots$$

Un nod $y \in \Gamma^k(x)$ se numeşte *descendent* al nodului x şi un nod $z \in \Gamma^{-k}(x)$

se numeşte *ascendent* al nodului x.

Exemplul 1.7. Fie digraful din figura 1.4.

Fig.1.4.

Dacă digraful este dat sub forma $G = (N, A)$ cu $N = \{1, 2, 3\}$, $A = \{a_1, a_2, a_3, a_4\} = \{(1, 2), (1, 3), (2, 1), (2, 3)\}$, atunci $\Gamma(1) = V^+(1) = \{2, 3\}, \Gamma(2) = V^+(2) = \{1, 3\}, \Gamma(3) = V^+(3) = \emptyset$ şi am obţinut digraful sub forma $G = (N, \Gamma)$.

Dacă digraful este dat sub forma $G = (N, \Gamma)$ cu $N = \{1, 2, 3\}, \Gamma(1) = \{2, 3\}, \Gamma(2) = \{1, 3\}, \Gamma(3) = \emptyset$, atunci $E^+(1) = \{(1, 2), (1, 3)\}, E^+(2) = \{(2, 1), (2, 3)\}$, $E^+(3) = \emptyset$ şi $A = E^+(1) \cup E^+(2) \cup E^+(3) = \{(1, 2), (1, 3), (2, 1), (2, 3)\} = \{a_1, a_2, a_3, a_4\}$.

Definiţia 1.9. Două arce se numesc *adiacente* dacă au cel puţin o extremitate în comun.

Definiţia 1.10. Fie arcul $a = (x, y) \in A$. Se spune că arcul a este *incident către exterior* la nodul x şi *incident către interior* la nodul y. Mulţimea arcelor incidente către exterior la nodul x este $E^+(x) = \{a_i \mid a_i = (x, y) \in A\}$, mulţimea arcelor incidente către interior la nodul x este $E^-(x) = \{a_j \mid a_j = (y, x) \in A\}$ şi mulţimea arcelor incidente la nodul x este $E(x) = E^+(x) \cup E^-(x)$. Numărul $\rho^+(x) = |E^+(x)|$ se numeşte *semigradul exterior* al nodului x, numărul $\rho^-(x) = |E^-(x)|$ se numeşte *semigradul interior* al nodului x şi numărul $\rho(x) = \rho^+(x) + \rho^-(x)$ se numeşte *gradul nodului* x.

Exemplul 1.8. Se consideră multigraful orientat din figura 1.1. Avem $\rho^+(3) = 3, \rho^-(3) = 1$, deci $\rho(3) = 3 + 1 = 4$.

Dacă $G = (N, A, g)$ este un graf neorientat atunci $\rho(x) = \rho^+(x) = \rho^-(x)$. În cazul în care $G = (N, A, g)$ este multigraf orientat atunci $\rho^+(x) \geq |V^+(x)|, \rho^-(x) \geq |V^-(x)|$ şi în care $G = (N, A)$ este un digraf relaţiile sunt verificate cu egalităţi. Evident că dacă x este nod izolat atunci $\rho(x) = 0$. Un nod x cu $\rho(x) = 1$ se numeşte *perdant*. Dacă toate nodurile lui G au acelaşi grad ρ atunci G se numeşte *graf* ρ *- regulat*. Un graf 0-regulat se numeşte *graf nul* şi un graf 3-regulat se numeşte *graf trivalent* sau *cubic*.

Definiţia 1.11. Într-un graf orientat $G = (N, A, g)$ se numeşte *lanţ* de la nodul x_1 la nodul x_{k+1}, o secvenţă $L = (x_1, a_1, x_2, \ldots, x_k, a_k, x_{k+1})$, $x_i \in N$, $i = 1, \ldots, k + 1, a_i \in A$, $i = 1, \ldots, k$, cu proprietatea că fiecare arc a_i este de forma (x_i, x_{i+1}) sau (x_{i+1}, x_i). Dacă $a_i = (x_i, x_{i+1})$, atunci a_i se numeşte *arc direct* al lanţului şi dacă $a_i = (x_{i+1}, x_i)$, atunci a_i se numeşte *arc invers* al lanţului. Numărul de arce din secvenţă este, prin definiţie, *lungimea lanţului* L. Lanţul L se numeşte *simplu* dacă fiecare arc a_i din secvenţă este utilizat o singură dată şi se numeşte *elementar* dacă fiecare nod x_i din secvenţă este utilizat o singură dată. Dacă $x_{k+1} = x_1$ atunci lanţul simplu L se numeşte *ciclu* şi se notează $\overset{\circ}{L}$.

Un lanţ poate fi reprezentat şi ca o secvenţă de arce $L = (a_1, \ldots, a_k)$ şi în cazul când $G = (N, A)$ este digraf ca o secvenţă de noduri $L = (x_1, \ldots, x_{k+1})$.

Exemplul 1.9. Se consideră graful orientat din figura 1.1. Secvenţa $L = (1, a_1, 2, a_8, 4, a_3, 2, a_6, 3, a_2, 2, a_8, 4) = (a_1, a_8, a_3, a_6, a_2, a_8)$ este un lanţ, dar nu este nici lanţ simplu, nici lanţ elementar. Secvenţa $L = (1, a_1, 2, a_3, 4, a_4, 2, a_2, 3, a_6, 2) = (a_1, a_3, a_4, a_2, a_6)$ este un lanţ simplu, dar nu este lanţ elementar. Secvenţa $L = (1, a_1, 2, a_6, 3, a_7, 5, a_9, 4) = (a_1, a_6, a_7, a_9)$ este un lanţ elementar.

Noţiunea de lanţ se poate defini şi într-un graf neorientat $G = (N, A, g)$ ca o secvenţă $L = (x_1, a_1, x_2, \ldots, x_k, a_k, x_{k+1})$ cu proprietatea că fiecare muchie a_i din secvenţă este de forma $a_i = [x_i, x_{i+1}]$ şi lanţul poate fi reprezentat şi sub forma $L = (a_1, \ldots, a_k)$. Un lanţ $L = (x_1, a_1, x_2, \ldots, x_k, a_k, x_{k+1})$, al grafului orientat $G = (N, A, g)$, în care fiecare arc a_i este arc direct, adică este de forma $a_i = (x_i, x_{i+1})$, se numeşte *lanţ orientat* sau *drum* şi se notează prin D. Un lanţ simplu orientat cu $x_{k+1} = x_1$ se numeşte *ciclu orientat* sau *circuit* şi se notează prin $\overset{\circ}{D}$. Noţiunile de drum şi circuit au sens numai pentru grafuri orientate.

Exemplul 1.10. Se consideră graful orientat din figura 1.1. Secvenţa $D = (1, a_1, 2, a_3, 4, a_8, 2, a_2, 3, a_6, 2, a_3, 4, a_8, 2) = (a_1, a_3, a_8, a_2, a_6, a_3, a_8)$ este un drum, dar nici drum simplu, nici drum elementar. Secvenţa $D = (1, a_1, 2, a_3, 4, a_8, 2, a_2, 3, a_7, 5) = (a_1, a_3, a_8, a_2, a_7)$ este un drum simplu, dar nu este un drum elementar. Secvenţa $D = (3, a_6, 2, a_4, 4, a_9, 5) = (a_6, a_4, a_9)$ este un drum elementar.

În paragrafele următoare vom considera, în general, lanţuri şi drumuri elementare, dar fără a mai specifica de fiecare dată atributul "elementar", ci numai în cazurile când este necesar.

Definiţia 1.12. Se spune că graful orientat $G' = (N', A', g')$ este un *subgraf* al grafului orientat $G = (N, A, g)$ dacă $N' \subseteq N$ şi $A' \subseteq A$. Dacă $N' \subseteq N$ şi $A' = (N' \times N') \cap A$ atunci $G' = (N', A', g')$ se numeşte *subgraf indus* în G de mulţimea de noduri N'. Dacă $N' = N$ şi $A' \subseteq A$ atunci $G' = (N', A', g')$ se numeşte *subgraf parţial* al lui G.

Concepte similare se pot defini în mod analog şi pentru grafuri neorientate.

Fie $G = (N, A)$ un digraf cu mulţimea nodurilor $N = \{1, \ldots, n\}$ şi mulţimea arcelor $A = \{1, \ldots, m\}$. *Matricea de adiacenţă* asociată digrafului G este matricea

$$M = (m_{ij})_{n \times n}$$

unde

$$m_{ij} = \begin{cases} 1 & \text{dacă} \quad (i,j) \in A \\ 0 & \text{dacă} \quad (i,j) \notin A \end{cases}$$

Evident că au loc relaţiile:

$$\rho^+(i) = \sum_{j=1}^{n} m_{ij}, i \in N, \quad \rho^-(j) = \sum_{i=1}^{n} m_{ij}, j \in N.$$

Matricea de incidenţă asociată digrafului G este matricea $\overline{M} = (\overline{m}_{ij})_{n\times m}$ unde

$$\overline{m}_{ij} = \begin{cases} 1 & \text{dacă} \quad j \in E^+(i) \\ -1 & \text{dacă} \quad j \in E^-(i) \\ 0 & \text{dacă} \quad j \notin E(i) \end{cases}$$

În acest caz au loc relaţiile:

$$\rho^+(i) = \sum_{j|\overline{m}_{ij}>0} \overline{m}_{ij}, i \in N, \quad \rho^-(i) = \sum_{j|\overline{m}_{ij}<0} |\overline{m}_{ij}|, i \in N.$$

Exemplul 1.11. Pentru digraful din figura 1.4. avem:

$$M = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \overline{M} = \begin{bmatrix} 1 & 1 & -1 & 0 \\ -1 & 0 & 1 & 1 \\ 0 & -1 & 0 & -1 \end{bmatrix}$$

Se remarcă faptul că matricea de incidenţă $\overline{M}$ are pe fiecare coloană un 1, un -1, şi $n-2$ zerouri.

Dacă $G = (N, A)$ este un graf simplu neorientat atunci

$$m_{ij} = \begin{cases} 1 & \text{dacă} \quad [i,j] \in A, \\ 0 & \text{dacă} \quad [i,j] \notin A \end{cases}$$

şi

$$\overline{m}_{ij} = \begin{cases} 1 & \text{dacă} \quad j \in E(i), \\ 0 & \text{dacă} \quad j \notin E(i) \end{cases}$$

Evident că în cazul unui graf simplu neorientat matricea de adiacentă M este simetrică, adică $m_{ij} = m_{ji}$ pentru $i = 1, \ldots, n$ şi $j = 1, \ldots, n$.

1.2 Clase de grafuri

O clasă de grafuri este alcătuită din grafuri cu proprietăţi particulare.

Definiţia 1.13. Se spune că digraful $G = (N, A)$ este *simetric* dacă oricare ar fi $(x, y) \in A$ implică $(y, x) \in A$.

Cu alte cuvinte, digraful $G = (N, A)$ este simetric dacă orice pereche de noduri adiacente este legată prin arce în ambele sensuri.

Definiţia 1.14. Se spune că digraful $G = (N, A)$ este *antisimetric* dacă oricare ar fi $(x, y) \in A$ implică $(y, x) \notin A$.

Deci, digraful $G = (N, A)$ este antisimetric dacă orice pereche de noduri adiacente este legată cel mult printr-un singur arc.

Fig.1.5.

Exemplul 1.12. Digraful reprezentat în figura 1.5(a) este simetric şi digraful reprezentat în figura 1.5(b) este antisimetric.

Definiţia 1.15. Se spune că digraful $G = (N, A)$ este *pseudosimetric* dacă $\rho^+(x) = \rho^-(x)$ pentru fiecare nod $x \in N$.

Orice digraf simetric este şi pseudosimetric. Reciproca nu este adevărată.

Exemplul 1.13. Digraful reprezentat în figura 1.5(a) este simetric deci este şi pseudosimetric, iar cel reprezentat în figura 1.5(b) este antisimetric, dar nu este pseudosimetric. Digraful reprezentat în figura 1.6(a) este pseudosimetric, dar nu este simetric şi nici antisimetric, iar cel reprezentat în figura 1.6(b) este antisimetric şi pseudosimetric.

Fig.1.6.

Definiţia 1.16. Se spune că digraful $G = (N, A)$ este *nesimetric* dacă există un arc $(x, y) \in A$ pentru care $(y, x) \notin A$.

Cu alte cuvinte, digraful $G = (N, A)$ este nesimetric dacă nu este simetric. Orice graf antisimetric este nesimetric. Reciproca nu este adevărată. Orice graf pseudosimetric care nu este simetric este nesimetric. Reciproca nu este adevărată.

Definiţia 1.17. Se spune că digraful $G = (N, A)$ este *complet* dacă oricare ar fi $(x, y) \notin A$ implică $(y, x) \in A$.

Un graf simplu neorientat $G = (N, A)$ cu n noduri care este complet se notează cu K_n şi are $n(n-1)/2$ muchii.

Exemplul 1.14. Digraful din figura 1.5(b) este antisimetric deci este nesimetric şi digrafurile reprezentate în figura 1.6 sunt pseudosimetrice care nu sunt simetrice, deci sunt nesimetrice. Digraful din figura 1.7(a) este nesimetric, dar nu este nici antisimetric şi nici pseudosimetric.

Fig.1.7.

Digraful reprezentat în figura 1.5(b) este complet. Graful K_4 este reprezentat în figura 1.7(b).

Definiţia 1.18. Se spune că digraful $G = (N, A)$ este *graf turneu* dacă este antisimetric şi complet.

Denumirea de graf turneu se justifică prin faptul că un astfel de graf poate reprezenta un turneu sportiv în care fiecare jucător (echipă) joacă câte un meci cu toţi (toate) ceilalţi (celelalte) jucători (echipe).

Definiţia 1.19. Se spune că digraful $G = (N, A)$ este *tranzitiv* dacă oricare ar fi

$(x, y) \in A$ şi $(y, z) \in A$ implică $(x, z) \in A$.

Exemplul 1.15. Digraful reprezentat în figura 1.5.(b) este un graf turneu şi de asemenea este tranzitiv.

Definiţia 1.20. Se spune că digraful $G = (N, A)$ este o *clică* dacă este simetric şi complet.

Denumirea de clică se justifică prin faptul că un digraf simetric şi complet poate reprezenta o coaliţie sociologică, politică etc. Noţiunea de clică are sens şi pentru grafuri neorientate, astfel graful K_n este o clică.

Definiţia 1.21. Se spune că digraful $G = (N, A)$ este *bipartit* dacă mulţimea nodurilor N admite o partiţie în două submulţimi N_1 şi $N_2(N_1 \neq \emptyset, N_2 \neq \emptyset, N_1 \cap N_2 = \emptyset, N_1 \cup N_2 = N)$, astfel încât orice arc $(x, y) \in A$ are una dintre extremităţi în N_1, iar cealaltă extremitate în N_2.

Un digraf bipartit se notează $G = (N_1, N_2, A)$ şi se caracterizează prin inexistenţa ciclurilor care conţin un număr impar de arce. Un graf neorientat bipartit şi complet $G = (N_1, N_2, A)$ cu $|N_1| = n_1$ şi $|N_2| = n_2$ se notează K_{n_1,n_2}.

Exemplul 1.16. Digraful reprezentat în figura 1.8(a) este o clică. În figura 1.8.(b) este reprezentat graful $K_{2,3}$.

Fig.1.8.

Definiţia 1.22. Se spune că digraful $G = (N, A)$ este *planar* dacă este posibil să-l reprezentăm pe un plan astfel încât oricare două arce să se întâlnească eventual numai în extremităţile lor.

Analog se defineşte un graf simplu neorientat planar. Arcele (muchiile) unui graf planar $G = (N, A)$ determină contururi care mărginesc regiuni numite *feţe*. Există o singură regiune din plan fără contur numită *faţă nemărginită*.

Exemplul 1.17. Digraful reprezentat în figura 1.5(b) este planar, deoarece poate fi reprezentat ca în figura 1.9. Feţele 1,2,3 sunt feţele mărginite, iar faţa 4 este

Fig.1.9.

faţa nemărginită. Exemple de grafuri neplanare minimale sunt K_5 şi $K_{3,3}$.

O altă clasă de grafuri este alcătuită din grafuri asociate unui graf dat.

În paragraful 1.1. s-a definit subgraful $G' = (N', A')$ al unui graf dat $G = (N, A)$.

Definiţia 1.23. Se spune că digraful $\overline{G} = (\overline{N}, \overline{A})$ este *complementarul* digrafului $G = (N, A)$ dacă $\overline{N} = N$ şi $\overline{A} = \{(x,\ y) \mid x, y \in N,\ x \neq y,\ (x, y) \notin A\}$.

Analog se defineşte complementarul unui graf simplu neorientat. Operaţia de complementare este involutivă, adică complementarul complementarului este

graful iniţial.
Exemplul 1.18. Digraful reprezentat în figura 1.10(b) este complementarul digrafului reprezentat în figura 1.10.(a)
Definiţia 1.24. Se spune că digraful $G^{-1} = (N^{-1}, A^{-1})$ este *inversul* digrafului $G = (N, A)$ dacă $N^{-1} = N$ şi A^{-1} se obţine din A prin inversarea sensului arcelor.

Fig.1.10.

Operaţia de inversare este involutivă, adică inversul inversului este graful iniţial ($(G^{-1})^{-1} = G$). Graful autoreciproc ($G^{-1} = G$) este simetric.
Exemplul 1.19. Digraful reprezentat în figura 1.11(b) este inversul digrafului reprezentat în figura 1.11(a).

Fig.1.11.

1.3 Operaţii cu grafuri

Definiţia 1.25. *Suma cartezianǎ* a două digrafuri $G_1 = (N_1, A_1)$ şi $G_2 = (N_2, A_2)$ este notată $G_1 + G_2$ şi este digraful $G = (N, A)$ definit astfel:

$$N = N_1 \times N_2 = \{x_1x_2 | x_1 \in N_1, x_2 \in N_2\},$$
$$A = \{(x_1x_2, y_1y_2) | x_1, y_1 \in N_1, x_2, y_2 \in N_2,$$
$$x_1 = y_1 \text{ şi } (x_2, y_2) \in A_2 \text{ sau } x_2 = y_2 \text{ şi} (x_1, y_1) \in A_1\}.$$

Definiţia 1.26. *Produsul cartezian* al două digrafuri $G_1 = (N_1, A_1)$ şi $G_2 = (N_2, A_2)$ este notat $G_1 \times G_2$ şi este digraful $G = (N, A)$ definit astfel:

$$N = N_1 \times N_2 = \{x_1x_2 | x_1 \in N_1, x_2 \in N_2\},$$
$$A = \{(x_1x_2, y_1y_2) | x_1, y_1 \in N_1, x_2, y_2 \in N_2,$$
$$(x_1, y_1) \in A_1 \quad \text{şi } (x_2, y_2) \in A_2\}.$$

Suma cartezianǎ şi produsul cartezian a două grafuri simple neorientate se definesc analog ca pentru două digrafuri. De asemenea suma cartezianǎ şi produsul cartezian a p grafuri, $p > 2$, se definesc asemănător ca pentru două grafuri.
Exemplul 1.20. Digraful reprezentat în figura 1.13(a) este suma cartezianǎ a digrafurilor reprezentate în figura 1.12.

Fig.1.12.

Arcul $(x_1x_2, x_1y_2) \in A$ deoarece $x_1 = x_1$ şi $(x_2, y_2) \in A_2$; arcul $(x_1x_2, y_1x_2) \in A$ deoarece $x_2 = x_2$ şi $(x_1, y_1) \in A_1$ etc. Digraful reprezentat în figura 1.13(b) este produsul cartezian al digrafurilor reprezentate în figura 1.12.

Fig.1.13.

Arcul $(x_1x_2, z_1y_2) \in A$ deoarece $(x_1, z_1) \in A_1$ şi $(x_2, y_2) \in A_2$; arcul $(y_1y_2, z_1x_2) \in A$ deoarece $(y_1, z_1) \in A_1$ şi $(y_2, x_2) \in A_2$ etc.

1.4 Structuri de date utilizate în reprezentarea grafurilor

Performanţa unui algoritm utilizat pentru rezolvarea unei probleme din teoria grafurilor depinde nu numai de structura algoritmului utilizat ci şi de modul de reprezentare în calculator a topologiei grafului.

Fie un digraf $G = (N, A)$ cu $N = \{1, \ldots, n\}$ şi $A = \{1, \ldots, m\}$. Se consideră funcţia valoare $b : A \to \Re$ şi funcţia capacitate $c : A \to \Re^+$. Funcţia valoare b reprezintă fie funcţia lungime, fie funcţia cost etc. Funcţiile b, c etc. se pot defini şi pe mulţimea nodurilor.

Definiţia 1.27. Un digraf $G = (N, A)$ pe care s-a/s-au definit funcţia b sau/şi funcţia c se numeşte *reţea orientată* şi se notează fie $G = (N, A, b)$, fie $G = (N, A, c)$, respectiv $G = (N, A, b, c)$.

Analog se defineşte o *reţea neorientată*.

În acest paragraf se vor prezenta cele mai uzuale structuri de date utilizate pentru reprezentarea reţelelor. Pentru reprezentarea unei reţele este necesar, în general, să memorăm două tipuri de informaţii:

a) informaţii privind topologia reţelei;

b) informaţii privind funcţia b sau/şi funcţia c.

În acest paragraf se prezintă reprezentările reţelelor orientate. Reprezentările corespunzătoare reţelelor neorientate sunt similare cu reprezentările reţelelor orientate. La sfârşitul acestui paragraf sunt prezentate pe scurt şi reprezentările reţelelor neorientate.

Reprezentarea prin *matricea de adiacenţă* constă în memorarea matricei de adiacenţă

$$M = (m_{ij})_{n \times n}$$

unde

$$m_{ij} = \begin{cases} 1 & \text{dacă} \quad (i,j) \in A \\ 0 & \text{dacă} \quad (i,j) \notin A \end{cases}$$

Dacă este necesar să memorăm funcţia b sau/şi funcţia c, atunci memorăm matricea valoare

$$B = (b_{ij})_{n\times n}$$

unde

$$b_{ij} = \begin{cases} b(i,j) & \text{dacă} \quad (i,j) \in A, \\ 0 & \text{dacă} \quad (i,j) \notin A \end{cases}$$

sau/şi matricea capacitate

$$C = (c_{ij})_{n\times n}$$

unde

$$c_{ij} = \begin{cases} c(i,j) & \text{dacă} \quad (i,j) \in A, \\ 0 & \text{dacă} \quad (i,j) \notin A \end{cases}$$

Reprezentarea prin *matricea de incidenţă* constă în memorarea matricei de incidenţă

$$\overline{M} = (\overline{m}_{ij})_{n\times m}$$

unde

$$\overline{m}_{ij} = \begin{cases} 1 & \text{dacă} \quad j \in E^+(i), \\ -1 & \text{dacă} \quad j \in E^-(i), \\ 0 & \text{dacă} \quad j \notin E(i) \end{cases}$$

Exemplul 1.21. Considerăm reţeaua orientată din figura 1.14.

Fig.1.14.

Matricele $M, B, C, \overline{M}$ sunt următoarele:

$$M = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 25 & 35 & 0 & 0 \\ 0 & 0 & 0 & 15 & 0 \\ 0 & 45 & 0 & 0 & 0 \\ 0 & 0 & 15 & 0 & 45 \\ 0 & 0 & 25 & 35 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 30 & 50 & 0 & 0 \\ 0 & 0 & 0 & 40 & 0 \\ 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 30 & 0 & 60 \\ 0 & 0 & 20 & 50 & 0 \end{bmatrix}, \overline{M} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 \end{bmatrix}$$

Matricele M, B, C au fiecare n^2 elemente şi numai m elemente nenule. Matricea $\overline{M}$ are nm elemente şi numai $2m$ elemente nenule (pe fiecare coloană câte

2 elemente nenule, un 1 şi un -1). Rezultă că reprezentarea prin matrice asociate reţelei este eficientă numai în cazul reţelelor dense ($m \gg n$). Pentru reţelele rare ($m \ll n^2$) este eficient să utilizăm liste de adiacenţă sau liste de incidenţă.

Lista nodurilor adiacente către exterior nodului i este $V^+(i) = \{j|(i,j) \in A\}$ şi lista arcelor incidente către exterior nodului i este $E^+(i) = \{(i,j)|j \in V^+(i)\}$.

Considerăm că listele $V^+(i)$ şi $E^+(i)$ sunt ordonate, şi avem $|V^+(i)| = |E^+(i)| = \rho^+(i),\ i = 1, \ldots, n$.

Reprezentarea prin *liste de adiacenţă* poate fi utilizată în una din următoarele două variante:

- reprezentarea cu ajutorul tablourilor;
- reprezentarea cu ajutorul listelor simplu înlănţuite.

Reprezentarea cu ajutorul *tablourilor* constă în a defini două tablouri unidimensionale. Primul tablou, notat P, are $n+1$ elemente şi reprezintă o listă de pointeri. Tabloul P se defineşte astfel:$P(1) = 1, P(i+1) = P(i) + \rho^+(i), \quad i = 1, \ldots, n$. Al doilea tablou, notat V, are m elemente şi reprezintă o listă de noduri. Tabloul V conţine nodurile j din lista $V^+(i)$, în ordinea stabilită, între locaţiile $P(i)$ şi $P(i+1) - 1, \quad i = 1, \ldots, n$.

Reprezentarea cu ajutorul tablourilor poate fi realizată şi prin utilizarea unui tablou bidimensional

$$T = \begin{bmatrix} T(1,1) & \ldots & T(1,p) \\ T(2,1) & \ldots & T(2,p) \end{bmatrix}$$

unde $p = n + m$. Elementele $T(1,k), \quad k = 1, \ldots, p$ sunt noduri şi elementele $T(2,k), \quad k = 1, \ldots, p$ sunt pointeri. Prima linie a tabloului T se defineşte astfel:

$$T(1,k) = \begin{cases} k, & \text{pentru} \quad k = 1, \ldots, n, \\ V(k-n), & \text{pentru} \quad k = n+1, \ldots, p \end{cases}$$

Elementul $T(1,k)$ reprezintă:

- nodul k, dacă $1 \le k \le n$,
- un nod j din $V^+(i)$, dacă $n + P(i) \le k \le n + P(i+1),\ i = 1, \ldots, n$.

A doua linie a tabloului T se defineşte astfel:

$$T(2,k) = \begin{cases} n + P(k), & \text{dacă} \quad V^+(k) \neq \emptyset, \text{ pentru} \quad k = 1, \ldots, n \\ 0, & \text{dacă} \quad V^+(k) = \emptyset, \text{ pentru} \quad k = 1, \ldots, n \\ k+1, & \text{dacă} \quad T^(1,k) = j \text{ nu este ultimul nod din } V^+(i) \\ & \text{pentru} \quad k = n + P(i), \ldots, n + P(i+1) - 1,\ i = 1, \ldots, n \\ 0, & \text{dacă} \quad T^(1,k) = j \text{ este ultimul nod din } V^+(i) \\ & \text{pentru} \quad k = n + P(i), \ldots, n + P(i+1) - 1,\ i = 1, \ldots, n \end{cases}$$

Elementul $T(2,k) \neq 0$ reprezintă adresa (coloana) din $T(1,k')$, adică $k' = T(2,k)$:

- a primului nod j din $V^+(k)$, dacă $1 \leq k \leq n$,

- a următorului nod j' a lui $j = T(1,k)$ din $V^+(i)$, dacă $n + P(i) \leq k < n + P(i+1) - 1,\ i = 1, \ldots, n$.

Reprezentarea cu ajutorul *listelor simplu înlănţuite* constă din n liste, fiecare listă corespunde la un nod i şi are $\rho^+(i)$ locaţii. Fiecare locaţie corespunde unui arc (i,j) şi conţine mai multe câmpuri: un câmp pentru memorarea nodului j, un câmp pentru memorarea pointerului care indică legătura la următoarea locaţie şi eventual un câmp sau două câmpuri pentru $b(i,j)$ sau/şi $c(i,j)$. Dacă locaţia este ultima din lista înlănţuită atunci prin convenţie în câmpul pointerului punem valoarea 0. Deoarece avem n liste, una pentru fiecare nod i cu $\rho^+(i)$ locaţii, este necesar un tablou unidimensional, notat CAP, cu n elemente ce conţine pointeri care indică prima locaţie din fiecare listă înlănţuită. Dacă $\rho^+(i) = 0$ atunci $CAP(i) = 0$.

Exemplul 1.22. Considerând reţeaua din figura 1.14 avem următoarele reprezentări cu ajutorul tablourilor şi cu ajutorul listelor simplu înlăţuite:
$V^+(1) = \{2,3\},\ \rho^+(1) = 2, V^+(2) = \{4\},\ \rho^+(2) = 1,\ V^+(3) = \{2\},\ \rho^+(3) = 1,\ V^+(4) = \{3,5\}, \rho^+(4) = 2, V^+(5) = \{3,4\}, \rho^+(5) = 2; P = (1,3,4,5,7,9), V = (2,3,4,2,3,5,3,4); n = 5, m = 8, n + m = 5 + 8 = 13.$

$$T = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 2 & 3 & 4 & 2 & 3 & 5 & 3 & 4 \\ 6 & 8 & 9 & 10 & 12 & 7 & 0 & 0 & 0 & 11 & 0 & 13 & 0 \end{bmatrix}$$

Reprezentarea prin liste de adiacenţă revine la descrierea matricei de adiacenţă linie cu linie. În mod similar se poate descrie matricea de incidenţă coloană cu coloană utilizând liste de incidenţă.

Reprezentarea prin *liste de incidenţă* constă în utilizarea tabloului unidimensional P al pointerilor şi a unui tablou bidimensional notat E. Dacă reprezentăm o reţea $G = (N, A, b)$ sau o reţea $G = (N, A, c)$ atunci tabloul E are trei linii şi m coloane şi dacă reprezentăm reţeaua $G = (N, A, b, c)$ atunci tabloul E are patru linii şi m coloane. Fiecare coloană k, corespunde unui arc (i,j) şi anume $E(1,k) = i, E(2,k) = j, E(3,k) = b(i,j)$ sau $c(i,j)$ dacă reţeaua este $G = (N, A, b)$ sau $G = (N, A, c)$ şi $E(1,k) = i, E(2,k) = j, E(3,k) = b(i,j), E(4,k) = c(i,j)$ dacă reţeaua este $G = (N, A, b, c)$. Tabloul E conţine arcele din $E^+(i)$, în ordinea stabilită, şi informaţiile asociate acestor arce, de la coloana $P(i)$ inclusiv până la coloana $P(i+1) - 1$ inclusiv.

Exemplul 1.23. Considerăm reţeaua reprezentată în figura 1.14 Tablourile P şi E sunt următoarele:

$$P = (1,3,4,5,7,9)$$

$$E = \begin{bmatrix} 1 & 1 & 2 & 3 & 4 & 4 & 5 & 5 \\ 2 & 3 & 4 & 2 & 3 & 5 & 3 & 4 \\ 25 & 35 & 15 & 45 & 15 & 45 & 25 & 35 \\ 30 & 50 & 40 & 10 & 30 & 60 & 20 & 50 \end{bmatrix}$$

Un alt avantaj al reprezentării prin liste de adiacenţă sau liste de incidenţă este că permite reprezentarea reţelelor care conţin arce paralele, adică arce care au aceeaşi extremitate iniţială şi aceeaşi extremitate finală.

Reprezentarea prin liste de adiacenţă este avantajoasă când este implementată într-un limbaj care are posibilitatea să lucreze cu liste înlănţuite (PASCAL, C). În acest caz topologia reţelei se poate modifica uşor. Reprezentarea prin liste de incidenţă are avantajul că are nevoie de mai puţină memorie decât reprezentarea prin liste de adiacenţă. Alegerea unei reprezentări depinde de problema de rezolvat şi de limbajul ales.

O reţea neorientată $G = (N, A, b, c)$ se transformă într-o reţea orientată $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{b}, \widetilde{c})$ cu $\widetilde{N} = N, \widetilde{A} = \{(i, j), (j, i) \,|\, [i,\ j] \in A\}$, funcţiile $\widetilde{b}$ şi $\widetilde{c}$ sunt definite în raport cu problema de rezolvat şi $\widetilde{G}$ se reprezintă prin una din metodele descrise mai sus.

1.5 Noţiuni de complexitatea algoritmilor în teoria grafurilor

În acest paragraf se va nota cu $\lceil r \rceil$ cel mai mic întreg mai mare sau egal cu r pentru $r \in \Re$.

Fie un digraf $G = (N, A)$ cu $|N| = n$ noduri şi $|A| = m$ arce. Se consideră funcţia valoare $b : A \to \Re$ şi funcţia capacitate $c : A \to \Re$. Funcţia valoare b reprezintă fie funcţia lungime, fie funcţia cost etc. Fie $\overline{b} = max\{b(x, y)|(x, y) \in A\}$ şi $\overline{c} = max\{c(x, y)|(x, y) \in A\}$.

Definiţia 1.28. Prin *problemă* se înţelege o funcţie total definită $P : I \to F$, unde I este mulţimea informaţiilor iniţiale (datele de intrare ale problemei) şi F este mulţimea informaţiilor finale (datele de ieşire ale problemei).

Se presupune că mulţimile I şi F sunt nevide şi cel mult numărabile.

Exemplul 1.24. În reţeaua $G = (N, A, b)$ se poate formula problema drumului minim, iar în reţeaua $G = (N, A, c)$ problema fluxului maxim.

Definiţia 1.29. Se numeşte *instanţă* a problemei P, precizarea problemei $P(i)$ pentru o valoare specificată $i \in I$.

Pentru o instanţă $P(i)$ se va folosi şi notaţia p, iar prin abuz de notaţie uneori vom scrie $p \in P$.

Exemplul 1.25. În problema drumului minim în reţeaua $G = (N, A, b)$, pentru a defini o instanţă a acestei probleme este necesar să specificăm topologia digrafului $G = (N, A)$, nodurile sursă şi destinaţie şi funcţia b.

Un *algoritm* este o procedură care rezolvă pas cu pas o instanţă $p, p \in P$.

Definiţia 1.30. Se spune că *algoritmul α rezolvă problema P* dacă algoritmul determină soluţia oricărei instanţe $p \in P$.

Un algoritm α care rezolvă instanţa p va porni de la o codificare a informaţiei iniţiale $i \in I$ şi va furniza o codificare a rezultatului.

Definiţia 1.31. Se numeşte *dimensiunea problemei P* un număr natural notat $d(p), p \in P$, care reprezintă lungimea unei codificări binare a informaţiei iniţiale.

În general, $d(p)$ este un număr natural care reprezintă dimensiunea structurală a informaţiei iniţiale, deoarece lungimea codificării binare a unei informaţii iniţiale va fi mărginită de o funcţie având ca argument dimensiunea sa structurală.

Exemplul 1.26. Să considerăm o problemă care are ca dată de intrare un digraf $G = (N, A)$ reprezentat prin matricea sa de adiacenţă M. Pentru această problemă trebuie $\lceil \log n \rceil$ biţi pentru codul numărului n şi n^2 biţi pentru reprezentarea matricei. Dimensiunea problemei, este $\lceil \log n \rceil + n^2$. Pentru n suficient de mare se poate considera dimensiunea problemei egală cu n^2.

Exemplul 1.27. Să considerăm o problemă care are ca dată de intrare, o reţea $G = (N, A, b)$. Presupunem că pentru reprezentarea digrafului $G = (N, A)$ se folosesc listele de adiacenţă P (lista de pointeri) şi V (lista nodurilor). Dacă notăm cu $\rho^+(i)$ semigradul exterior al nodului $i \in N$, atunci lista P este un tablou unidimensional ce are $n + 1$ elemente cu $P(1) = 1$ şi $P(i + 1) = P(i) + \rho^+(i), \quad i = 1, 2, \ldots, n$. Lista V este un tablou unidimensional ce are m elemente şi $V(P(i))$ până la $V(P(i + 1) - 1)$ conţin succesorii nodului i. Rezultă $1 \leq P(i) \leq m + 1,\ i = 1, \ldots, n + 1$ şi $1 \leq V(i) \leq n,\ i = 1, \ldots, m$. Deci pentru a descrie această problemă sunt necesari:

$\lceil \log n \rceil$ biţi pentru codificarea lui n;
$\lceil \log m \rceil$ biţi pentru codificarea lui m;
$(n + 1)\lceil \log(m + 1) \rceil$ biţi pentru codificarea elementelor tabloului P;
$m\lceil \log n \rceil$ biţi pentru codificarea elementelor tabloului V;
$m\lceil \log \overline{b} \rceil$ biţi pentru codificarea valorilor numerice $b(x, y), (x, y) \in A$.

Rezultă că dimensiunea problemei este $\lceil \log n \rceil + \lceil \log m \rceil + (n + 1)\lceil \log(m + 1) \rceil + m\lceil \log n \rceil + m\lceil \log \overline{b} \rceil$. Pentru m şi n suficienţi de mari se poate considera că problema are dimensiunea $m\lceil \log n \rceil$.

În continuare se vor prezenta diferite abordări ale complexităţii unui algoritm.

Resursele de calcul asociate execuţiei unui algoritm sunt spaţiul de memorie şi timpul necesar de execuţie. Ne vom ocupa numai de resursa timp, deoarece progresele tehnologice din ultima perioadă conduc la o scădere a importanţei memoriei folosite de algoritm.

Vom nota $T'_\alpha(p)$ timpul de execuţie necesar algoritmului α pentru rezolvarea instanţei $p, p \in P$. Acest timp reprezintă numărul paşilor efectuaţi de algoritm

pentru rezolvarea instanţei p. Un pas al unui algoritm α este o operaţie elementară (instrucţiune elementară). În general, operaţiile elementare sunt: operaţii de atribuire, operaţii aritmetice (+, -, $\star$, /), operaţii logice (comparaţii). Se presupune, că execuţia unei instrucţiuni elementare nu depinde de mărimea operanzilor şi de timpul de memorare a rezultatelor. Aceasta înseamnă că resursa timp este studiată independent de sistemul de calcul pe care se face implementarea algoritmului.

În literatura de specialitate există trei tipuri de abordări ale complexităţii unui algoritm α: analiza empirică, analiza cazului mediu şi analiza cazului cel mai defavorabil.

Obiectivul *analizei empirice* constă în studierea comportării în practică a unui algoritm. Se scrie un program pentru algoritmul respectiv şi se testează performanţele programului pe diferite clase de instanţe ale problemei. Această analiză are câteva dezavantaje majore: (1) performanţele unui program depind de limbajul de programare, de calculatorul folosit şi de priceperea programatorului care a scris programul; (2) de obicei această analiză este mare consumatoare de timp şi este scumpă; (3) compararea algoritmilor prin această analiză poate să conducă la rezultate eronate.

Obiectivul *analizei cazului mediu* constă în studierea comportării în medie a unui algoritm, care presupune rezolvarea următoarelor două etape: (a) precizarea unei distribuţii de probabilitate pe mulţimea instanţelor $p, p \in P$; (b) determinarea mediei variabilei aleatoare $T'_\alpha(p), T^m_\alpha(k) = M(\{T'_\alpha(p) \mid p \in P, d(p) = k\})$. Analiza cazului mediu are, de asemenea, dezavantaje majore: (1) în general, este dificil să se determine o distribuţie de probabilitate corectă pe mulţimea instanţelor $p, p \in P$; (2) această analiză depinde de distribuţia de probabilitate aleasă; (3) în general, determinarea mediei $T^m_\alpha(k)$ se reduce la calculul unor sume finite care sunt evaluate cu mari dificultăţi. Totuşi această analiză este folosită în cazul când algoritmul are performanţe bune pentru majoritatea instanţelor şi pentru un număr mic de instanţe, care apar rar în practică, algoritmul funcţionează prost sau foarte prost.

Analiza *cazului cel mai defavorabil* elimină multe din dezavantajele prezentate mai sus. Această analiză constă în a determina $T_\alpha(k) = sup\{T'_\alpha(p) | p \in P, d(p) = k\}$. Rezultă că analiza cazului cel mai defavorabil: (1) furnizează o margine superioară pentru numărul operaţiilor elementare (instrucţiunilor elementare) necesare algoritmului α pentru rezolvarea oricărei instanţe $p, p \in P$; (2) este independentă de calculatorul pe care se face implementarea algoritmului; (3) face posibilă compararea algoritmilor; (4) are şi avantajul că $T_\alpha(k)$ se determină mai uşor decât $T^m_\alpha(k)$. Totuşi, analiza cazului cel mai defavorabil nu este perfectă. Un dezavantaj major este faptul că $T_\alpha(k)$ poate fi determinată de instanţe "patologice" care apar destul de rar în practică. Dar avantajele acestei analize

depăşesc dezavantajele şi de aceea este cea mai folosită metodă pentru a măsura performanţele unui algoritm.

Deoarece determinarea exactă a lui $T_\alpha(k)$ este uneori dificilă, iar pe de altă parte, aceasta ar însemna considerarea unor detalii de implementare, se vor căuta margini superioare şi inferioare pentru $T_\alpha(k)$, se va studia comportarea sa asimptotică.

În cele ce urmează se vor introduce notaţiile O, Ω, Θ şi se vor defini algoritmii polinomiali şi cei exponenţiali.

Fie funcţia $f : \mathcal{N} \to \mathcal{N}$. Se folosesc următoarele notaţii:

$$O(f) = \{g | g : \mathcal{N} \to \mathcal{N},\ \exists\, r_1 \in \Re_+^*,\ \exists k_0 \in \mathcal{N} : g(k) \leq r_1 f(k),\ \forall\, k \geq k_0\};$$

$$\Omega(f) = \{g | g : \mathcal{N} \to \mathcal{N},\ \exists\, r_2 \in \Re_+^*,\ \exists k_0 \in \mathcal{N} : g(k) \geq r_2 f(k),\ \forall\, k \geq k_0\};$$

$$\Theta(f) = \{g \mid g : N \to N,\ g \in O(f) \cap \Omega(f)\}.$$

Se obişnueşte ca în loc de a scrie $g \in O(f)$ să se folosească notaţia $g = O(f)$ (similar pentru Ω şi Θ).

Definiţia 1.32. Se numeşte *complexitatea timp* (sau simplu *complexitate*) a algoritmului α comportarea asimptotică a lui $T_\alpha(k)$.

Definiţia 1.33. Se spune că o problemă algoritmică P are *complexitatea O(f(k))* dacă există un algoritm α care rezolvă problema P şi algoritmul α are complexitatea $T_\alpha(k) = O(f(k))$.

Definiţia 1.34. Se spune că o problemă algoritmică P are *complexitatea* $\Omega(f(k))$ dacă orice algoritm α care rezolvă problema P are complexitatea $T_\alpha(k) = \Omega(f(k))$.

Definiţia 1.35. Se spune că un algoritm α pentru rezolvarea problemei P este *optimal* dacă problema P are complexitatea $\Omega(T_\alpha(k))$.

Demonstrarea optimalităţii unui algoritm α pentru o problemă P este o activitate foarte complicată. Există foarte puţine rezultate de acest fel.

Definiţia 1.36. Se numeşte *algoritm polinomial* un algoritm α cu complexitatea $O(f(k))$ unde $f(k)$ este un polinom în k. Un algoritm care nu este polinomial se numeşte *exponenţial*.

Exemplul 1.28. Presupunem că o operaţie elementară necesită pentru execuţie 10^{-6} secunde, adică $O(1) = 10^{-6}$ secunde. În tabelul de mai jos sunt prezentate complexităţile timp pentru câteva funcţii, unde m înseamnă minute, z înseamnă zile şi s înseamnă secole.

n	n	n^5	2^n	n^n
20	$0,33 \times 10^{-6}$m	$5,33 \times 10^{-2}$m	$1,66 \times 10^{-2}$ m	3×10^{10}s
40	$0,66 \times 10^{-6}$ m	$0,17 \times 10^{1}$ m	$1,27 \times 10$ z	...

Exemplul 1.29. Să considerăm graful $G = (N, A)$ cu $N = \{1, \ldots, n\}, A = \{1, \ldots, m\}$ reprezentat prin matricea sa de adiacenţă M. Unele probleme de

teoria grafurilor necesită calcularea matricei M^{n-1}, unde prin M^i notăm, în acest exemplu, puterea booleană de ordinul i a matricei M. Deci problema P constă în calculul matricei M^{n-1}.

Dintr-un exemplu prezentat mai sus rezultă că dimensiunea problemei este n^2.

Dacă avem de efectuat produsul boolean a două matrice boolene (cu elemente 0 şi 1) B şi C pătratice de dimensiune n, atunci pentru calculul matricei produs $D = B\dot{\times}C$ trebuie determinate toate elementele $d_{ij} = b_{i1}\dot{\times}c_{1j}\dot{+}\ldots\dot{+}b_{in}\dot{\times}c_{nj}$, unde $\dot{\times}, \dot{+}$ reprezintă înmulţirea booleană, respectiv adunarea booleană. Convenim să considerăm ca operaţie elementară perechea ordonată $(\dot{\times}, \dot{+})$. Rezultă că, pentru calculul unui element d_{ij} sunt necesare n operaţii elementare. Deci pentru calculul celor n^2 elemente d_{ij} sunt necesare n^3 operaţii elementare. Fie:

$$n-1 = \sum_{i=0}^{k} a_i\, 2^i, a_k \neq 0, a_i \in \{0,1\}.$$

În acest caz avem $M^{n-1} = M^{a_0}\dot{\times}(M^2)^{a_1}\dot{\times}\ldots\dot{\times}(M^{2^k})^{a_k}$. Acest produs boolean conţine cel mult $\lceil \log(n-1) \rceil$ înmulţiri boolene, deoarece dacă $a_i = 0$ atunci $(M^{2^i})^{a_i} = U$ (matricea unitate) şi nu se mai efectuează înmulţirea. Considerăm cazul cel mai defavorabil cu $a_i = 1, \quad i = 0, \ldots, k, \quad k = \lceil \log(n-1) \rceil - 1$ $(2^k < n-1 < 2^{k+1})$ şi $k = \lceil \log(n-1) \rceil$ $(n-1 = 2^k)$. Algoritmul necesită atunci
1) calculul matricelor booleene:

$$M, M^2 = M\dot{\times}M, \ldots, M^{2^k} = M^{2^{k-1}}\dot{\times}M^{2^{k-1}}$$

ce reprezintă k înmulţiri booleene de matrice booleene;
2) calculul matricelor booleene:

$$M^3 = M\dot{\times}M^2, \quad M^7 = M^3\dot{\times}M^4, \quad M^{15} = M^7\dot{\times}M^8, \ldots$$

ce reprezintă k înmulţiri booleene de matrice booleene.

Deci pentru a calcula M^{n-1} trebuie să efectuăm, în cazul cel mai defavorabil, $2k$ înmulţiri booleene de matrice booleene. Cum pentru înmulţirea booleană a două matrice booleene sunt necesare n^3 operaţii elementare, rezultă că pentru calculul matricei M^{n-1} sunt necesare $2n^3k$ operaţii elementare. Deoarece $k = \lceil \log(n-1) \rceil - 1$ şi dimensiunea problemei este n^2, rezultă că algoritmul pentru calculul matricei M^{n-1} are complexitatea $O(n^{3/2}\log n)$.

Referitor la problemele din teoria grafurilor, în practica curentă, se exprimă complexitatea unui algoritm în funcţie de $m, n, \overline{b}, \overline{c}$. Cum problema este din teoria grafurilor se consideră că algoritmul pentru calculul matricei M^{n-1} are complexitatea $O(n^3 \log n)$.

În teoria grafurilor, referitor la complexitatea algoritmilor, avem următoarele noţiuni specifice. Dacă complexitatea algoritmului este $O(f(n, m, \log \overline{b}, \log \overline{c}))$, unde $f(n, m, \log \overline{b}, \log \overline{c})$ este o funcţie polinomială de $n, m, \log \overline{b}, \log \overline{c}$, atunci se spune că algoritmul este *polinomial*. Un algoritm polinomial se spune că este algoritm *tare polinomial* dacă f este o funcţie polinomială numai de n şi m şi algoritm *slab polinomial* altfel. În general, algoritmii tare polinomial sunt preferaţi faţă de algoritmii slab polinomial, deoarece ei pot rezolva probleme cu valori arbitrar de mari pentru funcţiile b şi c. Remarcăm faptul că definiţia algoritmului polinomial din teoria grafurilor se încadrează în definiţia algoritmului polinomial pentru cazul general (Definiţia 1.36). De asemenea, conform definiţiei 1.36, un algoritm din teoria grafurilor este *exponenţial* dacă nu este polinomial. De exemplu, $O(2^n), O(n!), O(m \cdot n \cdot \overline{c})$ sunt complexităţi pentru algoritmi exponenţiali. Se spune că un algoritm este *pseudopolinomial* dacă f este o funcţie polinomială de $m, n, \overline{b}, \overline{c}$. Algoritmii pseudopolinomiali constituie o subclasă importantă a clasei algoritmilor exponenţiali.

Evident, că sunt preferaţi algoritmii polinomiali faţă de algoritmii exponenţiali şi în cadrul algoritmilor exponenţiali sunt preferaţi algoritmii pseudopolinomiali.

În continuare se vor prezenta anumite reguli de calcul pentru O, Ω, Θ. Mai întâi să precizăm că relaţiile O, Ω, Θ pot fi considerate ca relaţii între funcţii: O este relaţia "este dominată asimptotic de", Ω este relaţia "domină asimptotic pe" şi Θ este relaţia "are acelaşi ordin de mărime ca". Se constată uşor că relaţiile O şi Ω sunt relaţii de preordine (reflexive şi tranzitive) şi că relaţia Θ este o relaţie de echivalenţă (reflexivă, simetrică şi tranzitivă). Regulile de calcul pentru O, Ω, Θ sunt următoarele:

1) Reflexivitatea relaţiilor O, Ω, Θ:
   $a) f = O(f), \quad b) f = \Omega(f), \quad c) f = \Theta(f).$
2) Simetria relaţiei Θ:
   $g = \Theta(f)$ implică $f = \Theta(g)$.
3) Tranzitivitatea relaţiilor O, Ω, Θ:
   $a) g = O(f)$ şi $f = O(h)$ implică $g = O(h)$,
   $b) g = \Omega(f)$ şi $f = \Omega(h)$ implică $g = \Omega(h)$,
   $c) g = \Theta(f)$ şi $f = \Theta(h)$ implică $g = \Theta(h)$.
4) Fie $c \in \Re_+^*$:
   $a) g = O(f)$ implică $cg = O(f)$,
   $b) g = \Omega(f)$ implică $cg = \Omega(f)$,
   $c) g = \Theta(f)$ implică $cg = \Theta(f)$.
5) Fie $g_1 \geq g_2$ sau $g_1 \leq g_2$:
   $a) g_1 = O(f_1)$ şi $g_2 = O(f_2)$ implică $g_1 + g_2 = O(\max(f_1, f_2))$,
   $b) g_1 = \Omega(f_1)$ şi $g_2 = \Omega(f_2)$ implică $g_1 + g_2 = \Omega(\min(f_1, f_2))$,
   $c) g_1 = \Theta(f_1)$ şi $g_2 = \Theta(f_2)$ implică $g_1 + g_2 = \Theta(\max(f_1, f_2))$.

6) Fie $g_1 \geq g_2$:
$a)g_1 = O(f_1)$ şi $g_2 = O(f_2)$ implică $g_1 - g_2 = O(f_1)$,
$b)g_1 = \Omega(f_1)$ şi $g_2 = \Omega(f_2)$ implică $g_1 - g_2 = \Omega(f_2)$,
$c)g_1 = \Theta(f_1), g_2 = \Theta(f_2)$ şi $f_1 = \Omega(f_2), f_2 = O(f_1)$ implică $g_1 - g_2 = \Theta(f_1)$.
7) Fie $g_1 \geq g_2$ sau $g_1 \leq g_2$:
$a)g_1 = O(f_1)$ şi $g_2 = O(f_2)$ implică $g_1 \cdot g_2 = O(f_1 \cdot f_2)$,
$b)g_1 = \Omega(f_1)$ şi $g_2 = \Omega(f_2)$ implică $g_1 \cdot g_2 = \Omega(f_1 \cdot f_2)$,
$c)g_1 = \Theta(f_1)$ şi $g_2 = \Theta(f_2)$ implică $g_1 \cdot g_2 = \Theta(f_1 \cdot f_2)$.

Aceste reguli sunt consecinţe evidente ale definiţiilor.

Deseori se pot compara funcţiile f şi g calculând

$$\lim_{n\to\infty} \frac{g(n)}{f(n)}.$$

Se obţin următoarele proprietăţi:

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = c > 0 \quad \text{implică} \quad g = \Theta(f) \tag{1.1}$$

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0 \quad \text{implică} \quad g = O(f) \quad \text{şi } g \neq \Theta(f) \tag{1.2}$$

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty \quad \text{implică} \quad g = \Omega(f) \quad \text{şi } g \neq \Theta(f) \tag{1.3}$$

Aceste proprietăţi rezultă din definiţia limitei: de exemplu pentru proprietatea (1.1) avem: $\forall \epsilon > 0, \exists n_0 \in \mathcal{N}$ astfel încât $\forall n \geq n_0$ avem $\left|\frac{g(n)}{f(n)} - c\right| < \epsilon$ sau $\forall \epsilon > 0, \ni n_0 \in \mathcal{N}$ astfel încât $\forall n \geq n_0$ avem

$$(c - \epsilon)f(n) < g(n) < (c + \epsilon)f(n).$$

Reciprocile proprietăţilor (1.1), (1.2), (1.3) sunt false. În cazul când nu există limită, trebuie revenit la definiţiile O, Ω, Θ pentru a putea compara g şi f. De exemplu, dacă $f(n) = 2n, g(n) = n$ pentru $n = 2k+1$ şi $g(n) = 2n$ pentru $n = 2k$, atunci limita de mai sus nu există, dar $g \in \Theta(f)$ deoarece pentru orice n avem $\frac{1}{2}f(n) \leq g(n) \leq f(n)$.

1.6 Aplicaţii şi comentarii bibliografice

1.6.1 Reţele de comunicaţii

O reţea de comunicaţie se poate descrie printr-o reţea $G = (N, A, b, c)$. Astfel, pot fi descrise traseele dintre localităţi pe şosele, autostradă sau cale ferată,

traseele din localităţi ale autobuzelor, troleibuzelor sau tramvaielor, traseele aeriene sau maritime, reţelele cristalografice, geografice, topografice, cartografice, cadastrale, hidrografice, de irigaţii, de drenaj, de alimentare cu apă, gaze sau energie electrică, de termoficare, telefonice, telecomunicaţii, telegrafice, radio, televiziune, calculatoare. De asemenea repartiţiile de materiale sunt reţele de comunicaţii, fluxuri tehnologice, programele de investiţii, de organizare a muncii, sistemul informaţional. Totodată sunt reţele aparatul circulator al sângelui, sistemul nervos, sistemul limfatic etc.

Exemplul 1.30. În figura 1.15 este reprezentată reţeaua $G = (N, A, b)$ a traseelor pe şosele din judeţul Braşov, unde fiecare nod $x \in N$ reprezintă un oraş din judeţ, o muchie $[x, y] \in A$ reprezintă faptul că oraşele x, y sunt conectate printr-o şosea modernizată şi b reprezintă funcţia distanţă. S-a considerat reţeaua $G = (N, A, b)$ neorientată, deoarece pe o şosea $[x, y]$ se circulă în ambele sensuri. Pe fiecare muchie s-a trecut distanţa în kilometri.

Fig.1.15.
B - Braşov; $\overline{B}$ - Bran; C - Codlea; F - Făgăraş; P - Predeal; R - Râşnov; $\overline{R}$ - Rupea; S - Săcele; V - Victoria; Z - Zărneşti

Exemplul 1.31. În digraful din figura 1.16. este schematizată circulaţia sângelui în corpul omenesc.

Fig.1.16.
1 - auricul drept; 2 - ventricul drept; 3 - auricul stâng;
4 - ventricul stâng; 5 - plămâni; 6 - membre superioare;
7 - cap; 8 - trunchi celiac; 9 - membre inferioare.

Dacă pentru o persoană ar exista arcul $(1, 3)$, atunci această persoană ar avea boala lui Roger, iar dacă pentru o persoană se instituie comunicarea directă de la nodul 4 la nodul 2, în digraf ar apărea arcul $(4, 2)$, atunci persoana decedează.

Multe alte aplicaţii ale grafurilor vor fi prezentate în capitolele următoare.

1.6.2 Comentarii bibliografice

Deşi erau cunoscute, de foarte multă vreme, probleme care conduceau implicit la grafuri, evident, fără ca să se cunoscă prezenţa acestora, oficial, actul de naştere al Teoriei grafurilor poate fi considerat rezolvarea problemei celor şapte poduri

din Königsberg de către Euler în anul 1736. Oraşul Königsberg este aşezat pe malurile râului Pregel şi pe două insule ale acestuia, formând, pe acea vreme, patru cartiere. Aceste cartiere erau legate prin şapte poduri. În plimbările lor, localnicii încercau să traverseze toate cele şapte poduri, astfel încât plecând dintr-un anumit cartier să se înapoieze în acesta trecând doar o singură dată peste fiecare pod. Dar acest lucru nu izbuteau să-l facă. Euler, pe atunci instalat în Rusia, captivat de această problemă, a considerat drumul ce trebuie parcurs ca un graf şi a demonstrat că un astfel de drum nu poate exista. De la această rezolvare s-a dat denumirea de drumuri (lanţuri) euleriene drumurilor (lanţurilor) care parcurg arcele (muchiile) grafului o singură dată.

O altă problemă celebră se datorează matematicianului irlandez Hamilton. În anul 1859 Hamilton a imaginat şi realizat următorul joc. În fiecare vârf al unui dodecaedru regulat s-a bătut câte un cui pe floarea căruia era scris numele unui oraş. Problema consta în a se întinde o sfoară între aceste cuie astfel încât fiecare oraş să fie legat de celelalte oraşe o singură dată. Proiectat pe plan dodecaedrul reprezintă un graf în care vârfurile şi muchiile dodecaedrului sunt nodurile respectiv muchiile grafului. De la această problemă s-a dat denumirea de drumuri (lanţuri), circuite (cicluri) hamiltoniene drumurilor (lanţurilor), circuitelor (ciclurilor) care trec prin nodurile grafului o singură dată.

Alte probleme celebre în teoria grafurilor sunt: problema barcagiului, problema celor trei case şi trei fântâni, problema colorării hărţilor, problema mozaicului etc.

Datorită aplicaţiilor teoriei grafurilor în toate domeniile de activitate şi apariţiei calculatoarelor, algoritmica grafurilor s-a bucurat de un interes deosebit din partea cercetătorilor după anul 1950. Următoarele cărţi tratează algoritmica grafurilor şi servesc ca un ghid pentru cei interesaţi în literatura de specialitate:

1. *Flows in Networks* (**Ford and Fulkerson** (1962)).
2. *Programmes, Jeux et Réseaux de Transport* (**Berge and Ghouila-Houri** (1962)).
3. *Finite Graphs and Networks* (**Busacker and Saaty** (1965)).
4. *Network Flow, Transportation and Scheduling* (**Iri** (1969)).
5. *Integer Programming and Network Flows* (**Hu** (1969)).
6. *Communication, Transmission and Transportation Networks* (**Frank and Frisch** (1971)).
7. *Flows in Transportation Networks* (**Potts and Oliver** (1972)).
8. *Graph Theory: An Algorithmic Approach* (**Cristophides** (1975)).

9. *Combinatorial Optimization: Networks and Matroids* (**Lawler (**1976)).
10. *Graph Algorithms* (**Even (**1979)).
11. *Probleme de Combinatorică şi Teoria Grafurilor* (**Tomescu (**1981)).
12. *Combinatorial Optimization: Algorithms and Complexity* (**Papadimitriou and Steigglitz (**1982)).
13. *Discrete Optimization Algorithms* (**Syslo, Deo and Kowalik (**1983)).
14. *Data Structures and Network Algorithms* (**Tarjan (**1983)).
15. *Data Structures and Algorithms* (**Mehlhorn (**1984)).
16. *Graphs and Algorithms* (**Gondran and Minoux**(1984)).
17. *Programming in Networks and Graphs* (**Derigs** (1988)).
18. *Linear Programming and Network Flows* (**Bazaraa, Jarvis and Sherali** (1990)).
19. *Theory of Nets: Flows in Networks* (**Chen** (1990)).
20. *Algorithmic Aspects of Flows in Networks* (**Ruhe** (1991)).
21. *Linear Network Optimization: Algorithms and Codes* (**Bertsekas** (1991)).
22. *Network Programming* (**Murty** (1992)).
23. *Optimization Algorithms for Networks and Graphs* (**Evans and Minieka** (1992)).
24. *Tehnici de Bază în Optimizarea Combinatorie* (**Croitoru** (1992)).
25. *Network Flows: Theory, Algorithms and Applications* (**Ahuja, Magnanti and Orlin** (1993)).
26. *Graph Coloring Problems* (**Jensen and Toft** (1995)).
27. *Local Search in Combinatorial Optimization* (**Aarts and Lenstra** (1997)).
28. *Graphs, Networks and Algorithms* (**Jungnickel** (1999)).
29. *Graph Theory and its Applications* (**Gross and Yellen** (1999)).
30. *Digraphs: Theory, Algorthms and Applications* (**Bang-Jensen and Gutin** (2001)).

Capitolul 2

Parcurgeri de grafuri

Problema parcurgerii unui digraf $G = (N, A), \quad N = \{1, \ldots, n\}, \quad A = \{1, \ldots, m\}$ are următoarea formulare: să se genereze mulţimea $W \subset N$ a nodurilor y pentru care există drum de la un *nod sursă* dat s la nodul y în digraful G. Dacă există drum, în digraful G, de la nodul sursă s la nodul y atunci se spune că nodul y este *accesibil* din nodul s.

Algoritmii pe care îi vom prezenta pentru rezolvarea problemei parcurgerii unui digraf G sunt metode sistematice de *vizitare* a nodurilor y accesibile din s. Fiecare iteraţie a execuţiei oricărui algoritm de parcurgere stabileşte pentru fiecare nod apartenenţa la una din următoarele trei *stări*:

- *nevizitat*;
- *vizitat şi neanalizat*, adică un nod vizitat ai cărui succesori au fost parţial vizitaţi;
- *vizitat şi analizat*, adică un nod vizitat ai cărui succesori au fost în totalitate vizitaţi.

Dacă nodul x este vizitat şi neanalizat, există arc (x, y) şi nodul y este nevizitat, atunci se poate *vizita* nodul y. În acest caz se spune că arcul (x, y) este arc *admisibil* şi dacă nodul y este vizitat explorând arcul (x, y) se spune că nodul x este *predecesorul parcurgere* al nodului y.

În acest capitol se vor prezenta următorii algoritmi pentru parcurgerea unui digraf: algoritmul generic, algoritmul BF şi algoritmul DF. Aceşti algoritmi utilizează următoarele notaţii comune:

U mulţimea nodurilor nevizitate;
V mulţimea nodurilor vizitate şi neanalizate;
W mulţimea nodurilor vizitate şi analizate;
p tabloul predecesor, care este unidimensional cu n elemente.

2.1 Parcurgerea generică a grafurilor

Se folosesc notaţiile precizate mai sus şi în plus notăm cu *o tabloul ordine* care este unidimensional şi are n elemente.

Algoritmul parcurgerii generice (algoritmul PG) este următorul:

(1) PROGRAM PG;
(2) BEGIN
(3) $U := N - \{s\}$; $V := \{s\}$; $W := \emptyset$;
(4) FOR toţi $y \in N$ DO $p(y) := 0$;
(5) $k := 1$; $o(s) := 1$;
(6) FOR toţi $y \in U$ DO $o(y) := \infty$;
(7) WHILE $V \neq \emptyset$ DO
(8) BEGIN
(9) se selectează un nod x din V;
(10) IF există $\text{arc}(x, y) \in A$ şi $y \in U$
(11) THEN $U := U - \{y\}$; $V := V \cup \{y\}$;
$p(y) := x$; $k := k + 1$; $o(y) := k$
(12) ELSE $V := V - \{x\}$; $W := W \cup \{x\}$;
(13) END;
(14) END.

Teorema 2.1 *Algoritmul PG este convergent şi la terminarea execuţiei determină mulţimea tuturor nodurilor care sunt accesibile din nodul sursă s în digraful $G = (N, A)$.*

Demonstraţie Din liniile $(10), (11)$ şi (12) ale algoritmului rezultă că toate nodurile introduse în V sunt eliminate după ce sunt analizate. Deci după un număr finit de iteraţii se obţine $V = \emptyset$ şi execuţia algoritmului se opreşte. Pentru a arăta că la terminarea execuţiei, algoritmul determină mulţimea tuturor nodurilor care sunt accesibile din nodul sursă s în digraful $G = (N, A)$, trebuie să arătăm că la terminarea execuţiei algoritmului mulţimea W este:

$$W = \{y | y \in N \quad \text{şi există drum de la } s \text{ la } y\}.$$

Din liniile $(10), (11)$ şi (12) ale algoritmului rezultă că în V sunt introduse numai noduri y care sunt accesibile din s şi că după ce un nod $x \in V$ a fost analizat el este eliminat din V şi introdus în W. Deoarece algoritmul se opreşte când $V = \emptyset$ rezultă că W conţine toate nodurile $y \in N$ care sunt accesibile din s şi introduse în V. Să arătăm că W conţine toate nodurile $y \in N$ care sunt accesibile din s. Prin reducere la absurd să presupunem că există un drum $D = (y_1, y_2, \ldots, y_{k-1}, y_k)$ cu $y_1 = s$, $y_k = y$ în G şi $y \notin W$. Rezultă că $y_k \notin V$. Deoarece $y_k \notin V$ şi $(y_{k-1}, y_k) \in A$ deducem că $y_{k-1} \notin V$, astfel y_k ar fi fost introdus în V. Continuând procedeul vom deduce în final că $s = y_1 \notin V$. Aceasta

contrazice faptul că în linia (3) a algoritmului iniţializăm $V{:=}\{s\}$. Rezultă că $y \in V$, deci $y \in W$ şi teorema este demonstrată. ■

Teorema 2.2 *Algoritmul PG are complexitatea* $O(m)$.

Demonstraţie Din liniile (10), (11) şi (12) ale algoritmului rezultă că fiecare nod al digrafului G este introdus şi eliminat din V cel mult o dată. Deoarece execuţia algoritmului se termină când $V = \emptyset$ deducem că algoritmul execută cel mult $2n$ iteraţii. Fiecare arc $(x, y) \in A$ este explorat cel mult o dată pentru identificarea arcelor admisibile. Deci complexitatea algoritmului PG este $O(m + n) = O(m)$.■

Un digraf $\widehat{G} = \left(\widehat{N}, \widehat{A}\right)$ se numeşte *arborescenţă* cu rădăcina s dacă este fără cicluri şi $\widehat{\rho}^{-}(s) = 0$, $\widehat{\rho}^{-}(x) = 1$ pentru toate nodurile x din $\widehat{N}$ cu $x \neq s$.

Subgraful $G_p = (N_p, G_p)$ cu $N_p = \{y \in N \mid p(y) \neq 0\} \cup \{s\}$, $A_p = \{(p(y), y) \in A \mid y \in N_p - \{s\}\}$ se numeşte *subgraf predecesor* al digrafului $G = (N, A)$. Dacă $N_p = W$ şi G_p este o arborescenţă atunci G_p se numeşte *arborescenţă parcurgere*. Arcele din A_p se numesc *arce arborescenţă*.

Teorema 2.3 *Algoritmul PG determină elementele tabloului p astfel încât subgraful predecesor* $G_p = (N_p, A_p)$ *este o arborescenţă parcurgere.*

Demonstraţie Din liniile (10), (11) şi (12) ale algoritmului rezultă că $p(y) := x$ numai dacă y este accesibil din s. Evident că la terminarea execuţiei algoritmului avem $N_p = W$. Din modul cum sunt definite N_p şi A_p rezultă că G_p este o arborescenţă. Deci subgraful predecesor G_p este o arborescenţă parcurgere a digrafului $G = (N, A)$.■

Observaţia 2.1. Mulţimea W este în general o submulţime a mulţimii nodurilor N. Pentru parcurgerea întregului digraf $G = (N, A)$ se utilizează algoritmul parcurgerii totale generice (algoritmul PTG). Algoritmul PTG este următorul:

```
(1)    PROGRAM PTG;
(2)     BEGIN;
(3)       U := N − {s} ; V := {s} ; W := ∅;
(4)       FOR toţi y ∈ N DO p(y) := 0;
(5)       k := 1; o(s) := 1;
(6)       FOR toţi y ∈ U DO o(y) := ∞;
(7)       WHILE W ≠ N DO
(8)       BEGIN
(9)          WHILE V ≠ ∅ DO
(10)          BEGIN
(11)             se selectează un nod x din V;
(12)             IF există arc (x, y) ∈ A şi y ∈ U
(13)                THEN U := U − {y} ; V := V ∪ {y} ; p(y) := x;
                              k := k + 1; o(y) := k
(14)                ELSE V := V − {x} ; W := W ∪ {x} ;
(15)          END;
```

(16) se selectează $s \in U; U := U - \{s\}; V := \{s\};$
$k := k + 1; o(s) := k;$
(17) END;
(18) END.

Este evident că algoritmul PTG are complexitatea tot $O(m)$ şi că vectorul p determină una sau mai multe arborescenţe parcurgere.

Observaţia 2.2. Din modul cum se calculează $o(y) := k$, în linia (11) a algoritmului PG sau în liniile (13) sau (16) a algoritmului PTG, rezultă că y este al k-lea nod vizitat în parcurgerea digrafului $G = (N, A)$.

Observaţia 2.3. Algoritmul PG sau PTG poate fi aplicat şi grafurilor neorientate. În acest caz subgraful predecesor $G_p = (N_p, A_p)$ este tot o arborescenţă sau mai multe arborescenţe.

Observaţia 2.4. Drumul unic de la nodul sursă s la un nod y din arborescenţa parcurgere $G_p = (N_p, A_p)$ este furnizat de următoarea procedură:

```
(1)  PROCEDURA DRUM (G, s, y);
(2)  BEGIN;
(3)     se tipăreşte y;
(4)     WHILE p(y) ≠ 0 DO
(5)     BEGIN
(6)        x := p(y);
(7)        se tipăreşte x;
(8)        y := x
(9)     END;
(10) END.
```

După ordinea de selectare şi adăugare a nodurilor la mulţimea V se cunosc mai multe moduri de parcurgere a unui digraf $G = (N, A)$. Cele mai uzuale moduri de parcurgere sunt:

- parcurgerea se face "mai întâi în lăţime", în engleză breadth first (BF);
- parcurgerea se face "mai întâi în adâncime", în engleză depth first (DF).

2.2 Parcurgerea BF a grafurilor

Fie digraful $G = (N, A)$ cu nodul sursă s şi $\mathcal{D}_x$ mulţimea drumurilor de la nodul sursă s la nodul $x \in N$. Numărul de arce ce compun un drum $D_x \in \mathcal{D}_x$ defineşte *lungimea* acestui drum pe care o notăm $l(D_x)$. *Distanţa* de la nodul s la nodul x se defineşte în modul următor:

$$d(x) = \begin{cases} min\{l(D_x) \mid D_x \in \mathcal{D}_x\} & \text{dacă } \mathcal{D}_x \neq \emptyset \\ \infty & \text{dacă } \mathcal{D}_x = \emptyset \end{cases}$$

Un drum $\widehat{D}_x \in \mathcal{D}_x$ cu $l\left(\widehat{D}_x\right) = d(x)$ se numeşte *cel mai scurt drum* de la nodul sursă s la nodul x.

Observaţia 2.5. Pentru orice arc $(x, y) \in A$ avem $d(y) \leq d(x) + 1$.

Într-adevăr dacă $\mathcal{D}_x = \emptyset$ atunci $d(x) = \infty$ şi inegalitatea se păstrează. Dacă $\mathcal{D}_x \neq \emptyset$ atunci evident $\mathcal{D}_y \neq \emptyset$. Fie $\widehat{D}_x$ un cel mai scurt drum de la s la x, deci $l\left(\widehat{D}_x\right) = d(x)$. Mulţimea $\mathcal{D}_y$ poate conţine un drum D_y astfel încât $l(D_y) < d(x) + 1$. Conform definiţiei distanţei avem $d(y) \leq l(D_y)$ şi rezultă că $d(y) < d(x)+1$. Avem egalitate când un drum cel mai scurt $\widehat{D}_y$ de la s la y are lungimea $d(y) = l\left(\widehat{D}_y\right) = d(x) + 1$, de exemplu $\widehat{D}_y = \widehat{D}_x \cup \{(x, y)\}$.

În algoritmul parcurgerii BF (algoritmul PBF) se folosesc aceleaşi notaţii ca în algoritmul PG cu deosebirea că în locul tabloului ordine o se utilizează tabloul lungime l care este unidimensional şi are n elemente. Mulţimea nodurilor vizitate şi neanalizate V este organizată, ca structură de date, ca o coadă.

Algoritmul PBF este următorul:

```
(1)   PROGRAM PBF;
(2)   BEGIN;
(3)      U := N − {s} ; V := {s} ; W := ∅;
(4)      FOR toţi y ∈ N DO p(y) := 0;
(5)      l(s) := 0;
(6)      FOR toţi y ∈ U DO l(y) := ∞;
(7)      WHILE V ≠ ∅ DO
(8)      BEGIN
(9)         se selectează cel mai vechi nod x introdus în V;
(10)        FOR (x, y) ∈ A DO
(11)           IF y ∈ U
(12)              THEN U := U − {y} ; V := V ∪ {y} ; p(y) := x;
                            l(y) := l(x) + 1;
(13)        V := V − {x} ; W := W ∪ {x} ;
(14)     END;
(15)  END.
```

Teorema 2.4 *(1) Algoritmul PBF calculează elementele tabloului l astfel încât $d(y) \leq l(y)$, $y \in N$;*
(2) Dacă la iteraţia k oarecare a algoritmului PBF avem $V = \{x_1, \ldots, x_r\}$ în această ordine, atunci $l(x_r) \leq l(x_1) + 1$ şi $l(x_i) \leq l(x_{i+1})$, $i = 1, \ldots, r-1$.

Demonstraţie *(1)* Utilizăm inducţia după k numărul de iteraţii ale ciclului WHILE. Iniţial $l(s) := 0$, $l(y) := \infty$ pentru $y \in U$ şi evident $d(y) \leq l(y)$, $y \in N$. Presupunem că la iteraţia k avem $d(y) \leq l(y)$pentru $y \in N$. La iteraţia $k + 1$ pot exista cazurile:

(c1) Există arc (x, y) admisibil ($(x, y) \in A$ şi $y \in U$). În acest caz $l(y) = l(x) + 1$ şi $d(y) \leq d(x) + 1 \leq l(x) + 1 = l(y)$ ($l(x)$ pentru $x \in V$ nu se

modifică). Deci pentru toate arcele $(x,y) \in A$ şi $y \in U$ avem $d(y) \leq l(y)$. Pentru celelalte noduri y, conform ipotezei inducţiei, avem $d(y) \leq l(y)$, deoarece la iteraţia $k+1$ $l(y)$ nu se mai modifică.
(c2) Nu există arc (x,y) admisibil ($(x,y) \notin A$ sau $y \notin U$). În acest caz la iteraţia $k+1$ nu se modifică nici un element $l(y)$ şi conform ipotezei inducţiei $d(y) \leq l(y)$ pentru $y \in N$.
(2) Utilizăm inducţia după k numărul de iteraţii ale ciclului WHILE. Iniţial $V := \{s\}$. Deci $x_1 = s$, $x_r = s$ şi $l(s) < l(s) + 1$, $l(s) = l(s)$. Presupunem că la iteraţia k avem $l(x_r) \leq l(x_1) + 1$ şi $l(x_i) < l(x_{i+1})$, pentru $i = 1, \ldots, r-1$. La iteraţia $k+1$ pot exista cazurile:
(c1) Există arc (x,y) admisibil ($(x,y) \in A$ şi $y \in U$). În acest caz $V = \{x_1, \ldots, x_r, x_{r+1}\}, x_1 = x$, $x_{r+1} = y$. Astfel $l(x_{r+1}) = l(y) = l(x) + 1 = l(x_1) + 1$. De asemenea, avem $l(x_r) \leq l(x_1) + 1 = l(x) + 1 = l(y) = l(x_{r+1})$ şi inegalităţile $l(x_i) \leq l(x_{i+1})$, $i = 1, \ldots, r-1$ au rămas nemodificate.
(c2) Nu există arc (x,y) admisibil ($(x,y) \notin A$ sau $y \notin U$). În acest caz $V = \{x_2, \ldots, x_r\}$. Avem $l(x_r) \leq l(x_1) + 1 \leq l(x_2) + 1$ şi inegalităţile $l(x_i) \leq l(x_{i+1})$, $i = 1, \ldots, r-1$ au rămas nemodificate.■

Teorema 2.5. *Algoritmul PBF este convergent şi determină:*
(1) mulţimea tuturor nodurilor care sunt accesibile din nodul sursă s;
(2) elementele tabloului l astfel încât $l(y) = d(y)$ pentru $y \in N$.

Demonstraţie Convergenţa şi punctul *(1)* se demonstrează la fel ca Teorema 2.1. Punctul *(2)* se demonstrează prin inducţie după k numărul iteraţiilor ciclului WHILE. Fie mulţimea $N_k = \{y \in N \mid d(y) = k\}$. Pentru $k = 0$ avem $N_0 = \{s\}$ şi deci $d(s) = l(s) = 0$. Presupunem afirmaţia adevărată pentru k. Afirmaţia pentru $k+1$ rezultă cu uşurinţă, deoarece, în conformitate cu Teorema 2.4 punctul *(2)*, un nod $y \in N_{k+1}$ este vizitat plecând de la un nod $x \in N_k$ numai după ce toate nodurile din N_k sunt vizitate. Deci, dacă $y \in N_{k+1}$ şi este vizitat explorând arcul (x,y), $x \in N_k$, atunci, $l(y) = l(x) + 1 = d(x) + 1 = k + 1 = d(y)$.■

Teorema 2.6. *Algoritmul PBF are complexitatea $O(m)$.*

Demonstraţie Evident că algoritmul PBF are aceeaşi complexitate ca a algoritmului PG, adică $O(m)$. ■

În parcurgerea BF, dacă $N_p = W$ şi subgraful predecesor $G_p = (N_p, A_p)$ este o arborescenţă atunci G_p se numeşte *arborescenţă parcurgere BF.*

Teorema 2.7. *Algoritmul PBF determină elementele tabloului p astfel încât subgraful predecesor $G_p = (N_p, A_p)$ este o arborescenţă parcurgere BF.*

Demonstraţie Se demonstrează analog cu Teorema 2.3.■

Observaţia 2.6. Algoritmul parcurgerii totale BF (algoritmul PTBF) se obţine din algoritmul PBF analog cum s-a obţinut algoritmul PTG din algoritmul PG.

Observaţia 2.7. Drumul unic de la nodul sursă s la un nod y din arborescenţa parcurgere BF este un cel mai scurt drum de la nodul sursă s la acelaşi nod y din digraful G, conform punctului *(2)* al Teoremei 2.5.
Observaţia 2.8. Algoritmul PBF sau PTBF se poate aplica şi grafurilor neorientate. În acest caz subgraful predecesor $G_p = (N_p, A_p)$ este o arborescenţă sau mai multe arborescenţe.
Observaţia 2.9. Drumul unic de la nodul sursă s la un nod y din arborescenţa parcurgere BF se poate determina cu PROCEDURA DRUM prezentată în Observaţia 2.4.
Exemplu 2.1. Se aplică algoritmul PBF digrafului din figura 2.1.

Fig. 2.1.

Iniţializări: $s = 1$, $U = \{2, 3, 4, 5, 6\}$, $V = \{1\}$, $W = \emptyset$, $p = (0, 0, 0, 0, 0, 0)$, $l = (0, \infty, \infty, \infty, \infty, \infty)$.
Iteraţia 1: $x = 1$, $(1, 2) \in A$, $2 \in U : U = \{3, 4, 5, 6\}$, $V = \{1, 2\}$, $p = (0, 1, 0, 0, 0, 0)$, $l = (0, 1, \infty, \infty, \infty, \infty)$; $(1, 3) \in A$, $3 \in U : U = \{4, 5, 6\}$, $V = \{1, 2, 3\}$, $p = (0, 1, 1, 0, 0, 0)$, $l = (0, 1, 1, \infty, \infty, \infty)$; $V = \{2, 3\}$; $W = \{1\}$.
Iteraţia 2: $x = 2$, $(2, 4) \in A$, $4 \in U : U = \{5, 6\}$, $V = \{2, 3, 4\}$, $p = (0, 1, 1, 2, 0, 0)$, $l = (0, 1, 1, 2, \infty, \infty)$; $(2, 5) \in A$, $5 \in U : U = \{6\}$, $V = \{2, 3, 4, 5\}$, $p = (0, 1, 1, 2, 2, 0)$, $l = (0, 1, 1, 2, 2, \infty)$; $V = \{3, 4, 5\}$; $W = \{1, 2\}$.
Iteraţia 3: $x = 3$, $V = \{4, 5\}$, $W = \{1, 2, 3\}$.
Iteraţia 4: $x = 4$, $(4, 6) \in A$, $6 \in U : U = \emptyset$, $V = \{4, 5, 6\}$, $p = (0, 1, 1, 2, 2, 4)$, $l = (0, 1, 1, 2, 2, 3)$; $V = \{5, 6\}$; $W = \{1, 2, 3, 4\}$.
Iteraţia 5: $x = 5$, $V = \{6\}$, $W = \{1, 2, 3, 4, 5\}$.
Iteraţia 6: $x = 6$, $V = \emptyset$, $W = \{1, 2, 3, 4, 5, 6\}$.
$N_p = \{1, 2, 3, 4, 5, 6\} = W$.

Arborescenţa parcurgere BF este prezentată în figura 2.2.

Fig. 2.2.

Drumul unic de la nodul 1 la nodul 6 se obţine cu PROCEDURA DRUM în modul următor:
$y = 6$ este ultimul nod al drumului;
Iteraţia 1: $x = p(6) = 4$, $y = 4$.
Iteraţia 2: $x = p(4) = 2$, $y = 2$.
Iteraţia 3: $x = p(2) = 1$, $y = 1$.
Drumul este: $1, 2, 4, 6$.

2.3 Parcurgerea DF a grafurilor

În algoritmul parcurgerii DF (algoritmul PDF) se folosesc aceleaşi notaţii ca în algoritmul PG cu deosebirea că în locul tabloului unidimensional *ordine o* se utilizează tablourile *timp* unidimensionale t_1 şi t_2 care au fiecare n elemente. Mulţimea nodurilor vizitate şi neanalizate V este organizată, ca structură de date, ca stivă.

Algoritmul PDF este următorul:

```
(1)   PROGRAM PDF;
(2)   BEGIN;
(3)     U := N − {s} ; V := {s} ; W := ∅;
(4)     FOR toţi y ∈ N DO p (y) := 0;
(5)     t := 1; t1 (s) := 1; t2 (s) := ∞;
(6)     FOR toţi y ∈ U DO t1 (y) := ∞; t2 (y) := ∞;
(7)     WHILE V ≠ ∅ DO
(8)     BEGIN
(9)       se selectează cel mai nou nod x introdus în V;
(10)      IF există arc (x, y) ∈ A şi y ∈ U
(11)          THEN U := U − {y} ; V := V ∪ {y} ; p (y) := x;
                       t := t + 1; t1 (y) := t
(12)          ELSE V := V − {x} ; W := W ∪ {x} ; t := t + 1; t2 (x) := t;
(13)    END;
(14)  END.
```

Din liniile (10), (11), (12) ale algoritmului PDF rezultă că elementul $t_1(y)$ reprezintă momentul când y devine nod vizitat şi neanalizat şi elementul $t_2(x)$ reprezintă momentul când x devine nod vizitat şi analizat.

Deoarece algoritmul parcurgerii totale DF (algoritmul PTDF) are multiple aplicaţii, se va studia în continuare acest algoritm.

Algoritmul PTDF este următorul:

```
(1)   PROGRAM PTDF;
(2)   BEGIN;
(3)     U := N − {s} ; V := {s} ; W := ∅;
(4)     FOR toţi y ∈ N DO p (y) := 0;
(5)     t := 1; t1 (s) := 1; t2 (s) := ∞;
(6)     FOR toţi y ∈ U DO t1 (y) := ∞; t2 (y) := ∞;
(7)     WHILE W ≠ N DO
(8)     BEGIN
(9)       WHILE V ≠ ∅ DO
(10)      BEGIN
(11)        se selectează cel mai nou nod x introdus în V;
(12)        IF există arc (x, y) ∈ A şi y ∈ U
```

(13) THEN $U := U - \{y\}; V := V \cup \{y\}; p(y) := x;$
$t := t + 1; t_1(y) := t$
(14) ELSE $V := V - \{x\}; W := W \cup \{x\};$
$t := t + 1; t_2(x) := t;$
(15) END;
(16) se selectează $s \in U; U := U - \{s\}; V := \{s\};$
$t := t + 1; t_1(s) := t;$
(17) END;
(18) END.

Fie mulţimea $S = \{s \mid s \in N,\ s$ selectate în linia (3) şi linia (16)$\}$.

Teorema 2.8. *Algoritmul PTDF este convergent şi determină mulţimile nodurilor accesibile din s, $s \in S$.*

Demonstraţie. Se demonstrează la fel ca Teorema 2.1.■

Teorema 2.9 *Algoritmul PTDF are complexitatea $O(m)$.*

Demonstraţie. Evident că algoritmul PTDF are aceeaşi complexitate ca a algoritmului PTG, adică $O(m)$. ■

Un digraf $\widehat{G} = \left(\widehat{N}, \widehat{A}\right)$se numeşte *pădure* dacă este format din una sau mai multe arborescenţe. Un graf neorientat $\widehat{G} = \left(\widehat{N}, \widehat{A}\right)$se numeşte pădure dacă este format din unul sau mai mulţi arbori.

În parcurgerea totală DF, dacă subgraful predecesor $G_p = (N_p, A_p)$, $N_p = \{y \mid p(y) \neq 0\} \cup S$, $A_p = \{(p(y), y) \mid y \in N_p - S\}$ este o pădure şi $N_p = W$, atunci G_p se numeşte *pădure parcurgere* DF.

Teorema 2.10. *Algoritmul PTDF determină elementele tabloului p astfel încât subgraful predecesor $G_p = (N_p, A_p)$ este o pădure parcurgere DF.*

Demonstraţie. Se demonstrează analog ca Teorema 2.3.■

Teorema 2.11. *În orice parcurgere totală DF a unui digraf $G = (N, A)$ pentru oricare două noduri x şi y, una din următoarele trei condiţii este îndeplinită:*
(c1) intervalele $[t_1(x), t_2(x)]$ şi $[t_1(y), t_2(y)]$ sunt disjuncte;
(c2) intervalul $[t_1(x), t_2(x)]$ include strict intervalul $[t_1(y), t_2(y)]$ şi x este un ascendent al lui y în pădurea parcurgere DF;
(c3) intervalul $[t_1(x), t_2(x)]$ este inclus strict în intervalul $[t_1(y), t_2(y)]$ şi x este un descendent al lui y în pădurea parcurgere DF.

Demonstraţie. Există patru posibilităţi de considerat:
(p1) $t_1(x) < t_1(y)$ şi $t_2(x) < t_1(y)$. Rezultă $t_1(x) < t_2(x) < t_1(y) < t_2(y)$, adică (c1).
(p2) $t_1(x) < t_1(y)$ şi $t_2(x) > t_1(y)$. Rezultă că x este mai vechi decât y în V şi deci x va fi introdus după y în W. Evident că x este ascendent al lui y în pădurea parcurgere DF şi $t_2(x) > t_2(y)$, adică (c2).
(p3) $t_1(x) > t_1(y)$ şi $t_1(x) > t_2(y)$. Rezultă $t_1(y) < t_2(y) < t_1(x) < t_2(x)$, adică (c1).

(p4) $t_1(x) > t_1(y)$ şi $t_1(x) < t_2(y)$. Rezultă că x este mai nou decât y în V şi deci x va fi introdus înaintea lui y în W. Evident că x este descendent al lui y în pădurea parcurgere DF şi $t_2(x) < t_2(y)$, adică (c3).■

Parcurgerea totală DF poate fi utilizată la clasificarea arcelor digrafului $G = (N, A)$ în raport cu pădurea parcurgere DF. În această clasificare există două clase de arce:

- arce arborescenţă;
- arce nonarborescenţă.

Mulţimea *arcelor arborescenţă*, notată P, este $P = A_p$. Aceste arce introduc, în timpul parcurgerii totale DF a digrafului $G = (N, A)$, noduri în mulţimea nodurilor vizitate şi neanalizate V. Mulţimea *arcelor nonarborescenţă*, notată $\overline{P}$, este $\overline{P} = A - A_p$. Aceste arce introduc, în timpul parcurgerii totale DF a digrafului $G = (N, A)$, noduri în mulţimea nodurilor vizitate şi analizate W.

Clasa arcelor nonarborescenţă $\overline{P}$ este alcătuită din trei subclase:

- arce de înaintare;
- arce de revenire;
- arce de traversare.

Mulţimea arcelor de *înaintare*, notată I, este alcătuită din acele arce (x, y) pentru care nodul x este un ascendent al nodului y în pădurea parcurgere DF. Mulţimea *arcelor de revenire*, notată R, este alcătuită din acele arce (x, y) pentru care nodul x este un descendent al nodului y în pădurea parcurgere DF. Mulţimea *arcelor de traversare*, notată T, este alcătuită din acele arce (x, y) pentru care nodul x nu este nici ascendent nici descendent al nodului y în pădurea parcurgere DF. Aceste trei mulţimi de arce nonarborescenţă sunt disjuncte două câte două şi $\overline{P} = I \cup R \cup T$.

Teorema 2.12. *Într-o parcurgere totală DF a unui digraf $G = (N, A)$, arcul (x, y) este:*
(1) un arc arborescenţă sau de înaintare dacă şi numai dacă $t_1(x) < t_1(y) < t_2(y) < t_2(x)$;
(2) un arc de revenire dacă şi numai dacă $t_1(y) < t_1(x) < t_2(x) < t_2(y)$;
(3) un arc de traversare dacă şi numai dacă $t_1(y) < t_2(y) < t_1(x) < t_2(x)$.
Demonstraţie. Rezultă din Teorema 2.11 şi definiţiile clasificării arcelor.■
Observaţia 2.10. Algoritmul PDF sau PTDF se poate aplica şi grafurilor neorientate. În acest caz subgraful predecesor $G_p = (N_p, A_p)$ este tot o arborescenţă respectiv o pădure. Într-o parcurgere totală DF a unui graf neorientat $G = (N, A)$ avem $I = \emptyset$ şi $T = \emptyset$. Într-adevăr, fie $[x, y] \in A$ şi presupunem fără a restrânge generalitatea că $t_1(x) < t_1(y)$. În acest caz, nodul x este introdus înainte de nodul y în V şi după nodul y în W. Dacă muchia $[x, y]$ este explorată de la x la y, atunci $[x, y]$ devine un arc arbore (x, y). Dacă muchia $[x, y]$ este explorată de la y la x, atunci muchia $[x, y]$ devine un arc de revenire (y, x), deoarece x este

nod vizitat la acest timp.

Exemplul 2.2. Se aplică algoritmul PTDF digrafului reprezentat în figura 2.3.

Iniţializări: $s = 1, U = \{2,3,4,5,6,7,8\}, V = \{1\}, W = \emptyset,$

Fig. 2.3

$p = (0,0,0,0,0,0,0,0), t = 1, t_1 = (1,\infty,\infty,\infty,\infty,\infty,\infty,\infty),$
$t_2 = (\infty,\infty,\infty,\infty,\infty,\infty,\infty,\infty).$
Iteraţia 1: $x = 1, (1,2) \in A, 2 \in U : U = \{3,4,5,6,7,8\}, V = \{1,2\},$
$p = (0,1,0,0,0,0,0,0), t = 2, t_1 = (1,2,\infty,\infty,\infty,\infty,\infty,\infty).$
Iteraţia 2: $x = 2, (2,3) \in A, 3 \in U : U = \{4,5,6,7,8\}, V = \{1,2,3\},$
$p = (0,1,2,0,0,0,0,0), t = 3, t_1 = (1,2,3,\infty,\infty,\infty,\infty,\infty).$
Iteraţia 3: $x = 3, (3,4) \in A, 4 \in U : U = \{5,6,7,8\}, V = \{1,2,3,4\},$
$p = (0,1,2,3,0,0,0,0), t = 4, t_1 = (1,2,3,4,\infty,\infty,\infty,\infty).$
Iteraţia 4: $x = 4 : V = \{1,2,3\}, W = \{4\}, t = 5, t_2 = (\infty,\infty,\infty,5,\infty,\infty,\infty,\infty).$
Iteraţia 5: $x = 3 : V = \{1,2\}, W = \{4,3\}, t = 6, t_2 = (\infty,\infty,6,5,\infty,\infty,\infty,\infty).$
Iteraţia 6: $x = 2, (2,5) \in A, 5 \in U : U = \{6,7,8\}, V = \{1,2,5\}, p = (0,1,2,3,2,0,$
$0,0), t = 7, t_1 = (1,2,3,4,7,\infty,\infty,\infty).$
Iteraţia 7: $x = 5 : V = \{1,2\}, W = \{4,3,5\}, t = 8, t_2 = (\infty,\infty,6,5,8,\infty,\infty,\infty).$
Iteraţia 8: $x = 2 : V = \{1\}, W = \{4,3,5,2\}, t = 9, t_2 = (\infty,9,6,5,8,\infty,\infty,\infty).$
Iteraţia 9: $x = 1 : V = \emptyset, W = \{4,3,5,2,1\}, t = 10, t_2 = (10,9,6,5,8,\infty,\infty,\infty).$
Actualizări: $s = 6, U = \{7,8\}, V = \{6\}, t = 11, t_1 = (1,2,3,4,7,11,\infty,\infty)$
Iteraţia 10: $x = 6, (6,7) \in A, 7 \in U : U = \{8\}, V = \{6,7\}, p = (0,1,2,3,2,0,6,0),$
$t = 12, t_1 = (1,2,3,4,7,11,12,\infty).$
Iteraţia 11: $x = 7 : V = \{6\}, W = \{4,3,5,2,1,7\}, t = 13, t_2 = (10,9,6,5,8,\infty,13,$
$\infty).$
Iteraţia 12: $x = 6, (6,8) \in A, 8 \in U : U = \emptyset, V = \{6,8\}, p = (0,1,2,3,2,0,6,6),$
$t = 14, t_1 = (1,2,3,4,7,11,12,14).$
Iteraţia 13: $x = 8 : V = \{6\}, W = \{4,3,5,2,1,7,8\}, t = 15, t_2 = (10,9,6,5,8,\infty,$
$13,15).$
Iteraţia 14: $x = 6 : V = \emptyset, W = \{4,3,5,2,1,7,8,6\}, t = 16, t_2 = (10,9,6,5,8,16,$
$13,15).$

În figura 2.4 se prezintă pădurea parcurgere DF formată din două arborescenţe parcurgere DF şi se prezintă intervalele $[t_1(x), t_2(x)]$.

Fiecare dreptunghi în care este înscris un nod x acoperă intervalul $[t_1(x), t_2(x)]$. Tabloul predecesor este $p = (0,1,2,3,2,0,6,6)$.

Fig. 2.4

În figura 2.5 se prezintă redesenat graful din figura 2.3. Toate arcele arbore şi de înaintare sunt orientate de sus în jos. Pe fiecare arc se specifică clasa la care

aparţine.

Fig. 2.5

2.4 Aplicaţii şi comentarii bibliografice

2.4.1 Sortarea topologică

Teorema 2.13. *Un digraf $G = (N, A)$ este fără circuite dacă şi numai dacă orice parcurgere totală DF a lui G nu produce arce de revenire.*

Demonstraţie. Presupunem că digraful G este fără circuite. Prin reducere la absurd presupunem că o parcurgere totală DF a lui G produce arce de revenire ($R \neq \emptyset$). Fie arcul $(z,\ x) \in R$. În acest caz nodul z este un descendent al nodului x în pădurea parcurgere DF. Deci există un drum D de la x la z în G. Atunci $\overset{\circ}{D} = D \cup \{z, x\}$ este un circuit în G şi aceasta contrazice ipoteza că digraful G este fără circuite.

Reciproc, presupunem că o parcurgere totală DF a digrafului G nu produce arce de revenire ($R = \emptyset$). Prin reducere la absurd presupunem că G conţine un circuit $\overset{\circ}{D}$. Fie x primul nod vizitat din $\overset{\circ}{D}$ şi fie arcul $(z, x) \in \overset{\circ}{D}$. Deoarece nodurile $x,\ z \in \overset{\circ}{D}$ rezultă faptul că există un drum D de la x la z. De asemenea x fiind primul nod vizitat din $\overset{\circ}{D}$ rezultă că nodul z devine un descendent al nodului x la pădurea PDF. Deci (z, x) este un arc de revenire ce contrazice ipoteza $R = \emptyset$.■

Sortarea topologică a unui digraf $G = (N, A)$ fără circuite constă într-o ordonare liniară a nodurilor din N astfel încât dacă $(x, y) \in A$ atunci x apare înaintea lui y în ordonare.

Algoritmul sortare topologică (algoritmul ST) se obţine din algoritmul PTDF făcând următoarele două completări:
(1) în partea de iniţializări (liniile (3)-(6)) se iniţializează o listă a nodurilor;
(2) în linia (16) după calculul lui $t_2(x)$, nodul x se introduce la începutul listei.

La terminarea algoritmului ST, lista furnizează sortarea topologică a digrafului $G = (N, A)$ fără circuite şi nodurile x sunt plasate în listă în ordinea descrescătoare a timpilor $t_2(x)$.

2.4.2 Componentele conexe ale unui graf

Definiţia 2.1 Un digraf $G = (N, A)$ se numeşte *slab conex* sau *conex* dacă pentru oricare două noduri diferite x, y există un lanţ care are aceste două noduri drept extremităţi.

Noţiunea de conexitate are sens şi pentru grafuri neorientate.

Definiţia 2.2. Se numeşte *componentă conexă* a unui digraf $G = (N, A)$ un subgraf $G' = (N', A')$ al lui G, care este conex şi care este maximal în raport cu

incluziunea faţă de această proprietate (oricare ar fi $x \in \overline{N}' = N - N'$, subgraful G'_x generat de $N'_x = N' \cup \{x\}$ nu este conex).

O componentă conexă $G' = (N', A')$ a unui digraf $G = (N, A)$ se poate identifica cu mulţimea N' care generează subgraful G'.

Deoarece în problema conexităţii sensul arcelor nu contează se va considera că grafurile sunt neorientate. Dacă $G = (N, A)$ este digraf atunci i se asociază graful neorientat $\widehat{G} = (\widehat{N}, \widehat{A})$, unde $\widehat{N} = N, \widehat{A} = \{[x, y] \mid (x, y) \in A$ şi /sau $(y, x) \in A\}$. Este evident că G şi $\widehat{G}$ au aceleaşi componente conexe.

Algoritmul componentelor conexe (algoritmul CC) este o adaptare a algoritmului PTDF aplicat unui graf neorientat $G = (N, A)$. Nu se calculează tablourile timp t_1, t_2 şi prin p notăm o variabilă scalară a cărei valoare reprezintă numărul componentelor conexe. Algoritmul CC este următorul:

```
(1)   PROGRAM CC;
(2)   BEGIN
(3)     U := N − {s}; V := {s}; W := ∅; p := 1; N' = {s};
(4)     WHILE W ≠ N DO
(5)     BEGIN
(6)       WHILE V ≠ ∅ DO
(7)       BEGIN
(8)         se selectează cel mai nou nod x introdus în V;
(9)         IF există muchie [x, y] ∈ A şi y ∈ U
(10)              THEN U := U − {y}; V := V ∪ {y}; N' := N' ∪ {y}
(11)              ELSE V := V − {x}; W := W ∪ {x};
(12)      END;
(13)      se tipăresc p şi N';
(14)      se selectează s ∈ U; U := U − {s}; V := {s}; p := p + 1;
                          N' := {s};
(15)    END;
(16)  END.
```

La terminarea algoritmului pot exista cazurile:
(c1) se tipăreşte o singură componentă conexă şi în acest caz graful $G = (N, A)$ este conex;
(c2) se tipăresc mai multe componente conexe şi în acest caz graful $G = (N, A)$ nu este conex, obţinându-se toate componentele conexe ale lui G.

Teorema 2.14. *Algoritmul CC determină componentele conexe ale unui graf neorientat* $G = (N, A)$.

Demonstraţie. La terminarea execuţiei ciclului WHILE se determină mulţimea N' a tuturor nodurilor accesibile printr-un lanţ cu aceeaşi extremitate, nodul s.

Mulţimea N' generează evident o componentă conexă $G' = (N', A')$. Deoarece la terminarea execuţiei algoritmului avem $W = N$ rezultă că algoritmul CC determină toate componentele conexe ale lui $G = (N, A)$. ■

Teorema 2.15. *Algoritmul CC are complexitatea* $O(m)$.

Demonstraţie. Algoritmul CC are aceeaşi complexitate cu a algoritmului PTDF, adică $O(m)$.■

Exemplul 2.3. Fie digraful din figura 2.6. Pentru a determina componentele conexe ale acestui digraf se transformă într-un graf neorientat reprezentat în figura 2.7 căruia i se aplică algoritmul CC.

Fig. 2.6

Fig. 2.7

Iniţializări: $s = 1, U = \{2, 3, 4, 5, 6, 7, 8\}, V = \{1\}, W = \emptyset, p = 1, N' = \{1\}$.
Iteraţia 1: $x = 1, [1, 2] \in A, 2 \in U : U = \{3, 4, 5, 6, 7, 8\}, V = \{1, 2\}, N' = \{1, 2\}$.
Iteraţia 2: $x = 2, [2, 3] \in A, 3 \in U : U = \{4, 5, 6, 7, 8\}, V = \{1, 2, 3\}, N' = \{1, 2, 3\}$.
Iteraţia 3: $x = 3, [3, 4] \in A, 4 \in U : U = \{5, 6, 7, 8\}, V = \{1, 2, 3, 4\}, N' = \{1, 2, 3, 4\}$.
Iteraţia 4: $x = 4 : V = \{1, 2, 3\}, W = \{4\}$.
Iteraţia 5: $x = 3 : V = \{1, 2\}, W = \{4, 3\}$.
Iteraţia 6: $x = 2, V = \{1\}, W = \{4, 3, 2\}$.
Iteraţia 7: $x = 1 : V = \emptyset, W = \{4, 3, 2, 1\}$.

Se tipăresc $p = 1$ şi $N' = \{1, 2, 3, 4\}$
Actualizări: $s = 5, U = \{6, 7, 8\}, V = \{5\}, p = 2, N' = \{5\}$.
Iteraţia 8: $x = 5, [5, 6] \in A, 6 \in U : U = \{7, 8\}, V = \{5, 6\}, N' = \{5, 6\}$.
Iteraţia 9: $x = 6, [6, 8] \in A, 8 \in U : U = \{7\}, V = \{5, 6, 8\}, N' = \{5, 6, 8\}$.
Iteraţia 10: $x = 6, [8, 7] \in A, 7 \in U : U = \emptyset, V = \{5, 6, 8, 7\}, N' = \{5, 6, 8, 7\}$.
Iteraţia 11: $x = 7 : V = \{5, 6, 8\}, W = \{4, 3, 2, 1, 7\}$.

După încă trei iteraţii se obţine $V = \emptyset, W = \{4, 3, 2, 1, 7, 8, 6, 5\}$ se tipăresc $p = 2, N' = \{5, 6, 8, 7\}$ şi execuţia algoritmului se opreşte.

2.4.3 Componentele tare conexe ale unui graf

Definiţia 2.3 Un digraf $G = (N, A)$ se numeşte *tare conex* dacă pentru oricare două noduri x, y există un drum de la x la y şi un drum de la y la x.

Noţiunea de tare conexitate are sens numai pentru grafuri orientate.

Definiţia 2.4. Se numeşte *componentă tare conexă* a unui digraf $G = (N, A)$ un subgraf $G' = (N', A')$ al lui G care este tare conex şi care este maximal în raport cu incluziunea faţă de acestă proprietate (oricare ar fi $x \in \overline{N}' = N - N'$, subgraful G'_x generat de $N'_x = N' \cup \{x\}$ nu este tare conex).

O componentă tare conexă $G' = (N', A')$ a unui digraf $G = (N, A)$ se poate identifica cu mulţimea N' care generează subgraful G'.

Definiţia 2.5. Se numeşte *digraf condensat* asociat digrafului $G = (N, A)$ digraful $\widehat{G} = \left(\widehat{N}, \widehat{A}\right), \widehat{N} = \{N'_1, \ldots, N'_p\}, \widehat{A} = \{\left(N'_i, N'_j\right) \mid N'_i, N'_j \in \widehat{N}$, există $(x, y) \in A$ cu $x \in N'_i, \ , y \in N'_j\}$, unde $N'_i, \ldots, N'_p$ sunt componentele tare conexe ale digrafului $G = (N, A)$.

Algoritmul componentelor tare conexe (algoritmul CTC) este următorul:

```
(1)  PROGRAM CTC;
(2)  BEGIN
(3)     PROCEDURA PTDF(G);
(4)     PROCEDURA INVERSARE(G);
(5)     PROCEDURA PTDF(G^-1);
(6)     PROCEDURA SEPARARE(G^-1);
(7)  END.
```

Procedurile din program rezolvă următoarele probleme:
PROCEDURA PTDF(G) aplică algoritmul PTDF digrafului G;
PROCEDURA INVERSARE(G) determină inversul G^{-1} al digrafului G;
PROCEDURA PTDF(G^{-1}) aplică algoritmul PTDF digrafului G^{-1}, unde nodul s este selectat întotdeauna astfel încât $t_2(s) > t_2(x), x \in U^{-1} - \{s\}$ cu t_2 calculat în PROCEDURA PTDF(G);
PROCEDURA SEPARARE(G^{-1}) extrage nodurile fiecărei arborescenţe parcurgere DF din pădurea parcurgere DF obţinută în PROCEDURA PTDF (G^{-1}) pentru separarea componentelor tare conexe.

Se spune că nodul $r(x) \in N$ este *ascendent rădăcină* al nodului x în parcurgerea totală DF a digrafului $G = (N, A)$ dacă $r(x)$ este accesibil din x şi $t_2(r(x)) = \max\{t_2(z) \mid z \text{ este accesibil din } x\}$.

Teorema 2.16. *În oricare parcurgere totală DF a unui digraf $G = (N, A)$:*
(1) toate nodurile din aceeaşi componentă tare conexă aparţin la aceeaşi arborescenţă a pădurii parcurgere DF;
(2) pentru orice nod $x \in N$, ascendentul rădăcină $r(x)$ este un ascendent al lui x în pădurea parcurgere DF şi nodurile $x, r(x)$ se găsesc în aceeaşi componentă tare conexă.

Demonstraţie. Se recomandă cititorului comentariile bibliografice din acest capitol.■

Teorema 2.17. *Algoritmul CTC determină componentele tare conexe ale unui digraf $G = (N, A)$.*

Demonstraţie. Se bazează pe Teorema 2.16.■

Teorema 2.18. *Algoritmul CTC are complexitatea $O(m)$.*

Demonstraţie. Algoritmul CTC are aceeaşi complexitate cu a algoritmului PTDF, adică $O(m)$.■

Exemplul 2.4. Fie digraful din figura 2.8. Pentru a determina componentele tare conexe ale acestui digraf se aplică algoritmul CTC.

Fig. 2.8

PROCEDURA PTDF determină $p = (5, 0, 0, 3, 2, 7, 3, 7), t_1 = (13, 11, 1, 8, 12, 3, 2, 5), t_2 = (14, 16, 10, 9, 15, 4, 7, 6)$. Iniţial $s = 3$ şi apoi $s = 2$.

PROCEDURA INVERSARE determină inversul G^{-1} al digrafului G şi este reprezentat în figura 2.9.

Fig. 2.9

PROCEDURA PTDF aplicată digrafului G^{-1} determină: $p^{-1} = (2, 0, 0, 3, 1, 7, 0, 0), t_1^{-1} = (2, 1, 7, 8, 3, 12, 11, 15), t_2^{-1} = (5, 6, 10, 9, 4, 13, 14, 16)$ şi nodurile s au fost selectate astfel înât $t_2(s) > t_2(t),\ , x \in U^{-1} - \{s\}$.

PROCEDURA SEPARARE determină componentele tare conexe reprezentate ca noduri ale digrafului condensat $\widehat{G} = (\widehat{N}, \widehat{A})$, din figura 2.10.

Fig. 2.10

2.4.4 Comentarii bibliografice

Algoritmii de parcurgere a grafurilor sunt subrutine importante pentru algoritmii de optimizare în reţele. Tremaux (vezi Lucas (1882)) a fost printre primii autori care au utilizat ideea parcurgerii DF a grafurilor. Această parcurgere a fost folosită la studiul labirintului de către Tarry (vezi Tarry (1895)).

Principalele cărţi care tratează mai amplu problema parcurgerii grafurilor sunt următoarele:

1. *The Design and Analysis of Computer Algorithms*(**Aho, Hopcroft and Ullman** (1974)).
2. *Graph Algorithms* (**Even** (1979)).
3. *Data Structures and Network Algorithms*(**Tarjan** (1983)).
4. *Introduction to Algorithms* (**Cormen, Leiserson and Rivest** (1990)).
5. *Graphs, Networks and Algorithms* (**Jungnickel** (1999)).

Capitolul 3

Arbori şi arborescenţe

3.1 Cicluri şi arbori

În acest paragraf se va lucra cu cicluri şi cicluri elementare.

Fie digraful $G = (N, A)$ cu $N = \{1, \ldots, n\}$ şi $A = \{a_1, \ldots, a_m\}$. Pentru un ciclu $\overset{\circ}{L}$ vom nota cu $\overset{\circ}{L}^+$ mulţimea arcelor directe şi prin $\overset{\circ}{L}^-$ mulţimea arcelor inverse ale ciclului. Unui ciclu $\overset{\circ}{L}$ i se poate asocia un vector $\overset{\circ}{l} = \left(\overset{\circ}{e}_1, \ldots, \overset{\circ}{e}_m\right)$, unde

$$\overset{\circ}{e}_i = \begin{cases} -1 & \text{dacă } a_i \in \overset{\circ}{L}^-, \\ 1 & \text{dacă } a_i \in \overset{\circ}{L}^+, \\ 0 & \text{dacă } a_i \notin \overset{\circ}{L} \end{cases}$$

În acest paragraf un ciclu $\overset{\circ}{L}$ va fi identificat uneori cu vectorul $\overset{\circ}{l}$ pe care îl defineşte, iar orice sumă de cicluri $\overset{\circ}{L}_1 + \ldots + \overset{\circ}{L}_s$ va fi suma lor vectorială $\overset{\circ}{l}_1 + \ldots + \overset{\circ}{l}_s$.

Teorema 3.1. *Orice ciclu $\overset{\circ}{L}$ al digrafului $G = (N, A)$ este o sumă de cicluri elementare fără arce comune.*

Demonstraţie. Când se parcurge ciclul $\overset{\circ}{L}$ se obţin cicluri elementare $\overset{\circ}{L}_i$ ($i = 1, \ldots, s'$), de fiecare dată când se ajunge într-un nod deja întâlnit. Ciclurile elementare nu au arce comune deoarece ciclul $\overset{\circ}{L}$ nu are arce care se repetă. ■

Definiţia 3.1. Se spune că ciclurile $\overset{\circ}{l}_1, \ldots, \overset{\circ}{l}_s$ sunt *independente* dacă egalitatea $r_1 \overset{\circ}{l}_1 + \ldots + r_s \overset{\circ}{l}_s = 0$ cu $r_1, \ldots, r_s$ numere reale implică $r_1 = r_2 = \ldots = r_s = 0$, altfel se spune că ciclurile $\overset{\circ}{l}_1, \ldots, \overset{\circ}{l}_s$ sunt *dependente*.

Definiţia 3.2. Se spune că ciclurile $\overset{\circ}{l}_1, \ldots, \overset{\circ}{l}_s$ formează *o bază* de cicluri dacă sunt cicluri elementare independente şi oricare alt ciclu $\overset{\circ}{l}$ se poate exprima sub forma $\overset{\circ}{l} = r_1 \overset{\circ}{l}_1 + \ldots + r_s \overset{\circ}{l}_s$ cu $r_1, \ldots, r_s$ numere reale.

Numărul s al ciclurilor care formează o bază este *dimensiunea* subspaţiului liniar $S_{\overset{\circ}{l}}$ al lui $\Re^m$ generat de ciclurile digrafului $G = (N, A)$.

Exemplul 3.1. Fie digraful din figura 3.1. Ciclul $\overset{\circ}{L} = (a_1, a_4, a_9, a_8, a_7, a_6)$ este suma ciclurilor elementare fără arce comune $\overset{\circ}{L}_1 = (a_1, a_4, a_6)$ şi $\overset{\circ}{L}_2 = (a_9, a_8, a_7)$ sau $\overset{\circ}{l} = \overset{\circ}{l}_1 + \overset{\circ}{l}_2$ cu $\overset{\circ}{l} = (1, 0, 0, 1, 0, 1, 1, -1, -1)$, $\overset{\circ}{l}_1 = (1, 0, 0, 1, 0, 1, 0, 0, 0), \overset{\circ}{l}_2 = (0, 0, 0, 0, 0, 0, 1, -1, -1)$. Evident că ciclurile $\overset{\circ}{l}, \overset{\circ}{l}_1, \overset{\circ}{l}_2$ sunt dependente.

Fig. 3.1.

Definiţia 3.3. Se numeşte *număr ciclomatic* al digrafului $G = (N, A)$ cu n noduri, m arce şi p componente conexe numărul $\nu(G) = m - n + p$.

Teorema 3.2. *Subspaţiul $S_{\overset{\circ}{l}}$ al lui $\Re^m$, generat de ciclurile digrafului $G = (N, A)$, admite o bază formată din $\nu(G)$ cicluri elementare independente.*

Demonstraţie. Vom arăta că digraful G admite $\nu(G) = m - n + p$ cicluri elementare independente. Pentru aceasta vom construi un şir de grafuri parţiale $G_i = (N, A_i)$ ale digrafului $G = (N, A)$ cu $\nu(G_i) = m_i - n + p_i$ cicluri elementare independente. Acest şir se construieşte cu algoritmul următor.

```
(1)   PROGRAM CICLURI;
(2)   BEGIN
(3)      A_0 := ∅;
(4)      FOR i := 1 TO m DO A_i := A_{i-1} ∪ {a_i};
(5)   END.
```

Prin inducţie după i vom arăta că $G_i = (N, A_i)$ admite $\nu(G_i) = m_i - n + p_i$ cicluri elementare independente. Pentru $i = 0$ avem $A_0 = \emptyset$ şi obţinem $m_0 = 0$, deci $\nu(G_0) = 0$. Deoarece $p_0 = n$ rezultă că $\nu(G_0) = m_0 - n + p_0$.

Presupunem afirmaţia adevărată pentru i şi o demonstrăm pentru $i + 1$.

La iteraţia $i + 1$ poate exista unul din cazurile:

(c1) arcul a_{i+1} nu închide un nou ciclu elementar. Deci $\nu(G_{i+1}) = \nu(G_i)$ şi $m_{i+1} = m_i + 1, p_{i+1} = p_i - 1$. Rezultă $\nu(G_{i+1}) = m_i - n + p_i = m_{i+1} - n + p_{i+1}$.

(c2) arcul a_{i+1} închide un nou ciclu elementar $\overset{\circ}{L}_k$. Deoarece arcul a_{i+1} aparţine ciclului elementar $\overset{\circ}{L}_k$ şi nu aparţine ciclurilor elementare independente $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_{k-1}$ obţinute până la această iteraţie, rezultă că ciclurile elementare $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_{k-1}, \overset{\circ}{L}_k$ sunt independente. Rezultă că $\nu(G_{i+1}) = \nu(G_i) + 1,\ m_{i+1} = m_i + 1,\ p_{i+1} = p_i$. Deci $\nu(G_{i+1}) = m_i - n + p_i + 1 = m_{i+1} - n + p_{i+1}$.

Deci afirmaţia este adevărată pentru orice i. La terminarea execuţiei algoritmului avem $G_m = G, m_m = m, p_m = p$ şi $\nu(G) = \nu(G_m) = m_m - n + p_m = m - n + p$.

În continuare vom arăta că G nu admite mai mult de $\nu(G) = m-n+p$ cicluri elementare independente. Fie $\overset{\circ}{L}$ un ciclu elementar diferit de ciclurile $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_\nu$ construite cu algoritmul de mai sus. Din modul cum au fost determinate ciclurile elementare $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_\nu$ rezultă că oricare ar fi un arc a_j al ciclului $\overset{\circ}{L}$ există cel puţin un $\overset{\circ}{L}_i$, $i \in \{1, \ldots, \nu\}$, astfel încât a_j aparţine ciclului $\overset{\circ}{L}_i$. Deci există o relaţie $r_1 \overset{\circ}{l}_1 + \ldots + r_\nu \overset{\circ}{l}_\nu + r \overset{\circ}{l} = 0$ fără ca $r_1 = \ldots = r_\nu = r = 0$. Această relaţie implică faptul că ciclurile $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_\nu, L$ sunt dependente. Am demonstrat că $dim\left(S_{\overset{\circ}{l}}\right) = \nu(G) = m - n + p$. ■

Definiţia 3.4. Se spune că un digraf $G' = (N, A')$ este un *arbore* dacă este un digraf fără cicluri şi conex. Se spune că digraful $G' = (N, A')$ este o *pădure* dacă fiecare componentă conexă a lui G' este un arbore.

Din definiţie rezultă că o pădure este un digraf $G' = (N, A)$ fără cicluri. Noţiunile de arbore şi pădure au sens şi pentru grafuri neorientate.

Exemplul 3.2. Digraful reprezentat în figura 3.2. este o pădure compusă din doi arbori. Orientarea arcelor poate fi ignorată.

Fig. 3.2.

Teorema 3.3. *Fie $G' = (N,\ A')$ un digraf cu $n \geq 2$. Proprietăţile următoare sunt echivalente şi caracterizează un arbore:*
(1). G' este fără cicluri şi conex;
(2). G' este fără cicluri şi are $n-1$ arce;
(3). G' este conex şi are $n-1$ arce;
(4). G' este fără cicluri şi dacă se adaugă un singur arc între oricare două noduri atunci graful obţinut $\widetilde{G}' = \left(N, \widetilde{A}'\right)$ conţine un ciclu unic;
(5). G' este conex şi dacă se elimină un arc oarecare, atunci digraful obţinut $\widehat{G}' = \left(N, \widehat{A}'\right)$ nu este conex;
(6). G' are un lanţ unic între oricare două noduri ale lui.

Demonstraţie. (1) $\Rightarrow$ (2). Din (1) rezultă că $\nu(G') = 0$ şi $p' = 1$. Deci $m' - n + 1 = 0$, de unde $m' = n - 1$. Deducem că G' este fără cicluri şi are $n-1$ arce.
(2) $\Rightarrow$ (3). Din (2) rezultă că $\nu(G') = 0$ şi $m' - n + p' = n - 1 - n + p' = 0$, de unde $p' = 1$. Deducem că G' este conex şi are $n-1$ arce.
(3) $\Rightarrow$ (4). Din (3) rezultă $p' = 1$ şi $m' = n - 1$. Deci $\nu(G') = m' - n + p' = n - 1 - n + 1 = 0$ şi G' este fără cicluri. Digraful $\widetilde{G}' = (N, \widetilde{A}')$ are $\widetilde{p}' = p' = 1$, $\widetilde{m}' = m' + 1 = n$ şi $\nu\left(\widetilde{G}'\right) = \widetilde{m}' - n + \widetilde{p}' = 1$. Rezultă că $\widetilde{G}'$ are un singur

ciclu.
(4) $\Rightarrow$ (5). Din (4) rezultă $\nu(G') = 0$ şi $p' = 1$. Într-adevăr, dacă $p' > 1$ atunci arcul adăugat la A' poate lega două componente conexe şi $\widetilde{G}'$ nu conţine ciclu. Din $\nu(G') = 0$ şi $p' = 1$ rezultă $m' = n - 1$. Digraful $\widehat{G}' = (N, \widehat{A}')$ are $\widehat{m}' = m' - 1 = n - 2$, $\nu(\widehat{G}') = 0, \widehat{p}' = 2$ şi deci $\widehat{G}'$ nu este conex.
(5) $\Rightarrow$ (6). Dacă G' este conex atunci între oricare două noduri există un lanţ. Deoarece $\widehat{G}'$ este neconex rezultă că lanţul este unic.
(6) $\Rightarrow$ (1). Dacă între oricare două noduri ale lui G' există un lanţ unic atunci este evident că G' este conex şi fără cicluri.■

Teorema 3.4. *Un digraf $G = (N, A)$ admite un subgraf parţial $G' = (N, A')$ care este un arbore dacă şi numai dacă G este conex.*

Demonstraţie. Obţinem subgraful parţial $G' = (N, A')$ cu următorul algoritm:

```
(1)  PROGRAM ARBORE1;
(2)  BEGIN
(3)     A_m := A;
(4)     FOR i := m DOWNTO n DO
(5)     BEGIN
(6)        se selectează un arc a din A_i care nu deconexează G_i;
(7)        A_{i-1} := A_i - {a};
(8)     END;
(9)  END.
```

Se obţine $G' = (N, A') = (N, A_{n-1})$. Din modul cum este construit G' rezultă că el este conex şi are $n - 1$ arce. Deci G' este arbore.

Reciproca este evidentă. ■

Deoarece arborele G' construit în Teorema 3.4 are aceeaşi mulţime de noduri ca a digrafului G se numeşte *arbore parţial* al lui G.

Observaţia 3.1. Un arbore parţial $G' = (N, A')$ al digrafului $G = (N, A)$ se poate construi şi cu algoritmul următor:

```
(1)  PROGRAM ARBORE2;
(2)  BEGIN
(3)     A_0 := ∅;
(4)     FOR i := 0 TO n - 2 DO
(5)     BEGIN
(6)        se selectează un arc a din Ā_i = A - A_i care nu formează
           ciclu în G_i;
(7)        A_{i+1} := A_i ∪ {a};
(8)     END;
(9)  END.
```

Se obţine $G' = (N, A') = (N, A_{n-1})$. Deoarece G' este fără ciclu şi are $n-1$ arce el este un arbore.

Teorema 3.5. *Dacă $G = (N, A)$ este un digraf conex, $G' = (N, A')$ este un arbore parţial al lui G şi $\overline{A}' = A - A'$, atunci oricare arc $a_i \in \overline{A}'$ determină cu arcele din A' un ciclu unic $\overset{\circ}{L}_k$ şi ciclurile $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_s$ obţinute în acest mod cu toate arcele din $\overline{A}'$ formează o bază de cicluri a digrafului G.*

Demonstraţie. Dacă arcul $a_i \in \overline{A}'$, atunci digraful $\widetilde{G}' = (N, \widetilde{A}')$ cu $\widetilde{A}' = A' \cup \{a_i\}$ conţine un ciclu $\overset{\circ}{L}_k$ conform proprietăţii (4) din Teorema 3.3. Ciclurile $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_s$ obţinute în acest mod sunt independente, deoarece fiecare ciclu $\overset{\circ}{L}_k$ conţine un arc a_i care nu este conţinut de celelalte cicluri. Numărul acestor cicluri este $s = |\overline{A}'| = |A| - |A'| = m - (n-1) = m - n + 1 = m - n + p = \nu(G)$. Deci ciclurile $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_s$ formează o bază de cicluri.■

Teorema 3.5 ne furnizează un algoritm pentru a construi o bază de cicluri a unui digraf conex G.

Exemplul 3.3. Fie digraful $G = (N, A)$ reprezentat în figura 3.1. Un arbore parţial $G' = (N, A')$, $A' = \{a_1, a_2, a_3, a_5\}$ al digrafului G este reprezentat în figura 3.3. Avem $\overline{A}' = \{a_4, a_6, a_7, a_8, a_9\}$.

Fig. 3.3.

Se obţin ciclurile $\overset{\circ}{L}_1 = (a_1, a_4, a_2)$, $\overset{\circ}{L}_2 = (a_1, a_5, a_9, a_2)$, $\overset{\circ}{L}_3 = (a_1, a_5, a_8, a_3)$, $\overset{\circ}{L}_4 = (a_2, a_6)$, $\overset{\circ}{L}_5 = (a_2, a_7, a_3)$.

3.2 Arbori parţiali minimi

3.2.1 Condiţii de optimalitate

Se consideră un graf neorientat şi conex $G = (N, A)$ cu $N = \{1, \ldots, x, \ldots, n\}$, $A = \{1, \ldots, a, \ldots, m\}$ şi o funcţie $b : A \to \Re$ care asociază fiecărei muchii $a \in A$ un număr real $b(a)$ numit *valoarea* acestei muchii, care poate reprezenta lungime, cost etc.

Deoarece graful $G = (N, A)$ este conex, el admite un arbore parţial $G' = (N, A')$. Valoarea unui arbore parţial $G' = (N, A')$, notată $b(G')$, este prin definiţie

$$b(G') = \sum_{A'} b(a).$$

Dacă notăm cu $\mathcal{A}_G$ mulţimea arborilor parţiali ai grafului G, atunci problema arborelui parţial minim este următoarea: să se determine $G' \in \mathcal{A}_G$ astfel încât $b(G') = min\{b(G'') \mid G'' \in \mathcal{A}_G\}$.

Dacă eliminăm o muchie arbore oarecare $a = [x, \overline{x}]$ din A' atunci se obţine o partiţie a mulţimii nodurilor $N = X \cup \overline{X},\ x \in X,\ \overline{x} \in \overline{X}$.

Mulţimea de muchii

$$[X, \overline{X}]_a = \{[y, \overline{y}] \mid [y, \overline{y}] \in A,\ y \in X,\ \overline{y} \in \overline{X}\}$$

se numeşte *tăietură* generată de eliminarea muchiei arbore $a = [x, \overline{x}]$ sau *cociclu* generat de eliminarea muchiei arbore $a = [x, \overline{x}]$.

Pentru problema arborelui parţial minim se pot formula condiţii de optimalitate în două moduri: condiţii de optimalitate ale tăieturii şi condiţii de optimalitate ale lanţului.

Teorema 3.6. (Condiţiile de optimalitate ale tăieturii)
Fie $G = (N, A)$ un graf neorientat şi conex. Un arbore parţial $G' = (N, A')$ al grafului G este minim dacă şi numai dacă, pentru fiecare muchie arbore $a = [x, \overline{x}] \in A'$, avem:

$$b[x, \overline{x}] \leq b[y, \overline{y}] \text{ pentru toate muchiile } [y, \overline{y}] \in [X, \overline{X}]_a \tag{3.1}$$

Demonstraţie. Presupunem că arborele parţial $G' = (N, A')$ al grafului $G = (N, A)$ este minim. Prin reducere la absurd presupunem că nu sunt satisfăcute condiţiile (3.1). Atunci există o muchie arbore $a = [x, \overline{x}] \in A'$ şi o muchie nonarbore $[y, \overline{y}] \in [X, \overline{X}]_a$ cu $b[x, \overline{x}] > b[y, \overline{y}]$. Arborele parţial $G'' = (N, A'')$ cu $A'' = A' - \{[x, \overline{x}]\} \cup \{[y, \overline{y}]\}$ are valoarea $b(G'') = b(G') - b[x, \overline{x}] + b[y, \overline{y}]$. Deoarece $b[x, \overline{x}] > b[y, \overline{y}]$ rezultă $b(G'') < b(G')$. Această relaţie contrazice faptul că arborele parţial G' este minim. Deci condiţiile (3.1) sunt satisfăcute.

Presupunem că sunt satisfăcute condiţiile (3.1). Prin reducere la absurd presupunem că arborele parţial $G' = (N, A')$ nu este minim. Atunci există un arbore parţial $G'_0 = (N, A'_0)$ cu $b(G'_0) < b(G')$. Deoarece $A'_0 \neq A'$, există o muchie $a = [x, \overline{x}] \in A'$ şi $a = [x, \overline{x}] \notin A'_0$. Muchia $a = [x, \overline{x}]$ generează tăietura $[X, \overline{X}]_a$ şi un ciclu $\overset{\circ}{L}_a$ cu muchiile din A'_0 (vezi figura 3.4). Atunci există o muchie $[y, \overline{y}] \in \overset{\circ}{L}_a$ astfel încât $[y, \overline{y}] \in [X, \overline{X}]_a,\ [y, \overline{y}] \in A'_0$ şi $[y, \overline{y}] \notin A'$. Deoarece condiţiile (3.1) sunt satisfăcute avem $b[x, \overline{x}] \leq b[y, \overline{y}]$. Arborele parţial $G'' = (N, A'')$ cu $A'' = A' - \{[x, \overline{x}]\} \cup \{[y, \overline{y}]\}$ are valoarea $b(G'') = b(G') - b[x, \overline{x}] + b[y, \overline{y}]$. Rezultă $b(G') \leq b(G'')$ şi G'' are în comun cu G'_0 cu o muchie mai mult decât G'. Iterând acest procedeu se obţine un şir de arbori parţiali $G', G'', \ldots, G'_0$ cu $b(G') \leq b(G'') \leq \ldots \leq b(G'_0)$. Deci obţinem $b(G') \leq b(G'_0)$ care contrazice presupunerea că $b(G'_0) < b(G')$. Rezultă că arborele parţial G' este minim. ■

Fig. 3.4.

Între două noduri $y, \overline{y} \in N$ există un lanţ unic $L'_{y\overline{y}}$ în arborele parţial $G' = (N, A')$ al grafului neorientat şi conex $G = (N, A)$.

Teorema 3.7. (Condiţiile de optimalitate ale lanţului)
Fie $G = (N, A)$ un graf neorientat şi conex. Un arbore parţial $G' = (N, A')$ al grafului $G = (N, A)$ este minim dacă şi numai dacă, pentru fiecare muchie nonarbore $[y, \overline{y}] \in \overline{A}'$, avem

$$b[x, \overline{x}] \leq b[y, \overline{y}] \text{ pentru toate muchiile } [x, \overline{x}] \in L'_{y\overline{y}} \tag{3.2}$$

Demonstraţie. Presupunem că arborele parţial $G' = (N, A')$ al grafului $G = (N, A)$ este minim. Prin reducere la absurd presupunem că nu sunt satisfăcute condiţiile (3.2). Atunci există o muchie nonarbore $[y, \overline{y}] \in \overline{A}'$ şi o muchie arbore $[x, \overline{x}] \in L'_{y\overline{y}}$ cu $b[x, \overline{x}] > b[y, \overline{y}]$. Arborele parţial $G'' = (N, A'')$ cu $A'' = A' - \{[x, \overline{x}]\} \cup \{[y, \overline{y}]\}$ are valoarea $b(G'') = b(G') - b[x, \overline{x}] + b[y, \overline{y}]$. Rezultă că $b(G'') < b(G')$ care contrazice faptul că arborele parţial G' este minim. Deci condiţiile (3.2) sunt satisfăcute.
Presupunem că sunt satisfăcute condiţiile (3.2). Fie o muchie arbore oarecare $a = [x, \overline{x}] \in A'$ şi $[X, \overline{X}]_a$ tăietura generată de muchia a. Se consideră o muchie oarecare nonarbore $[y, \overline{y}] \in [X, \overline{X}]_a$. Atunci există lanţul unic $L'_{y\overline{y}}$ în G' astfel încât $[x, \overline{x}] \in L'_{y\overline{y}}$ (vezi figura 3.4). Condiţiile (3.2) implică faptul că $b[x, \overline{x}] \leq b[y, \overline{y}]$. Deoarece muchiile $[x, \overline{x}]$, $[y, \overline{y}]$ sunt oarecare rezultă că sunt satisfăcute condiţiile (3.1). Conform Teoremei 3.6 arborele parţial G' este minim.■
Observaţia 3.2. Suficienţa Teoremei 3.7 a fost demonstrată utilizând suficienţa Teoremei 3.6. Aceasta stabileşte echivalenţa dintre condiţiile de optimalitate ale tăieturii şi condiţiile de optimalitate ale lanţului.

În continuare se prezintă algoritmi pentru determinarea arborelui parţial minim. Mai întâi se prezintă algoritmul generic. Ceilalţi algoritmi sunt cazuri speciale ale algoritmului generic.

3.2.2 Algoritmul generic

Fie $G = (N, A, b)$ o reţea neorientată şi conexă cu mulţimea nodurilor $N = \{1, \ldots, n\}$ şi mulţimea muchiilor $A = \{1, \ldots, a, \ldots, m\}$, $a = [x, \overline{x}]$.

```
(1)   PROGRAM GENERIC;
(2)   BEGIN
(3)      FOR i := 1 TO n DO
(4)      BEGIN
(5)         N_i := {i}; A'_i := ∅;
(6)      END;
(7)      FOR k := 1 TO n − 1 DO
(8)      BEGIN
(9)         se selectează N_i cu N_i ≠ ∅;
(10)        se selectează [y, ȳ] cu b[y, ȳ] := min{b[x, x̄] | x ∈ N_i, x̄ ∉ N_i};
```

(11) se determină indicele j pentru care $\overline{y} \in N_j$;
(12) $N_i := N_i \cup N_j$; $N_j := \emptyset$; $A'_i = A'_i \cup A'_j \cup \{[y, \overline{y}]\}$; $A'_j = \emptyset$;
(13) IF $k = n - 1$
(14) THEN $A' := A'_i$;
(15) END;
(16) END.

Teorema 3.8. *Algoritmul generic determină un arbore parţial minim* $G' = (N, A')$ *al grafului* $G = (N, A)$.

Demonstraţie. Prin inducţie după k, numărul de iteraţii ale ciclului FOR de la linia (7) la linia (15), vom arăta că $G'_i = (N_i, A'_i)$ este conţinut într-un arbore parţial minim al lui G'. Pentru $k = 0$ (înainte de prima iteraţie a ciclului) avem $G'_i = (\{i\}, \emptyset)$ şi afirmaţia este adevărată. Presupunem afirmaţia adevărată pentru execuţia a $k-1$ iteraţii ale ciclului FOR. Aceasta înseamnă că $G'_i = (N_i, A'_i)$, determinat după execuţia a $k-1$ iteraţii ale ciclului FOR este conţinut într-un arbore parţial minim $G' = (N, A')$ al lui G'.

După execuţia a k iteraţii ale ciclului FOR, referitor la muchia $[y, \overline{y}]$ selectată în linia (10), pot exista cazurile:
(c1) $[y, \overline{y}] \in A'$;
(c2) $[y, \overline{y}] \notin A'$.

În cazul (c1) demonstraţia prin inducţie după k s-a terminat. În cazul (c2), există în G' un lanţ unic $L'_{y\overline{y}}$. Deoarece $y \in N_i$ şi $\overline{y} \notin N_i$, evident că există o muchie $[x, \overline{x}]$ astfel încât $[x, \overline{x}] \in L'_{y\overline{y}'}$ cu $x \in N_i$ şi $\overline{x} \notin N_i$. Conform Teoremei 3.7 avem $b[x, \overline{x}] \leq b[y, \overline{y}]$. Pe de altă parte, muchia $[y, \overline{y}]$ este selectată în linia (10) astfel încât $b[x, \overline{x}] \geq b[y, \overline{y}]$. Rezultă că $b[x, \overline{x}] = b[y, \overline{y}]$. Arborele parţial $G'' = (N, A'')$ cu $A'' = A' - \{[x, \overline{x}]\} \cup \{[y, \overline{y}]\}$ are valoarea $b(G'') = b(G') - b[x, \overline{x}] + b[y, \overline{y}] = b(G')$. Deci G'' este un arbore parţial minim al lui G şi conţine $G'_i = (N_i, A'_i)$.

Evident că pentru $k = n-1$ $G'_i = (N_i, A'_i)$ are $N_i = N, |A'_i| = n-1$ şi $G' = G'_i$ este arborele parţial minim al grafului G. ■

Observaţia 3.3. Nu se poate da complexitatea precisă a algoritmului generic, deoarece ea depinde atât de modul de selectare a mulţimii N_i în linia (9) cât şi de modul de implementare.

În algoritmii care vor fi prezentaţi în continuare se precizează selectarea din liniile (9) şi (10) ale algoritmului generic.

3.2.3 Algoritmul Prim

Fie $G = (N, A, b)$ o reţea neorientată şi conexă cu mulţimea nodurilor $N = \{1, \ldots, n\}$ şi mulţimea muchiilor $A = \{1, \ldots, a, \ldots, m\}$, $a = [x, \overline{x}]$. Graful G este reprezentat prin listele de incidenţă $E(x)$. Algoritmul utilizează două funcţii: $v : N \to \Re$ şi $e : N \to A$ pentru a selecta mai uşor muchia $[y, \overline{y}]$ din linia (10) a

algoritmului generic.

```
(1)   PROGRAM PRIM;
(2)   BEGIN
(3)      v(1) := 0; N_1 := ∅; A' := ∅; N̄_1 := N − N_1;
(4)      FOR i := 2 TO n DO
(5)         v(i) := ∞;
(6)      WHILE N_1 ≠ N DO
(7)      BEGIN
(8)         v(y) := min{v(x̄) | x̄ ∈ N̄_1};
(9)         N_1 := N_1 ∪ {y}; N̄_1 := N̄_1 − {y};
(10)        IF y ≠ 1
(11)           THEN A' := A' ∪ {e(y)};
(12)        FOR [y, ȳ] ∈ E(y) ∩ [N_1, N̄_1] DO
(13)           IF v(ȳ) > b[y, ȳ]
(14)              THEN BEGIN v(ȳ) := b[y, ȳ], e(ȳ) := [y, ȳ];END;
(15)     END;
(16)  END.
```

Teorema 3.9. *Algoritmul Prim determină un arbore parţial minim $G' = (N, A')$ al grafului $G = (N, A)$.*

Demonstraţie. Algoritmul Prim s-a obţinut făcând următoarele două modificări în algoritmul generic:

- în linia (9) se selectează întotdeauna N_1;
- selectarea muchiei $[y, \overline{y}]$ în linia (10) se face cu ajutorul funcţiilor v şi e.

Deci, conform Teoremei 3.8 algoritmul Prim este corect.■

Teorema 3.10. *Algoritmul Prim are complexitatea $O(n^2)$.*

Demonstraţie. Ciclul WHILE se execută de n ori. În fiecare iteraţie se execută, în linia (8), $\overline{n}_1$ comparaţii cu $\overline{n}_1 = |\overline{N}_1|$ şi $1 \leq \overline{n}_1 < n$. Ciclul FOR se execută de cel mult $n-1$ ori. Rezultă că algoritmul Prim are complexitatea $O(n^2)$. ■

Exemplul 3.4. Fie reţeaua $G = (N, A, b)$ reprezentată în figura 3.5(a). Să se determine un arbore parţial minim $G' = (N, A')$ al grafului $G = (N, A)$ cu algoritmul Prim.

Iteraţia 1: $v(1) = 0,\ N_1 = \{1\},\ A' = \emptyset;\ v(2) = 35,\ e(2) = [1,2],\ v(3) = 40,\ e(3) = [1,3.]$

Iteraţia 2: $v(2) = 35,\ N_1 = \{1,2\},\ A' = \{[1,2]\};\ v(3) = 25,\ e(3) = [2,3],\ v(4) = 10,\ e(4) = [2,4.]$

Iteraţia 3: $v(4) = 10,\ N_1 = \{1,2,4\},\ A' = \{[1,2],\ [2,4]\};\ v(3) = 20,\ e(3) = [4,3],\ v(5) = 30,\ e(5) = [4,5.]$

Iteraţia 4: $v(3) = 20,\ N_1 = \{1,2,4,3\},\ A' = \{[1,2],\ [2,4],\ [4,3]\};\ v(5) =$

15, $e(5) = [3, 5]$.
Iteraţia 5: $v(5) = 15,\ N_1 = \{1, 2, 4, 3, 5\},\ A' = \{[1, 2],\ [2, 4],\ [4, 3],\ [3, 5]\}$;
$[N_1, \overline{N}_1] = \emptyset$.
Arborele parţial minim $G' = (N, A')$ obţinut este reprezentat în figura 3.5(b).
Fig. 3.5.

3.2.4 Algoritmul Kruskal

Fie $G = (N, A, b)$ o reţea neorientată şi conexă cu mulţimea nodurilor $N = \{1, \ldots, n\}$ şi mulţimea muchiilor $A = \{1, \ldots, a, \ldots, m\},\ a = [x, \overline{x}]$.

```
(1)   PROGRAM KRUSKAL;
(2)   BEGIN
(3)      SORTARE (G);
(4)      A' := ∅;
(5)      FOR i := 1 TO m DO
(6)         IF a_i nu formează ciclu cu A';
(7)            THEN A' := A' ∪ {a_i};
(8)      END.
```

```
(1)   PROCEDURA SORTARE (G);
(2)   BEGIN
(3)      se ordonează muchiile din A astfel încât A = {a_1, ..., a_m}
           cu b(a_1) ≤ ... ≤ b(a_m);
(4)   END;
```

Teorema 3.11. *Algoritmul Kruskal determină un arbore parţial minim* $G' = (N, A')$ *al grafului* $G = (N, A)$.
Demonstraţie. Algoritmul Kruskal s-a obţinut folosind procedura SORTARE şi condiţia din linia (6) pentru selectările din linia (9) şi (10) din algoritmul generic. Deci, conform Teoremei 3.8 algoritmul Kruskal este corect.■
Observaţia 3.4. Complexitatea algoritmului Kruskal depinde de modul de implementare a procedurii din linia (3) şi a condiţiei din linia (6). Astfel complexitatea poate fi $O(max(m \log n,\ n^2))$ sau $O(m \log n)$ etc.
Exemplul 3.5. Fie reţeaua $G = (N, A, b)$ reprezentată în figura 3.5(a). Să se determine un arbore parţial minim $G' = (N, A')$ al grafului $G = (N, A)$ cu algoritmul Kruskal.

Procedura SORTARE ordonează muchiile din A şi se obţine $A = \{[2, 4], [3, 5], [3, 4], [2, 3], [4, 5], [1, 2], [1, 3]\}$.
Iteraţia 1: $A' = \{[2, 4]\}$

Iteraţia 2: $A' = \{[2,4],[3,5]\}$
Iteraţia 3: $A' = \{[2,4],[3,5],[3,4]\}$
Iteraţia 4: $[2,3]$ formează ciclu cu A'
Iteraţia 5: $[4,5]$ formează ciclu cu A'
Iteraţia 6: $A' = \{[2,4],[3,5],[3,4],[1,2]\}$
Iteraţia 7: $[1,3]$ formează ciclu cu A'.
Arborela parţial minim $G' = (N, A')$ obţinut este prezentat în figura 3.5(b).

3.2.5 Algoritmul Boruvka

Fie $G = (N, A, b)$ o reţea neorientată şi conexă cu mulţimea nodurilor $N = \{1, \ldots, n\}$ şi mulţimea muchiilor $A = \{1, \ldots, a, \ldots, m\}$, $a = [x, \overline{x}]$. Funcţia valoare $b : A \to \Re$ are toate valorile muchie diferite.

```
(1)   PROGRAM BORUVKA;
(2)   BEGIN
(3)     FOR i := 1 TO n DO
(4)       N_i := {i};
(5)     A' := ∅; M := {N_1, ..., N_n};
(6)     WHILE |A'| < n − 1 DO
(7)     BEGIN
(8)       FOR N_i ∈ M DO
(9)       BEGIN
(10)        se selectează [y, ȳ] cu b[y, ȳ] := min{b[x, x̄] | x ∈ N_i,
              x̄ ∉ N_i};
(11)        se determină indicele j pentru care ȳ ∈ N_j;
(12)        A' := A' ∪ {[y, ȳ]};
(13)      END;
(14)      FOR N_i ∈ M DO BEGIN N_i := N_i ∪ N_j; N_j := N_i; END;
(15)      M := {..., N_i, ..., N_j, ... | N_i ∩ N_j = ∅, ∪N_i = N};
(16)    END;
(17)  END.
```

Mulţimile N_j din operaţia $N_i := N_i \cup N_j$ executată în linia (14) sunt cele determinate în linia (11).

Teorema 3.12. *Algoritmul Boruvka determină un arbore parţial minim* $G' = (N, A')$ *al grafului* $G = (N, A)$.

Demonstraţie. Este evident că algoritmul Boruvka este o variantă a algoritmului generic. Ipoteza că funcţia valoare b are toate valorile muchie diferite asigură că nu se creează cicluri în $G' = (N, A')$ în timpul execuţiei ciclului WHILE. Deci, conform Teoremei 3.8 algoritmul Boruvka este corect.■

Teorema 3.13. *Algoritmul Boruvka are complexitatea* $O(m \log n)$.
Demonstraţie. Deoarece numărul componentelor conexe N_i din M este cel puţin înjumătăţit la fiecare iteraţie, ciclul WHILE este executat de cel mult $\log n$ ori. Operaţiile din liniile (10), (11) au complexitatea $O(m)$. Deci algoritmul Boruvka are complexitatea $O(m \log n)$.■
Observaţia 3.5. Algoritmul Boruvka este cunoscut în literatura de specialitate şi sub numele de algoritmul Sollin.
Exemplul 3.6. Fie reţeaua $G = (N, A, b)$ reprezentată în figura 3.5(a). Să se determine un arbore parţial minim $G' = (N, A')$ al grafului $G = (N, A)$ cu algoritmul Boruvka.

Iniţial $A' = \emptyset,\ M = \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$.
Iteraţia 1.
$N_1,\ [y, \overline{y}] = [1, 2],\ N_j = N_2,\ A' = \{[1, 2]\}$;
$N_2,\ [y, \overline{y}] = [2, 4],\ N_j = N_4,\ A' = \{[1, 2], [2, 4]\}$;
$N_3,\ [y, \overline{y}] = [3, 5],\ N_j = N_5,\ A' = \{[1, 2], [2, 4], [3, 5]\}$;
$N_4,\ [y, \overline{y}] = [4, 2],\ N_j = N_2,\ A' = \{[1, 2], [2, 4], [3, 5]\}$;
$N_5,\ [y, \overline{y}] = [5, 3],\ N_j = N_3,\ A' = \{[1, 2], [2, 4], [3, 5]\}$;
$M = \{N_4 = \{1, 2, 4\},\ N_5 = \{3, 5\}\}$;
Iteraţia 2.
$N_4,\ [y, \overline{y}] = [4, 3],\ N_j = N_5,\ A' = \{[1, 2], [2, 4], [3, 5], [4, 3]\}$;
$N_5,\ [y, \overline{y}] = [3, 4],\ N_j = N_4,\ A' = \{[1, 2], [2, 4], [3, 5], [4, 3]\}$;
$M = \{N_5 = \{1, 2, 3, 4, 5\}\}$.

Arborele parţial minim $G' = (N, A')$ obţinut este prezentat în figura 3.5(b).

3.3 Arborescenţe

Definiţia 3.5. Un nod $r \in N$ al unui digraf $G = (N, A)$ se numeşte *nod rădăcină* dacă pentru orice alt nod $x \in N$ există un drum de la r la x.
Observaţia 3.6. Noţiunea de nod rădăcină are sens numai pentru grafuri orientate. Un nod rădăcină nu există întotdeauna.
Exemplul 3.7. Nodul 1 al digrafului reprezentat în figura 3.6 este un nod rădăcină.

Fig. 3.6.

Definiţia 3.6. Un digraf $G = (N, A)$ se numeşte *quasi-tare conex* dacă pentru oricare două noduri $x, y \in N$ există un nod $z \in N$ (care poate coincide cu x sau cu y) de la care pleacă un drum care ajunge în x şi un drum care ajunge în y.
Observaţia 3.7. Un digraf tare conex este quasi tare conex, dar reciproca nu este adevărată. Un digraf quasi tare conex este conex, dar reciproca nu este adevărată.
Definiţia 3.7. Un digraf $G' = (N, A')$ se numeşte *arborescenţă* dacă este fără

cicluri şi quasi tare conex.

Exemplul 3.8. Digraful reprezentat în figura 3.6 este o arborescenţă.

Teorema 3.14. *Un digraf $G = (N, A)$ admite un nod rădăcină dacă şi numai dacă el este quasi tare conex.*

Demonstraţie. Dacă digraful G admite un nod rădăcină atunci el este quasi tare conex deoarece pentru oricare două noduri $x, y \in N$ există un nod z (de exemplu nodul rădăcină r) de la care pleacă un drum care ajunge în x şi un drum care ajunge în y.

Dacă digraful $G = (N, A)$, $|N| = n$ este quasi tare conex, atunci există un nod z_1 de la care pleacă un drum care ajunge în x_1 şi un drum care ajunge în x_2; există un nod z_2 de la care pleacă un drum care ajunge în z_1 şi un drum care ajunge în x_3. Continuând acest procedeu există un nod z_{n-1} de la care pleacă un drum care ajunge în z_{n-2} şi un drum care ajunge în x_n. Este evident că nodul $r = z_{n-1}$ este nod rădăcină al digrafului $G = (N, A)$. ■

Teorema 3.15. *Fie $G' = (N, A')$ un digraf de ordinul $n \geq 2$. Proprietăţile următoare sunt echivalente şi caracterizează o arborescenţă:*

(1). G' este fără cicluri şi quasi tare conex;

(2). G' este quasi tare conex şi are $n-1$ arce;

(3). G' este un arbore şi admite un nod rădăcină r;

(4). G' conţine un nod r şi un drum unic de la nodul r la oricare alt nod $x \in N$;

(5). G' este quasi tare conex şi prin eliminarea unui arc oarecare se obţine digraful $\widehat{G}' = \left(N, \widehat{A}'\right)$ care nu este quasi tare conex;

(6). G' este conex şi conţine un nod r astfel încât $\rho^-(r) = 0,\ \rho^-(x) = 1$ pentru oricare nod $x \neq r$;

(7). G' este fără cicluri şi conţine un nod r astfel încât $\rho^-(r) = 0,\ \rho^-(x) = 1$ pentru oricare nod $x \neq r$.

Demonstraţie.

(1) $\Rightarrow$ (2). Dacă (1) este adevărată, atunci G' este conex şi fără cicluri, adică G' este un arbore. Deci G' are $n-1$ arce şi (2) este adevărată.

(2) $\Rightarrow$ (3). Dacă (2) este adevărată, atunci G' este conex şi are $n-1$ arce, adică G' este un arbore. În plus, conform Teoremei 3.14 G' admite un nod rădăcină r şi (3) este adevărată.

(3) $\Rightarrow$ (4). Dacă (3) este adevărată, atunci există un drum de la nodul r la oricare alt nod $x \in N$. Deoarece G' este arbore rezultă că acest drum este unic.

(4) $\Rightarrow$ (5). Dacă (4) este adevărată, atunci nodul r este un nod rădăcină şi conform Teoremei 3.14 G' este quasi tare conex. Fie un arc oarecare $(x, y) \in A'$. Construim digraful $\widehat{G}' = (N, \widehat{A}')$, unde $\widehat{A}' = A' - \{(x, y\})$. Prin reducere la absurd, presupunem că digraful $\widehat{G}'$ este quasi tare conex. Deci în $\widehat{G}'$ există un drum D_1 de la un nod z la nodul x şi un drum D_2 de la nodul z la nodul y. Rezultă că în G' există două drumuri de la z la y(D_2 şi $D_3 = D_1 \cup \{(x, y)\}$). Prin urmare există

două drumuri de la r la y ce contrazice (4). Deci $\widehat{G}'$ nu este quasi tare conex şi (5) este adevărată.
(5) $\Rightarrow$ (6). Dacă (5) este adevărată, atunci G' este conex şi conform Teoremei 3.14, admite un nod rădăcină r. Deci $\rho^-(r) \geq 0$ şi $\rho^-(x) \geq 1$ pentru oricare nod $x \neq r$. Dacă $\rho^-(r) > 0$, atunci există cel puţin un arc $(y, r) \in A'$. Digraful $\widehat{G}' = (N, \widehat{A}')$ cu $\widehat{A}' = A' - \{(y, r)\}$, admite evident nodul r ca nod rădăcină, adică $\widehat{G}'$ este quasi tare conex ce contrazice (5) şi deci $\rho^-(r) = 0$. Dacă $\rho^-(x) > 1$, atunci există cel puţin două arce $(y, x), (z, x)$ în A'. Aceasta înseamnă că există cel puţin două drumuri distincte de la r la x. Digraful $\widehat{G}' = (N, \widehat{A}')$ cu $\widehat{A}' = A' - \{(y, x)\}$ admite nodul r ca nod rădăcină, adică $\widehat{G}'$ este quasi tare conex ce contrazice (5) şi deci $\rho^-(x) = 1$. Rezultă că (6) este adevărată.
(6) $\Rightarrow$ (7). Dacă (6) este adevărată, atunci numărul de arce al digrafului $G' = (N, A')$ este $m' = \sum_N \rho^-(x) = n - 1$, adică G' este un arbore. Rezultă că G' este fără cicluri şi (7) este adevărată.
(7) $\Rightarrow$ (1). Dacă (7) este adevărată, atunci nodul r este evident un nod rădăcină. Conform Teoremei 3.14 digraful G' este quasi tare conex şi (1) este adevărată.■

Teorema 3.16. *Un digraf $G = (N, A)$ admite un subgraf parţial $G' = (N, A')$ care este o arborescenţă dacă şi numai dacă el este quasi tare conex.*

Demonstraţie. Dacă digraful G admite un subgraf parţial G' care este o arborescenţă atunci el admite un nod rădăcină r şi conform Teoremei 3.14 digraful G este quasi tare conex.

Reciproc, dacă digraful G este quasi tare conex, atunci conform proprietăţii (5) din Teorema 3.15 se poate obţine un subgraf parţial $G' = (N, A')$ care este o arborescenţă cu procedura următoare:

```
(1)  PROCEDURA ARBORESCENTA;
(2)  BEGIN
(3)     A' := A;
(4)     FOR i := 1 TO m DO
(5)        IF prin eliminarea arcului a_i ∈ A' noul digraf rămâne quasi
              tare conex
(6)           THEN A' := A' - {a_i}
(7)  END.
```

3.4 Aplicaţii şi comentarii bibliografice

Problema arborelui parţial minim apare în aplicaţii în două moduri: direct şi indirect. În aplicaţiile directe se doreşte conectarea unor puncte prin legături care au asociate lungimi sau costuri. Punctele reprezintă entităţi fizice ca utilizatori ai unui sistem sau componente ale unui cip care trebuie conectate între ele sau

cu un alt punct ca procesorul principal al unui calculator. În aplicaţiile indirecte problema practică nu apare evident ca o problemă a arborelui parţial minim şi în acest caz este necesar să modelăm problema practică astfel încât să o transformăm într-o problemă a arborelui parţial minim. În continuare se vor prezenta câteva aplicaţii directe şi indirecte ale arborelui parţial minim.

3.4.1 Proiectarea unui sistem fizic

Proiectarea unui sistem fizic constă în proiectarea unei reţele care conectează anumite componente. Sistemul fizic trebuie să nu aibă redundanţe ceea ce conduce la determinarea unui arbore parţial minim. Se prezintă în continuare câteva exemple.
1. Conectarea terminalelor din tablourile electrice astfel încât lungimea totală a firelor folosite să fie minimă.
2. Construirea unei reţele de conducte care să lege un număr de oraşe astfel încât lungimea totală a conductelor să fie minimă.
3. Conectarea telefonică a unor comune dintr-o zonă izolată astfel încât lungimea totală a liniilor telefonice care leagă oricare două comune să fie minimă.
4. Determinarea configuraţiei unei reţele de calculatoare astfel încât costul conectării în reţea să fie minim.

Fiecare dintre aceste aplicaţii este o aplicaţie directă a problemei arborelui parţial minim. În continuare se prezintă două aplicaţii indirecte.

3.4.2 Transmiterea optimă a mesajelor

Un serviciu de informaţii are n agenţi într-o anumită ţară. Fiecare agent cunoaşte o parte din ceilalţi agenţi şi poate stabili întâlniri cu oricare din cei pe care îi cunoaşte. Orice mesaj transmis la întâlnirea dintre agentul i şi agentul j poate să ajungă la inamic cu probabilitatea $p(i,j)$. Conducătorul grupului vrea să transmită un mesaj confidenţial tuturor agenţilor astfel încât probabilitatea totală ca mesajul să fie interceptat să fie minimă.

Dacă reprezentăm agenţii prin noduri şi fiecare întâlnire posibilă dintre doi agenţi printr-o muchie, atunci în graful rezultat $G = (N, A)$ dorim să identificăm un arbore parţial $G' = (N, A')$ care minimează probabilitatea interceptării dată de expresia $\left(1 - \prod_{A'}(1 - p(i,j))\right)$. Deci vrem să determinăm un arbore parţial $G' = (N, A')$ care maximizează $\left(\prod_{A'}(1 - p(i,j))\right)$. Putem determina un astfel de arbore dacă definim lungimea arcului (i,j) ca fiind $\log(1 - p(i,j))$ şi rezolvând problema arborelui parţial maxim.

3.4.3 Problema lanţului minimax între oricare două noduri

Într-o reţea $G = (N, A, b)$, definim valoarea unui lanţ $L_{z\overline{z}}$ de la nodul z la nodul $\overline{z}$ ca fiind valoarea maximă a muchiilor din $L_{z\overline{z}}$. Problema lanţului minimax între oricare două noduri constă în a determina între oricare două noduri $z, \overline{z}$ un lanţ de valoare minimă.

Problema lanţului minimax apare în multe situaţii. De exemplu, considerăm o navă spaţială care trebuie să intre în atmosfera Pământului. Nava trebuie să treacă prin zone cu temperaturi diferite pe care le putem reprezenta prin muchii într-o reţea. Pentru a ajunge pe suprafaţa Pământului trebuie aleasă o traiectorie astfel încât temperatura maximă la care urmează să fie expusă nava să fie minimă.

Problema lanţului minimax între oricare două noduri din reţeaua $G = (N, A, b)$ este o problemă a arborelui parţial minim. Fie $G' = (N, A')$ un arbore parţial minim al grafului $G = (N, A)$ şi $L'_{z\overline{z}}$ lanţul unic de la z la $\overline{z}$ în G'. Dacă $[x, \overline{x}]$ este muchia din $L'_{z\overline{z}}$ cu valoarea maximă atunci lanţul $L'_{z\overline{z}}$ are valoarea $b\,[x, \overline{x}]$. Muchia $a = [x, \overline{x}]$ generează tăietura $\left[X, \overline{X}\right]_a$ şi conform condiţiilor de optimalitate ale tăieturii avem

$$b\,[x, \overline{x}] \leq b\,[y, \overline{y}] \text{ pentru toate muchiile } [y, \overline{y}] \in \left[X, \overline{X}\right]_a$$

Oricare alt lanţ $L_{z\overline{z}}$ de la z la $\overline{z}$ conţine o muchie $[y, \overline{y}] \in \left[X, \overline{X}\right]_a$ şi valoarea acestui lanţ este, conform condiţiilor de mai sus, cel puţin $b\,[x, \overline{x}]$. Rezultă că $L'_{z\overline{z}}$ este un lanţ de valoare minimă de la z la $\overline{z}$. Deci laţul unic dintre oricare două noduri din G' este un lanţ de valoare minimă dintre oricare două noduri din G.

În continuare se prezintă două aplicaţii ale arborilor binari.

3.4.4 Comentarii bibliografice

Algoritmii pentru problema arborelui parţial minim, apăruţi încă din 1926, sunt printre primii algoritmi de optimizare în reţele. Articolul lui Graham şi Hell (1985) prezintă un excelent rezumat al dezvoltărilor istorice pentru algoritmii arborelui parţial minim. Boruvka (1926) şi Jarnik (1930) independent au formulat şi rezolvat problema arborelui parţial minim. Mai târziu, alţi cercetători au redescoperit aceşti algoritmi. Kruskal (1956) şi Prim (1957) au descoperit algoritmii descrişi în paragraful 3.2. Sollin a prezentat algoritmul său într-un seminar din 1961. Mai târziu, cercetătorii au constatat că algoritmul lui Sollin este similar cu algoritmul lui Boruvka şi că algoritmul lui Prim este similar cu algoritmul lui Jarnik.

Algoritmul lui Prim poate fi implementat utilizând tipuri diferite de ansambluri (heaps).

Implementarea cu d-ansamblu conduce la o complexitate $O\,(m \log_d n)$, cu $d = \max\{2, m/n\}$. Ansamblul binar este un caz special al d-ansamblului cu $d = 2$, astfel complexitatea algoritmului este $O(m \log n)$. Ansamblul Fibonacci utilizat în implementarea algoritmului Prim conduce la o complexitate $O(m + n \log n)$. Gabow, Galil, Spencer şi Tarjan (1986) au prezentat o implementare a algoritmului Prim care are complexitatea $O\,(m \log \beta(m, n))$ cu funcţia $\beta(m, n)$ definită ca $\beta(m, n) = \min\{i \mid \log^i(m/n) \leq 1\}$. În această expresie, $\log^i x = \log \log \ldots \log x$ cu

log iterat de i ori şi $\beta(m,n)$ este o funcţie crescătoare foarte lent. Implementarea algoritmului Kruskal datorată lui Tarjan (1984) conduce la complexitatea sortării a m numere plus $O(m\alpha(n,n))$ cu $\alpha(n,m)$ funcţia lui Ackermann care, pentru toate scopurile practice, este mai mică decât 6. Yao (1975) prezintă o implementare a algoritmului Boruvka care conduce la complexitatea $O(m \log\log n)$. Uzual, cel mai bun algoritm pentru rezolvarea problemei arborelui parţial minim este implementarea lui Tarjan (1984) a algoritmului Kruskal dacă arcele sunt deja sortate, altfel este implementarea lui Gabow şi alţii (1986) a algoritmului Prim. De asemenea, Gabow şi alţii (1986) prezintă algoritmi eficienţi care rezolvă problema arborelui parţial minim într-o reţea orientată (problema arborescenţei) şi rezolvă problema arborelui parţial minim cu o singură constrângere grad. Un algoritm cu complexitate liniară a fost descoperit de Fredman şi Willard (1994) dacă arcele sunt deja sortate în raport cu valoarea lor. Din nefericire, algoritmii cei mai buni din punct de vedere teoretic nu prezintă un mare interes practic. Matsui (1995) prezintă un algoritm simplu cu complexitatea $O(n)$ pentru grafuri planare.

Problema determinării unui nou arbore parţial minim dacă se schimbă valoarea unei muchii a fost discutată de Frederickson (1985) şi Eppstein (1994). Problema *analizei sensibilităţii* constă în creşterea valorii unei muchii date astfel încât să nu se schimbe arborele parţial minim deja cunoscut. Această problemă este prezentată în Dixon, Rauch şi Tarjan (1992).

Aplicaţii ale problemei arborelui parţial minim pot fi găsite în: Boruvka (1926), Prim (1957), Dijkstra (1959), Hu (1961), Held şi Karp (1970), Magnanti şi Wong (1984), Graham şi Hell (1985), Ahuja, Magnanti şi Orlin (1993), Gross şi Yellen (1999.)

Capitolul 4

Distanţe şi drumuri minime

4.1 Principalele probleme de drum minim

Se consideră un digraf $G = (N, A)$ cu $N = \{1, \ldots, n\}$, $A = \{a_1, \ldots, a_m\}$ şi o funcţie valoare $b : A \to \Re$ care asociază fiecărui arc $a \in A$ un număr real $b(a)$ numit *valoarea arcului a*. Această valoare poate fi interpretată ca lungime sau cost etc.

Vom nota cu D_{xyk} un drum în digraful G de la nodul x la nodul y şi cu $\mathcal{D}_{xy} = \{D_{xy1}, \ldots, D_{xyr}\}$ mulţimea acestor drumuri.

Valoarea unui drum $D_{xyk} = (a_1, \ldots, a_q)$, notată $b(D_{xyk})$, este numărul $b(D_{xyk}) = b(a_1) + \ldots b(a_q)$.

Definiţia 4.1. Un drum D_{xyp} cu valoarea $b(D_{xyp}) = \min\{b(D_{xyk}) \mid D_{xyk} \in \mathcal{D}_{xy}\}$ se numeşte *drum de valoare minimă* sau *drum minim*. Valoarea $b(D_{xyp})$ a drumului minim D_{xyp} se numeşte *distanţă* de la nodul x la nodul y şi o notăm $d(x, y)$.

Există două probleme cu dificultăţi privind această definiţie. Prima, poate să nu existe drum de la nodul x la nodul y şi a doua, poate exista drum D_{xyk} cu valoare arbitrar de mică. Prima problemă poate fi rezolvată dacă se defineşte $d(x, y) = \infty$. A doua problemă provine din posibilitatea existenţei circuitelor de valoare negativă în reţeaua $G = (N, A, b)$.

Exemplul 4.1. În reţeaua reprezentată în figura 4.1, putem determina un drum de valoare arbitrar de mică de la nodul 1 la nodul 5 utilizând circuitul de valoare negativă (2, 3, 4, 2), ori de câte ori este necesar. Vom presupune că reţeaua

Fig. 4.1.

$G = (N, A, b)$ nu conţine circuite cu valoare negativă.

Principalele probleme de drum minim care apar în aplicaţii practice sau sunt

subprobleme în rezolvarea altor probleme de optimizare în reţele sunt următoarele:
(PDM1) Determinarea unui drum minim D_{stp} de la un nod precizat s la un alt nod precizat t şi a valorii $b(D_{stp})$.
(PDM2) Determinarea unui drum minim D_{syp} de la un nod precizat s la nodul y şi a valorii $b(D_{syp})$, pentru toate nodurile $y \in N,\ y \neq s$.
(PDM3) Determinarea unui drum minim D_{xyp} de la un nod x la nodul y şi a valorii $b(D_{xyp})$, pentru toate nodurile $x, y \in N,\ x \neq y$.

Dacă $G' = (N', A')$ este un graf neorientat şi $b' : A' \to \Re^+$, atunci problemele de lanţ minim din reţeaua neorientată $G' = (N', A')$ pot fi reduse la problemele de drum minim din reţeaua orientată $G = (N, A, b)$ cu $N = N', A = \{(x, y),\ (y, x) \mid [x, y] \in A'\}$ şi $b(x, y) = b(y, x) = b'[x, y]$.

PDM1 se poate rezolva utilizând un algoritm de rezolvare a PDM2 la care se adaugă un test suplimentar de oprire; PDM3 se poate rezolva iterând după s un algoritm de rezolvare a PDM2. Există însă algoritmi eficienţi care rezolvă direct PDM3. De aceea în acest capitol se vor prezenta algoritmi de rezolvare pentru PDM2 şi PDM3.

Observaţia 4.1. Dacă $d(x, y) = \infty$ atunci considerăm că există un drum D_{xyk} cu $b(D_{xyk}) = \infty$.

4.2 Ecuaţiile lui Bellman

Fie reţeaua orientată $G = (N, A, b)$ şi presupunem că G nu conţine circuite cu valoare negativă. Pentru reprezentarea reţelei G se va folosi matricea valoare adiacenţă $B = (b_{ij})$ cu $i, j \in N$, unde

$$b_{ij} = \begin{cases} b(i,j) & \text{dacă} \quad i \neq j \text{ şi } (i,j) \in A; \\ 0 & \text{dacă} \quad i = j; \\ \infty & \text{dacă} \quad i \neq j \text{ şi } (i,j) \notin A. \end{cases}$$

Fără a restrânge generalitatea presupunem că nodul s este nodul 1. Dacă $D_{1jp} = ((1, h), \ldots, (k, i), (i, j))$ este un drum minim de la nodul 1 la nodul j, atunci $D_{1ip} = ((1, h), \ldots, (k, i))$ este un drum minim de la nodul 1 la nodul i, altfel contrazicem faptul că D_{1jp} este drum minim. Astfel distanţele $u_j = d(1, j)$ trebuie să satisfacă ecuaţiile lui Bellman:

$$u_1 = 0,\ u_j = \min\{u_i + b_{ij} \mid i \neq j\},\ j = 2, \ldots, n. \tag{4.1}$$

Fără a restrânge generalitatea, putem presupune că nodul 1 este un nod rădăcină al reţelei orientate G.

Teorema 4.1. *Dacă în reţeaua orientată $G = (N, A, b)$ nodul 1 este nod rădăcină şi G conţine numai circuite $\overset{\circ}{D}$ cu $b\left(\overset{\circ}{D}\right) > 0$ atunci G conţine o arborescenţă*

parţială $G' = (N, A')$ cu nodul rădăcină 1 şi drumul unic de la 1 la oricare alt nod j din G' este un drum minim D_{1jp} din G.

Demonstraţie. Dacă în reţeaua orientată $G = (N, A, b)$ nodul 1 este nod rădăcină atunci conform Teoremei 3.14 şi Teoremei 3.16, G admite o arborescenţă parţială $G' = (N, A')$ cu nodul rădăcină 1. Considerăm un nod j şi determinăm un drum D_{1jp}, $u_j = b(D_{1jp})$, cu următorul procedeu. Deoarece sunt verificate ecuaţiile (4.1), selectăm un arc (i, j) astfel încât $u_j = u_i + b_{ij}$. În continuare selectăm un arc (k, i) astfel încât $u_i = u_k + b_{ki}$ etc. până când se ajunge la nodul 1. Presupunem prin reducere la absurd că nu se ajunge în nodul 1. Aceasta este posibil dacă procedeul conduce la un circuit $\overset{\circ}{D} = (v, w, \ldots, q, v)$. Avem următoarele ecuaţii:$u_v = u_q + b_{qv} = \ldots = u_v + b_{vw} + \ldots + b_{qv}$. Rezultă că $b(\overset{\circ}{D}) = b_{vw} + \ldots + b_{qv} = u_v - u_v = 0$, care contrazice ipoteza că reţeaua G conţine circuite $\overset{\circ}{D}$ cu $b\left(\overset{\circ}{D}\right) > 0$. Deci procedeul determină un drum D_{1jp} cu $u_j = b(D_{1jp})$. Astfel, aplicând procedeul pentru toate nodurile j, pentru care nu s-a determinat deja un drum D_{1jp}, se obţine arborescenţa parţială $G' = (N, A')$ cu proprietatea din enunţul teoremei. ■

Următoarea teoremă întăreşte rezultatul din Teorema 4.1.

Teorema 4.2. *Dacă în reţeaua orientată $G = (N, A, b)$ nodul 1 este nod rădăcină şi G nu conţine circuite $\overset{\circ}{D}$ cu $b\left(\overset{\circ}{D}\right) < 0$, atunci G conţine o arborescenţă parţială $G' = (N, A')$ cu nodul rădăcină 1 şi drumul unic de la 1 la oricare alt nod j din G' este un drum minim D_{1jp} din G.*

Demonstraţie. Dacă în reţeaua orientată $G = (N, A, b)$ nodul 1 este nod rădăcină, atunci conform Teoremei 3.14 şi Teoremei 3.16, G admite o arborescenţă parţială $G' = (N, A')$ cu nodul rădăcină 1. Considerăm un nod j şi un drum minim $D_{1jp} = ((1, h), \ldots, (k, i), (i, j))$. Atunci avem:

$$d(1, j) = d(1, i) + b(i, j).$$

Deoarece nodul j a fost selectat arbitrar, putem să selectăm un arc (i, j) care verifică condiţia anterioară pentru orice nod $j \neq 1$. În acest mod putem alege $n-1$ arce şi fie A' mulţimea acestor arce. Este evident că în digraful $G' = (N,\ A')$ sunt verificate relaţiile $\rho^{-1} = 0,\ \rho^-(j) = 1,\ \ j = 2, \ldots, n$. Rezultă că $G' = (N, A')$ este o arborescenţă parţială cu nodul rădăcină 1. Drumul unic de la 1 la j din G' este un drum minim D_{1jp} din G, deoarece fiecare arc din A' verifică condiţia de mai sus. ■

4.3 Algoritmi pentru distanţe şi drumuri minime

4.3.1 Algoritmul Dijkstra

Fie reţeaua orientată $G = (N, A, b)$ cu $b : A \to \Re^+$. În acest caz ecuaţiile lui Bellman (4.1) pot fi rezolvate cu algoritmul Dijkstra. Lista nodurilor adiacente către exterior nodului x este $V^+(x) = \{y \mid (x, y) \in A\}$. Algoritmul Dijkstra este următorul:

```
(1)   PROGRAM DIJKSTRA;
(2)   BEGIN
(3)     W := N; d(s) := 0; p(s) := 0;
(4)     FOR y ∈ N − {s} DO
(5)     BEGIN
(6)       d(y) := ∞; p(y) := 0;
(7)     END;
(8)     WHILE W ≠ ∅ DO
(9)     BEGIN
(10)      se selectează un nod x ∈ W astfel încât d(x) este minimă;
(11)      W := W − {x};
(12)      FOR y ∈ W ∩ V+(x) DO
(13)        IF d(x) + b(x, y) < d(y)
(14)          THEN BEGIN d(y) := d(x) + b(x, y); p(y) := x; END;
(15)    END;
(16)  END.
```

Teorema 4.3. *Algoritmul Dijkstra determină distanţele $d(s, y)$ şi drumurile minime D_{syp}, $y \in N$, $y \neq s$, în raport cu nodul sursă s din reţeaua orientată $G = (N, A, b)$ cu $b : A \to \Re^+$.*

Demonstraţie. Din modul cum este calculată în algoritm valoarea $d(y)$ rezultă că $d(y)$ reprezintă valoarea unui drum de la nodul s la nodul y din reţeaua G. Deci are loc relaţia $d(s, y) \leq d(y)$ pentru fiecare nod $y \in N$. Trebuie să arătăm că $d(s, y) = d(y)$ pentru fiecare $y \in N$. Pentru aceasta vom demonstra prin inducţie după ordinea în care nodurile sunt eliminate din W că $d(y) \leq d(s, y)$ pentru fiecare nod $y \in N$.

Primul nod eliminat este s şi evident că $d(s) = 0 = d(s, s)$. Presupunem că $d(y) \leq d(s, y)$ pentru toate nodurile y care au fost eliminate din W înaintea nodului z. Fie $D_{szp} = (x_0, x_1, \ldots, x_{k-1}, x_k)$ un drum minim de la $x_0 = s$ la $x_k = z$. Atunci pentru $h = 1, \ldots, k$ avem

$$d(s, x_h) = \sum_{i=1}^{h} b(x_{i-1}, x_i)$$

Fie j indicele maxim astfel încât x_j a fost eliminat din W înaintea lui z. Deoarece $x_{j+1} \in W,\ x_{j+1} \in V^+(x_j)$ rezultă că după eliminarea lui x_j din W avem $d(x_{j+1}) \leq d(x_j) + b(x_j, x_{j+1})$. Această inegalitate se păstrează şi când este eliminat nodul z din W, deoarece $d(x_j)$ rămâne nemodificată ($x_j \notin W$) şi $d(x_{j+1})$ poate numai să descrească. De asemenea, din ipoteza inducţiei avem $d(x_j) = d(s, x_j)$. Astfel

$$d(x_{j+1}) \leq d(x_j) + b(x_j, x_{j+1}) = d(s, x_j) + b(x_j, x_{j+1}) = d(s, x_{j+1}) \leq d(s, z) \tag{4.2}$$

Pot exista următoarele două cazuri:
(c_1) $j = k-1$. În acest caz $x_{j+1} = x_k = z$. Din relaţia (4.2) rezultă $d(z) \leq d(s, z)$;
(c_2) $j < k-1$. În acest caz $x_{j+1} \neq x_k = z$. Dacă $d(s, z) < d(z)$, atunci din relaţia (4.2) rezultă că $d(x_{j+1}) < d(z)$. Conform liniei (10) din algoritm deducem că x_{j+1} este eliminat din W înaintea lui z. Aceasta contrazice modul de alegere a indicelui j. Astfel şi în acest caz avem $d(z) \leq d(s, z)$.

Am demonstrat că $d(y) = d(s, y)$ pentru fiecare $y \in N,\ y \neq s$. Orice drum minim D_{syp} de la nodul s la nodul y se determină cu elementele tabloului predecesor p. ■

Teorema 4.4. *Algoritmul Dijkstra are complexitatea* $O(n^2)$.

Demonstraţie. Pentru selectarea nodului $x \in W$ astfel încât $d(x)$ este minimă sunt necesare cel mult $n-1$ comparaţii. Ciclul FOR se execută de cel mult $n-1$ ori. Ciclul WHILE se execută de n ori. Deci algoritmul are complexitatea $O(n^2)$.■

Observaţia 4.2. Algoritmul Dijkstra poate să conducă la rezultate eronate dacă reţeaua $G = (N, A, b)$ conţine şi arce cu valoare negativă, chiar dacă nu conţine circuite cu valoare negativă. În acest caz estimarea din relaţia (4.2) nu mai este valabilă întotdeauna. De exemplu, dacă se aplică algoritmul Dijkstra reţelei reprezentate în figura 4.2 se determină drumul minim $D_{13p} = (1, 3)$ cu $b(D_{13p}) = 1$. **Fig. 4.2.**

În realitate $D_{13p} = (1, 2, 3)$ cu $b(D_{13p}) = 0$.

Algoritmul lui Dijkstra poate fi implementat în mai multe moduri în funcţie de structurile de date utilizate. Astfel, fiecare implementare poate avea o altă complexitate. Aceste aspecte vor fi prezentate în ultimul paragraf al acestui capitol.

Exemplul 4.2. Se consideră reţeaua reprezentată în figura 4.3. Să se aplice algoritmul Dijkstra acestei reţele. **Fig. 4.3.**

Iniţializări: $W = \{1, 2, 3, 4, 5, 6, 7, 8\},\ d = (0, \infty, \infty, \infty, \infty, \infty, \infty, \infty)$,
$p = (0, 0, 0, 0, 0, 0, 0, 0)$;

Iteraţia 1: $x = 1, W = \{2,3,4,5,6,7,8\},\ d = (0,28,1,2,\infty,\infty,\infty,\infty)$,
$p = (0,1,1,1,0,0,0,0)$;
Iteraţia 2: $x = 3, W = \{2,4,5,6,7,8\},\ d = (0,9,1,2,\infty,\infty,27,\infty)$,
$p = (0,3,1,1,0,0,3,0)$;
Iteraţia 3: $x = 4, W = \{2,5,6,7,8\},\ d = (0,9,1,2,\infty,\infty,26,29)$,
$p = (0,3,1,1,0,0,4,4)$;
Iteraţia 4: $x = 2, W = \{5,6,7,8\},\ d = (0,9,1,2,18,\infty,19,29)$,
$p = (0,3,1,1,2,0,2,4)$;
Iteraţia 5: $x = 5, W = \{6,7,8\},\ d = (0,9,1,2,18,26,19,25)$,
$p = (0,3,1,1,2,5,2,5)$;
Iteraţia 6: $x = 7, W = \{6,8\},\ d = (0,9,1,2,18,26,19,20)$,
$p = (0,3,1,1,2,5,2,7)$;
Iteraţia 7: $x = 8, W = \{6\},\ d = (0,9,1,2,18,26,19,20)$,
$p = (0,3,1,1,2,5,2,7)$;
Iteraţia 8: $x = 6, W = \emptyset, d = (0,9,1,2,18,26,19,20)$,
$p = (0,3,1,1,2,5,2,7)$.

Să determinăm D_{18p}. Avem $p(8) = 7, p(7) = 2, p(2) = 3, p(3) = 1$ şi $D_{18p} = (1,3,2,7,8)$ cu $b(D_{18p}) = d(1,8) = d(8) = 20$.

Arborescenţa parţială $G' = (N, A')$ este reprezentată în figura 4.4.

Fig. 4.4.

4.3.2 Algoritmul Bellman-Ford

Fie reţeaua orientată $G = (N, A, b)$ cu $b : A \to \Re$ şi G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$. În acest caz ecuaţiile lui Bellman (4.1.) pot fi rezolvate cu algoritmul Bellman-Ford. Lista nodurilor adiacente către interior nodului y este $V^-(y) = \{x \mid (x,y) \in A\}$. Algoritmul Bellman-Ford este următorul:

```
(1)   PROGRAM BELLMAN-FORD;
(2)   BEGIN
(3)     d(s) := 0; p(s) := 0;
(4)     FOR y ∈ N − {s} DO
(5)     BEGIN
(6)       d(y) := ∞; p(y) := 0;
(7)     END;
(8)     REPEAT
(9)       FOR y ∈ N DO
(10)        d'(y) := d(y);
(11)      FOR y ∈ N DO
```

(12) IF $V^-(y) \neq \emptyset$
(13) THEN BEGIN
(14) se selectează $x \in V^-(y)$ astfel încât $d'(x) + b(x,y)$ este minimă;
(15) IF $d'(x) + b(x,y) < d'(y)$
(16) THEN BEGIN $d(y) := d'(x) + b(x,y)$; $p(y) := x$; END;
(17) END;
(18) UNTIL $d(y) = d'(y)$ pentru toate nodurile $y \in N$;
(19) END.

Teorema 4.5. *Algoritmul Bellman-Ford determină distanţele $d(s,y)$ şi drumurile minime D_{syp}, $y \in N$, $y \neq s$, în raport cu nodul sursă s din reţeaua orientată $G = (N, A, b)$ cu $b : A \to \Re$ şi G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$.*

Demonstraţie. Mai întâi precizăm că liniile de la (12) la (17) sunt echivalente, referitor la valorile $d(y)$ cu

$$d(y) = \min\{d'(y), \min\{d'(x) + b(x,y) \mid x \in V^-(y)\}\}.$$

Dacă notăm cu $d_0(y)$ valorile definite în liniile (3), (6) şi prin $d_{k+1}(y)$ valorile calculate în timpul iteraţiei $k+1$ a ciclului REPEAT, atunci conform celor precizate mai sus avem

$$d_{k+1}(y) = \min\{d_k(y), \min\{d_k(x) + b(x,y) \mid x \in V^-(y)\}\}.$$

Prin $\ell(D_{syi})$ notăm numărul de arce ale drumului D_{syi} de la nodul s la nodul y şi prin $\mathcal{D}_{sy}$ mulţimea acestor drumuri. Vom arăta prin inducţie după k faptul că

$$d_k(y) = \min\{b(D_{syi}) \mid D_{syi} \in \mathcal{D}_{sy},\ \ \ell D_{syi} \leq k\}.$$

Pentru $k = 0$ afirmaţia este evident adevărată. Presupunem că afirmaţia este adevărată pentru k. Să arătăm că

$$d_{k+1}(y) = \min\{b\left(D'_{syi}\right) \mid D'_{syi} \in \mathcal{D}_{sy},\ \ \ell D_{syi} \leq k+1\}.$$

Avem $\min\{b\left(D'_{syi}\right) \mid D'_{syi} \in \mathcal{D}_{sy},\ \ \ell D_{syi} \leq k+1\} =$
$= \min\{\min\{b\left(D'_{syi}\right) \mid D'_{syi} \in \mathcal{D}_{sy},\ \ \ell D_{syi} \leq k\},$
$\min\{b\left(D'_{syi}\right) \mid D'_{syi} \in \mathcal{D}_{sy},\ \ \ell D_{syi} = k+1\}\} =$
$= \min\{d_k(y), \min\{d_k(x) + b(x,y) \mid x \in V^-(y)\}\} = d_{k+1}(y).$

Dacă G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$, atunci un drum minim de

la s la y poate să conţină cel mult $n-1$ arce. Deci condiţia din linia (18) este satisfăcută după cel mult n iteraţii ale ciclului REPEAT şi $d(y) = d(s,y)$ pentru toţi $y \in N,\ y \neq s$. Un drum minim D_{syp} de la nodul s la nodul y se determină cu elementele tabloului predecesor p.■

Teorema 4.6. *Algoritmul Bellman-Ford are complexitatea $O(mn)$.*

Demonstraţie. În Teorema 4.5 am arătat că ciclul REPEAT se execută de cel mult n ori. O iteraţie a ciclului REPEAT are complexitatea $O(m)$. Deci algoritmul are complexitatea $O(mn)$.■

Observaţia 4.3. Dacă eliminăm liniile (9),(10),(18), înlocuim în linia (8) REPEAT prin FOR $k = 1$ TO n DO şi valorile $d'(x), d'(y)$ prin $d(x)$ respectiv $d(y)$, atunci se obţine posibilitatea testării existenţei unui circuit de valoare negativă în reţeaua $G = (N, A, b)$. Într-adevăr, dacă după execuţia algoritmului există un arc (x,y) astfel încât $d(x) + b(x,y) < d(y)$, atunci înseamnă că $d(x)$ descreşte nelimitat, lucru posibil numai dacă un drum D_{sxi} conţine un circuit $\overset{\circ}{D}$ cu $b(\overset{\circ}{D}) < 0$.

Exemplul 4.3. Se consideră reţeaua reprezentată în figura 4.5. Avem $V^-(1) =$

Fig. 4.5.

$\emptyset,\ V^-(2) = \{1,3\},\ V^-(3) = \{1,5\},\ V^-(4) = \{1\},\ V^-(5) = \{2\},\ V^-(6) = \{5,7\},\ V^-(7) = \{2,3,4\},\ V^-(8) = \{4,5,6,7\}$.

Iniţializări: $d = (0,\infty,\infty,\infty,\infty,\infty,\infty,\infty),\ p = (0,0,0,0,0,0,0,0)$.

Iteraţia 1: $d' = (0,\infty,\infty,\infty,\infty,\infty,\infty,\infty)$;
$y = 1 : d(1) = 0,\ p(1) = 0$;
$y = 2 : x = 1,\ d(2) = 28,\ p(2) = 1$;
$y = 3 : x = 1,\ d(3) = 1,\ p(3) = 1$;
$y = 4 : x = 1,\ d(4) = 2,\ p(4) = 1$;
$y = 5 : x = 2,\ d(5) = \infty,\ p(5) = 0$;
$y = 6 : x = 5,\ d(6) = \infty,\ p(6) = 0$;
$y = 7 : x = 2,\ d(7) = \infty,\ p(7) = 0$;
$y = 8 : x = 4,\ d(8) = \infty,\ p(8) = 0$;

S-a obţinut $d = (0,28,1,2,\infty,\infty,\infty,\infty),\ p = (0,1,1,1,0,0,0,0)$.

Iteraţia 2: $d' = (0,28,1,2,\infty,\infty,\infty,\infty)$,
$d = (0,9,1,2,37,\infty,26,29),\ p = (0,3,1,1,2,0,4,4)$.

Iteraţia 3: $d' = (0,9,1,2,37,\infty,26,29)$,
$d = (0,9,1,2,28,34,19,27),\ p = (0,3,1,1,2,7,2,7)$.

Iteraţia 4: $d' = (0,9,1,2,18,34,19,27)$,
$d = (0,9,1,2,18,26,19,20),\ p = (0,3,1,1,2,5,2,7)$.

Iteraţia 5: $d' = (0,9,1,2,18,26,19,20)$,
$d = (0,9,1,2,18,26,19,20),\ p = (0,3,1,1,2,5,2,7)$.

4.3.3 Algoritmul Floyd-Warshall

Fie reţeaua orientată $G = (N, A, b)$. Se pune problema rezolvării PDM3, adică determinarea distanţei $d(i, j)$ şi a unui drum minim D_{ijp} între oricare două noduri $i, j \in N, i \neq j$. Dacă $b : A \to \Re^+$, atunci PDM3 se poate rezolva iterând după $s \in N$ algoritmul Dijkstra şi se obţine un algoritm cu complexitatea $O\left(n^3\right)$. Dacă $b : A \to \Re$ şi G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$, atunci PDM3 se poate rezolva iterând după $s \in N$ algoritmul Bellman-Ford şi se obţine un algoritm cu complexitatea $O\left(mn^2\right)$. Algoritmul Floyd-Warshall rezolvă PDM3 în cazul $b : A \to \Re$ şi G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$. Acest algoritm are complexitatea $O\left(n^3\right)$. Evident că algoritmul Floyd-Warshall este mai avantajos în rezolvarea PDM3 decât iterarea după nodul s a algoritmului Dijkstra sau a algoritmului Bellman-Ford.

Considerăm reţeaua orientată $G = (N, A, b)$ reprezentată prin matricea valoare adiacentă $B = (b_{ij}), i, j \in N$ cu

$$b_{ij} = \begin{cases} b(i,j) & \text{dacă} \quad i \neq j \text{ şi } (i,j) \in A; \\ 0 & \text{dacă} \quad i = j; \\ \infty & \text{dacă} \quad i \neq j \text{ şi } (i,j) \notin A. \end{cases}$$

Algoritmul Floyd-Warshall determină matricea distanţelor $D = (d_{ij}),\ i, j \in N$ şi matricea predecesor $P = (p_{ij}),\ i, j \in N$.

```
(1)   PROGRAM FLOYD-WARSHALL;
(2)   BEGIN
(3)     FOR i := 1 TO n DO
(4)       FOR j := 1 TO n DO
(5)       BEGIN
(6)         d_ij := b_ij;
(7)         IF i ≠ j AND d_ij < ∞
(8)           THEN p_ij := i
(9)           ELSE p_ij := 0
(10)      END;
(11)    FOR k := 1 TO n DO
(12)      FOR i := 1 TO n DO
(13)        FOR j := 1 TO n DO
(14)          IF d_ik + d_kj < d_ij
(15)            THEN BEGIN d_ij := d_ik + d_kj; p_ij := p_kj; END;
(16)  END.
```

Un drum minim D_{ijp} de la nodul i la nodul j se determină cu algoritmul următor:

```
(1)  PROGRAM DRUM;
(2)  BEGIN
(3)     k := n; x_k := j;
(4)     WHILE x_k ≠ i DO
(5)     BEGIN
(6)        x_{k-1} := p_{i x_k};
(7)        k := k - 1;
(8)     END;
(9)  END.
```

Drumul minim este $D_{ijp} = (x_k, x_{k+1}, \ldots, x_{n-1}, x_n) = (i, x_{k+1}, \ldots, x_{n-1}, j)$.

Teorema 4.7. *Algoritmul Floyd-Warshall determină matricea distanţelor D şi matricea predecesor P ale reţelei orientate $G = (N, A, b)$ cu $b : A \to \Re$ şi cu proprietatea că G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$.*

Demonstraţie. Fie D_0, P_0 matricele definite în liniile (3) la (10) şi D_k, P_k matricele calculate în liniile (11) la (15) la iteraţia k. Prin inducţie după k arătăm că $D_k = \left(d_{ij}^k\right)$ este matricea valorilor drumurilor minime de la i la j având nodurile interioare din $\{1, \ldots, k\}$. Pentru $k = 0$ avem $D_0 = B$ şi afirmaţia este evident adevărată. Presupunem afirmaţia adevărată pentru $k-1$. Liniile (14),(15) la iteraţia k sunt echivalente cu $d_{ij}^k = \min\{d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\}$. Din ipoteza inductivă şi principiul optimalităţii lui Bellman rezultă că $D_k = \left(d_{ij}^k\right)$ este matricea valorilor drumurilor minime de la i la j având nodurile interioare din $\{1, \ldots, k\}$. Deoarece G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$ şi în conformitate cu cele precizate mai sus rezultă că D_n este matricea distanţelor. De asemenea, din modul cum se determină p_{ij} rezultă că P_n este matricea predecesor cu ajutorul căreia se determină drumurile minime D_{ijp}. ■

Teorema 4.8. *Algoritmul Floyd-Warshall are complexitatea $O\left(n^3\right)$.*

Demonstraţie. Evident.■

Observaţia 4.4. Dacă se defineşte $b_{ii} = \infty,\ i \in N$, atunci elementul $d_{ii} < \infty$ reprezintă valoarea unui circuit minim ce trece prin i. Dacă $b_{ii} = 0,\ i \in N$ şi se renunţă la ipoteza restrictivă că oricare circuit $\overset{\circ}{D}$ are valoarea $b(\overset{\circ}{D}) \geq 0$, atunci se obţine posibilitatea testării existenţei circuitelor de valoare negativă în reţeaua orientată $G = (N, A, b)$. Într-adevăr, dacă $d_{ii} < 0$ atunci reţeaua conţine un circuit de valoare negativă care trece prin nodul i.

Exemplul 4.4. Se consideră reţeaua reprezentată în figura 4.6.

Fig. 4.6.

$$D_0 = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 8 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & \infty & \infty & 0 & 5 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}, \quad P_0 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 2 & 0 & 2 & 0 & 2 \\ 3 & 3 & 0 & 3 & 3 \\ 4 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 5 & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 6 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}, \quad P_1 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 & 2 \\ 3 & 3 & 0 & 3 & 3 \\ 4 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 5 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 6 & \infty & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 & 2 \\ 2 & 3 & 0 & 3 & 3 \\ 4 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 5 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 2 & 4 & 8 & 3 \\ 2 & 0 & 6 & 10 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}, \quad P_3 = \begin{bmatrix} 0 & 1 & 1 & 3 & 1 \\ 2 & 0 & 1 & 3 & 2 \\ 2 & 3 & 0 & 3 & 3 \\ 4 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 5 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 2 & 4 & 8 & 3 \\ 2 & 0 & 6 & 10 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ 2 & 4 & 6 & 1 & 0 \end{bmatrix}, \quad P_4 = \begin{bmatrix} 0 & 1 & 1 & 3 & 1 \\ 2 & 0 & 1 & 3 & 2 \\ 2 & 3 & 0 & 3 & 3 \\ 4 & 1 & 1 & 0 & 1 \\ 4 & 1 & 1 & 5 & 0 \end{bmatrix}$$

$$D_5 = \begin{bmatrix} 0 & 2 & 4 & 4 & 3 \\ 2 & 0 & 6 & 2 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ 2 & 4 & 6 & 1 & 0 \end{bmatrix}, \quad P_5 = \begin{bmatrix} 0 & 1 & 1 & 5 & 1 \\ 2 & 0 & 1 & 5 & 2 \\ 2 & 3 & 0 & 3 & 3 \\ 4 & 1 & 1 & 0 & 1 \\ 4 & 1 & 1 & 5 & 0 \end{bmatrix}$$

Să determinăm drumul minim D_{52p} : $x_5 = 2,\ x_4 = p_{52} = 1,\ x_3 = p_{51} = 4,\ x_2 = p_{54} = 5$, deci $D_{52p} = (5, 4, 1, 2)$ cu $b(D_{52p}) = b(5, 4) + b(4, 1) + b(1, 2) = 1 + 1 + 2 = 4 = d(5, 2)$.

4.4 Aplicaţii şi comentarii bibliografice

4.4.1 Reţele de comunicaţii

O reţea $G = (N, A, b)$ poate reprezenta o reţea de comunicaţie cu nodurile N şi rutele directe între noduri formând mulţimea arcelor A. Dacă $b(a)$ reprezintă lungimea arcului a, atunci PDM1, PDM2, PDM3 definite în paragraful 4.1 reprezintă probleme naturale care se pun în astfel de reţele: determinarea drumului/drumurilor cel/celor mai scurt/scurte. Un exemplu concret a fost prezentat în paragraful 1.6.

O problemă specială constă în determinarea drumului cel mai sigur de la nodul s la nodul t. Dacă $p(i, j)$ este o *probabilitate* de funcţionare a arcului $(i, j) \in A$ atunci, presupunând că arcele funcţionează independent unele de altele, probabilitatea de funcţionare a drumului D este: $p(D) = \prod_D p(i, j)$. Considerând $b(i, j) = -logp(i, j)$, problema drumului minim de la s la t semnifică determinarea drumului cel mai sigur de la s la t.

4.4.2 Problema rucsacului

Problema rucsacului este un model clasic în literatura cercetărilor operaţionale.

Un excursionist trebuie să decidă ce obiecte să includă în rucsacul său în vederea unei călătorii. El are de ales între p obiecte, obiectul i are greutatea g_i (în kilograme) şi o utilitate u_i adusă de introducerea obiectului i în rucsac. Obiectivul excursionistului este să maximizeze utilitatea călătoriei astfel încât greutatea obiectelor introduse în rucsac să nu depăşească g kilograme. Această problemă a rucsacului are următoarea formulare ca problemă de programare în numere întregi:

$$\max \sum_{i=1}^{p} u_i x_i$$

$$\sum_{i=1}^{p} g_i x_i \leq g$$

$$x_i \in \{0, 1\} \text{ pentru } i = 1, \ldots, p$$

Această problemă se poate rezolva utilizând metode ale programării dinamice. În continuare vom formula problema rucsacului ca o problemă de drum optim într-o reţea. Această aplicaţie pune în evidenţă legătura dintre modelele de programare dinamică discretă şi problemele de drum optim într-o reţea.

Asociem problemei rucsacului o problemă de drum optim într-o reţea $G = (N, A, b)$ care se defineşte în modul următor. Mulţimea nodurilor este $N = N_0 \cup N_1 \cup \ldots \cup N_p \cup N_{p+1}$, unde $N_0 = \{s\},\ N_i = \{i(0), \ldots, i(g)\},\ i = 1, \ldots, p,\ N_{p+1} = \{t\}$. Submulţimile de noduri $N_0, N_1, \ldots, N_p, N_{p+1}$ reprezintă straturi ale mulţimii

nodurilor: N_0 stratul corespunzător nodului sursă $s, N_1, \ldots, N_p$ straturile corespunzătoare obiectelor $1, \ldots, p$ şi N_{p+1} stratul corespunzător nodului stoc t. Nodul $i(k)$ are semnificaţia că obiectele $1, \ldots, i$ au consumat k unităţi din capacitatea rucsacului. Nodul $i(k)$ are cel mult două arce incidente către exterior, corespunzătoare următoarelor două decizii:
(1) nu se include obiectul $i+1$ în rucsac;
(2) se include obiectul $i+1$ în rucsac, dacă $k + g_{i+1} \leq g$.
Arcul corespunzător primei decizii este $(i(k), (i+1)(k))$ cu $b\,(i(k), (i+1)(k)) = 0$ şi arcul corespunzător celei de a doua decizii (cu condiţia $k + g_{i+1} \leq g$) este $(i(k), (i+1)(k+g_{i+1}))$ cu $b\,(i(k), (i+1)(k+g_{i+1})) = u_{i+1}$. Nodul sursă s are două arce incidente către exterior: $(s, 1(0))$ cu $b(s, 1(0)) = 0$ şi $(s, 1(g_1))$ cu $b(s, 1(g_1)) = u_1$ corespunzătoare celor două decizii de a nu include sau de a include obiectul 1 în rucsac. Se introduc şi arcele $(p(k), t)$ cu $b(p(k), t) = 0,\ k = 0, \ldots, g$.

Fiecare soluţie admisibilă a problemei rucsacului defineşte un drum de la nodul sursă s la nodul stoc t; soluţia admisibilă şi drumul au aceeaşi utilitate. Invers, fiecare drum de la nodul sursă s la nodul stoc t defineşte o soluţie admisibilă a problemei rucsacului cu aceeaşi utilitate.

Exemplul 4.5. Ilustrăm formularea prezentată mai sus pentru o problemă a rucsacului cu patru obiecte care au greutăţile şi utilităţile indicate în tabelul de mai jos.

i	1	2	3	4
u_i	40	15	20	10
g_i	4	2	3	1

Figura 4.7 arată reţeaua $G = (N, A, b)$ asociată problemei rucsacului presupunând că rucsacul are capacitatea de $g = 6$.

Fig. 4.7.

Drumul $D = (s, 1(0), 2(2), 3(5), 4(5), t)$ implică soluţia care include obiectele 2 şi 3 în rucsac şi exclude obiectele 1 şi 4.

Corespondenţa dintre problema rucsacului şi reţeaua asociată $G = (N, A, b)$ arată că dacă în reţeaua G se determină un drum de utilitate maximă (drum maxim) de la nodul sursă s la nodul stoc t se rezolvă problema rucsacului. Putem transforma problema drumului maxim într-o problemă de drum minim prin definirea costurilor arcelor egale cu valorile negative ale utilităţilor arcelor. Deoarece reţeaua $G = (N, A, b)$ este o reţea secvenţială (arcele sunt de la stratul N_i la stratul $N_{i+1},\ i = 0, 1, \ldots, p,$) ea nu conţine circuite şi deci în G nu există circuite $\overset{\circ}{D}$ cu $b(\overset{\circ}{D}) < 0$. Astfel problema drumului minim în reţeaua G poate fi rezolvată cu algoritmul Bellman-Ford, eventual uşor modificat.

4.4.3 Programarea proiectelor

Pentru executarea unui proiect complex (de exemplu, construcţia unui baraj, a unui centru comercial sau a unui avion), diferitele activităţi trebuie să fie bine coordonate pentru a evita pierderea de timp şi bani. Problemele practice sunt complexe datorită restricţiilor de utilizare concurentă a resurselor (oameni, utilaje etc.) de către diversele activităţi. Ne limităm la cazul simplu unde avem restricţii pe secvenţa cronologică a activităţilor: există unele activităţi care nu pot începe înaintea terminării altor activităţi. Se cere să se determine un plan de organizare a proiectului astfel încât timpul total de execuţie să fie minim. Două metode foarte similare pentru rezolvarea acestei probleme, numite *Critical Path Method (CPM)* şi *Project Evaluation and Review Technique (PERT)* au fost dezvoltate între anii 1956 şi 1958 de două grupuri diferite. CPM a fost introdus de E.I. du Pont de la Nemours & Company pentru programarea proiectelor de construcţii şi PERT a fost introdus de Remington Rand de la U.S. Navy pentru programarea cercetării şi dezvoltării activităţilor din cadrul programului rachetei Polaris. CPM - PERT este bazată pe determinarea drumurilor maxime într-o reţea orientată fără circuite. Vom utiliza o formulare în care activităţile proiectului sunt reprezentate prin noduri; alternativ, se pot reprezenta prin arce.
Asociem proiectului o reţea orientată $G = (N, A, b)$ în modul următor. Mulţimea nodurilor $N = \{1, \ldots, n\}$ este mulţimea activităţilor proiectului. Mulţimea arcelor este $A = \{(i, j) \mid i, j \in N,$ activitatea j urmează imediat după (nu există activităţi intermediare) activitatea $i\}$. Dacă τ_i este timpul de execuţie al activităţii i, atunci valoarea arcului (i, j) este $b(i, j) = \tau_i$. Observăm că G este fără circuite, deoarece altfel activităţile dintr-un circuit niciodată nu pot să înceapă. În acest caz G conţine cel puţin un nod y cu $\rho^-(y) = 0$ şi cel puţin un nod x cu $\rho^+(x) = 0$. Construim reţeaua extinsă $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{b})$ unde $\widetilde{N} = \widetilde{N}_1 \cup \widetilde{N}_2 \cup \widetilde{N}_3, \widetilde{N}_1 = \{s\}, \widetilde{N}_2 = N, \widetilde{N}_3 = \{t\}, \widetilde{A} = \widetilde{A}_1 \cup \widetilde{A}_2 \cup \widetilde{A}_3, \widetilde{A}_1 = \{(s, y) \mid y \in \widetilde{N}_2, \rho^-(y) = 0\}, \widetilde{A}_2 = A, \widetilde{A}_3 = \{(x, t) \mid x \in \widetilde{N}_2, \rho^+(x) = 0\}, \widetilde{b}(s, y) = 0, (s, y) \in \widetilde{A}_1, \widetilde{b}(x, y) = b(x, y), (x, y) \in \widetilde{A}_2, \widetilde{b}(x, t) = \tau_x, (x, t) \in \widetilde{A}_3$. Nodul sursă s este nod rădăcină al reţelei $\widetilde{G}$ şi considerăm $\widetilde{G}$ sortată topologic.

Notăm cu $\widetilde{d}(i)$ timpul cel mai devreme posibil la care poate să înceapă activitatea i. Deoarece toate activităţile predecesoare activităţii i au fost terminate, obţinem următorul sistem de ecuaţii:

$$\widetilde{d}(s) = 0, \widetilde{d}(j) = \max\{\widetilde{d}(i) + \widetilde{b}(i, j) \mid (i, j) \in \widetilde{A}\}.$$

Acest sistem de ecuaţii este similar sistemului ecuaţiilor lui Bellman şi descrie drumurile maxime din $\widetilde{G}$. La fel ca în cazul sistemului ecuaţiilor lui Bellman sistemul de mai sus are soluţie unică şi poate fi rezolvat recursiv deoarece $\widetilde{G}$ este sortată topologic. Timpul minim de execuţie al proiectului este $T = \widetilde{d}(t)$ valoarea maximă a drumului de la s la t. Dacă proiectul este terminat la timpul T, timpul

cel mai târziu $T(i)$ la care putem începe activitatea i este dat recursiv de

$$T(t) = T,\ T(i) = \min\{T(j) - \widetilde{b}(i,j) \mid (i,j) \in \widetilde{A}\}.$$

Astfel, $T(t) - T(i)$ este valoarea drumului maxim de la i la t. Evident, considerăm $T(s) = 0$. Rezerva de timp a activităţii i este $r(i) = T(i) - \widetilde{d}(i)$. Toate activităţile i având rezerva $r(i) = 0$ se numesc *activităţi critice*, deoarece ele trebuie să înceapă la timpul $T(i) = \widetilde{d}(i)$, astfel orice întârziere a lor conduce la întârzierea execuţiei proiectului. Observăm că fiecare drum maxim de la s la t conţine numai activităţi critice; pentru acest motiv fiecare astfel de drum este numit *drum critic*. În general există mai multe drumuri critice.

Exemplul 4.6. Considerăm simplificat construcţia unei case. Lista activităţilor, a timpilor necesari acestor activităţi şi activităţile predecesoare fiecărei activităţi sunt prezentate în tabelul de mai jos.

Nod	Activitate	Timp	Activitate predecesoare
1	Pregătirea şantierului de lucru	3	-
2	Furnizarea materialelor de construcţii	2	-
3	Săparea şanţurilor pentru fundaţie	2	1,2
4	Construirea fundaţiei	2	3
5	Construirea zidurilor	7	4
6	Construirea suporturilor acoperişului	3	5
7	Acoperirea acoperişului	1	6
8	Instalaţii exterioare casei	3	4
9	Tencuire exterioară	2	7,8
10	Punerea ferestrelor	1	7,8
11	Punerea tavanelor (plafoanelor)	3	5
12	Pregătirea grădinii	4	9,10
13	Instalaţii exterioare casei	5	11
14	Izolarea pereţilor	3	10,13
15	Zugrăvirea pereţilor	3	14
16	Mutarea	5	15

Reţeaua orientată $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{b})$ este reprezentată în figura 4.8.

Fig. 4.8.

Utilizând sistemul de ecuaţii de mai sus calculăm consecutiv $\widetilde{d}(s) = 0, \widetilde{d}(1) = 0, \widetilde{d}(2) = 0, \widetilde{d}(3) = 3, \widetilde{d}(4) = 5, \widetilde{d}(5) = 7, \widetilde{d}(8) = 7, \widetilde{d}(6) = 14, \widetilde{d}(11) = 14, \widetilde{d}(13) = 17, \widetilde{d}(7) = 17, \widetilde{d}(9) = 18, \widetilde{d}(10) = 18, \widetilde{d}(12) = 20, \widetilde{d}(14) = 22, \widetilde{d}(15) = 25, \widetilde{d}(16) = 28, \widetilde{d}(t) = 33$. Analog calculăm $T(t) = 33, r(t) = 0; T(16) = 28, r(16) = 0; T(15) = 25, r(15) = 0; T(12) = 29, r(12) = 9; T(14) = 22, r(14) = 0; T(9) = 27, r(9) =$

$9; T(10) = 21, r(10) = 3; T(7) = 20, r(7) = 3; T(13) = 17, r(13) = 0; T(6) = 17, r(6) = 3; T(11) = 14, r(11) = 0; T(5) = 7, r(5) = 0; T(8) = 18, r(8) = 11; T(4) = 5, r(4) = 0; T(3) = 3, r(3) = 0; T(1) = 0, r(1) = 0; T(2) = 1, r(2) = 1; T(s) = 0, r(s) = 0$. Astfel, activităţile critice sunt s, 1,3,4,5,11,13,14,15,16,t şi ele formează (în această ordine) drumul critic (care este, în acest caz, unic).

4.4.4 Comentarii bibliografice

Literatura de specialitate clasifică algoritmii care rezolvă problemele de drum minim în două clase: algoritmi de stabilizare a etichetelor şi algoritmi de corectare a etichetelor. Algoritmii din ambele clase sunt abordări iterative. Ei atribuie etichete distanţă nodurilor la fiecare pas; etichetele distanţă sunt estimări (margini superioare) ale distanţelor (valorile drumurilor minime). Algoritmii actualizează etichetele distanţă pas cu pas până când devin distanţele dintre noduri. Algoritmii de stabilizare a etichetelor desemnează o etichetă distanţă ca permanentă (eticheta distanţă este egală cu valoarea drumului minim) la fiecare iteraţie. În contrast, algoritmii de corectare a etichetelor consideră toate etichetele distanţă ca temporare până la pasul final, când ele devin permanente. Algoritmii de stabilizare a etichetelor rezolvă PDM1 şi PDM2 în următoarele două cazuri: valorile arcelor sunt arbitrare dacă reţeaua nu conţine circuite şi valorile arcelor sunt nenegative dacă reţeaua conţine circuite. Algoritmii de corectare a etichetelor sunt mai generali şi se aplică tuturor problemelor cu singura restricţie ca reţelele să nu conţină circuite cu valoare negativă, dar sunt mai puţin eficienţi decât algoritmii de stabilizare a etichetelor.

Primul algoritm de stabilizare a etichetelor a fost descoperit de Dijkstra (1959) şi, independent, de alţi cercetători. Implementarea originală a algoritmului Dijkstra are complexitatea $O\left(n^2\right)$, care este optimală pentru reţelele dense ($m = \Omega\left(n^2\right)$). Totuşi, utilizarea ansamblurilor (heaps) ne permite să obţinem îmbunătăţiri pentru reţelele rare. Implementarea algoritmului Dijkstra cu d-ansamblu, $d = \max\{2, [m/n]\}$, are complexitatea $O(m \log_d n)$ şi este dată de Johnson (1977). Implementarea cu ansamblul Fibonacci, dată de Fredman and Tarjan (1987), are complexitatea $O(m + n \log n)$. Cea mai bună margine teoretică cunoscută în prezent este $O(m + (n \log n)/(\log \log n))$ vezi Fredman and Willard (1994); totuşi, algoritmul lor nu prezintă interes practic. Dacă toate valorile arcelor sunt relativ mici (mărginite de o constantă $\overline{b}$), Ahuja, Mehlhorn, Orlin şi Tarjan (1990) au arătat că se poate obţine complexitatea $O\left(m + n\left(\log \overline{b}\right)^{\frac{1}{2}}\right)$. Pentru cazul planar, Frederickson (1987) a dat un algoritm cu complexitatea $O\left(n(\log n)^{\frac{1}{2}}\right)$. Pentru o scurtă dar interesantă discuţie despre abordări algoritmice diferite de interes practic, vezi Bertsekas (1993). Multe informaţii despre aspecte practice pot fi găsite în articolele lui Gallo and Pallottino (1988) şi Hung and Divoky

(1988).

Cercetătorii au studiat algoritmii de corectare a etichetelor mulţi ani. Ford (1956) a conturat primul algoritm de corectare a etichetelor pentru problema drumului minim. Ulterior, mai mulţi cercetători, incluzând pe Moore (1959) şi Ford şi Fulkerson (1962), au studiat proprietăţile algoritmilor de corectare a etichetelor. Algoritmul Bellman (1958) bazat pe principiul programării dinamice pentru problema drumului minim poate fi de asemenea, privit ca un algoritm de corectare a etichetelor. Implementarea FIFO a algoritmului generic de corectare a etichetelor a fost dată de asemenea de Bellman (1958). Cercetătorii au exploatat flexibilitatea algoritmului generic de corectare a etichetelor pentru descrierea algoritmilor care sunt foarte eficenţi în practică. Pape (1980) dă un listing FORTRAN al unui astfel de algoritm. Algoritmul Pape se execută în timp pseudopolinomial. Gallo and Pallattino (1988) descriu o implementare cu două cozi care se execută în timp polinomial. Lucrările autorilor Hung and Divoky (1988), Gallo and Pallattino (1988) şi Glover, Klingman and Phillips (1992), prezintă rezultate computaţionale extinse ale algoritmilor de stabilizare a etichetelor şi corectare a etichetelor. Algoritmul Floyd-Warshall, care a fost publicat de Floyd (1962), este bazat pe algoritmul Warshall (1962) pentru determinarea închiderii tranzitive a grafurilor.

În încheiere, recomandăm cititorului monografiile Ahuja, Magnanti and Orlin (1993) şi Jungnickel (1999).

Capitolul 5

Fluxuri maxime în reţele

5.1 Noţiuni introductive

Fie $G = (N, A)$ un digraf definit prin mulţimea N cu n noduri şi mulţimea A cu m arce. Se definesc funcţiile: capacitate $c : A \to \Re^+$, margine inferioară $l : A \to \Re^+$ şi funcţia valoare $v : N \to \Re$. Capacitatea $c(x, y)$, a arcului (x, y) din A, desemnează cantitatea maximă care poate traversa arcul, marginea inferioară $l(x, y)$, a arcului (x, y) din A, desemnează cantitatea minimă care poate traversa arcul şi numărul $v(x)$ desemnează valoarea nodului x. Problema fluxului în reţeaua $G = (N, A, l, c)$ constă în a determina funcţia flux $f : A \to \Re^+$ care verifică constrângerile:

$$\sum_y f(x, y) - \sum_y f(y, x) = v(x),\ x \in N \tag{5.1.a}$$

$$l(x, y) \le f(x, y) \le c(x, y),\ (x, y) \in A, \tag{5.1.b}$$

unde $\sum_N v(x) = 0$.

În formă matriceală, problema fluxului în reţeaua $G = (N, A, l, c)$ se reprezintă în modul următor:

$$\overline{M} f = v \tag{5.2.a}$$

$$l \le f \le c \tag{5.2.b}$$

cu f, v, ℓ, c vectori coloană. În această formulare, $\overline{M}$ este matricea de incidenţă nod-arc care este o matrice $n \times m$. Fiecare coloană $\overline{M}_{xy}$ din matrice corespunde arcului (x, y). Coloana $\overline{M}_{xy}$ are un $+1$ în linia corespunzătoare nodului x, un -1 în linia corespunzătoare nodului y şi restul elementelor sunt zero.

Constrângerile (5.1.a) se numesc *constrângerile de conservare a fluxului*. Primul termen al constrângerii corespunzătoare nodului x reprezintă fluxul total

de ieşire din nodul x şi termenul al doilea reprezintă fluxul total de intrare în nodul x; dacă $v(x) > 0$, atunci nodul x este un *nod sursă*; dacă $v(x) < 0$, atunci nodul x este *nod stoc*; dacă $v(x) = 0$, atunci nodul x este un *nod de transfer*. Constrângerile (5.1.b) se numesc *constrângerile de mărginire a fluxului*.

În problema (5.1) înlocuim variabila $f(x, y)$ prin $f'(x, y) + l(x, y)$ şi obţinem problema:

$$\sum_y f'(x, y) - \sum_y f'(y, x) = v'(x),\ \forall x \in N \tag{5.3.a}$$

$$0 \le f'(x, y) \le c'(x, y),\ \forall (x, y) \in A \tag{5.3.b}$$

unde

$$\sum_N v'(x) = 0,\ v'(x) = v(x) - \sum_y l(x, y) + \sum_y l(y, x),\ c'(x, y) = c(x, y) - l(x, y).$$

Deci problema fluxului cu margini inferioare pozitive se poate transforma într-o problemă cu margini inferioare zero. De aceea, în continuare considerăm $l(x, y) = 0,\ (x, y) \in A$ fără a restrânge generalitatea, altfel spus considerăm problema fluxului în reţeaua $G = (N, A, c)$. Cazul cu $l(x, y) \ge 0, (x, y) \in A$ este studiat în paragraful 6.

În mulţimea fluxurilor în reţeaua $G = (N, A, c)$ se introduc relaţia de ordine parţială şi operaţia de adunare (scădere). Presupunem că mulţimea arcelor $A = \{a_1, \ldots, a_m\}$ este ordonată după o anumită ordine.

Definiţia 5.1. Fie $f_1 = (f_1(a_1), \ldots, f_1(a_m))$ şi $f_2 = (f_2(a_1), \ldots, f_2(a_m))$ două fluxuri în reţeaua $G = (N, A, c)$. Se spune că fluxul f_1 este *mai mic* decât fluxul f_2 şi scriem $f_1 < f_2$ dacă $f_1(a_i) \le f_2(a_i)$ pentru $i = 1, \ldots, m$ şi există cel puţin un arc a_k astfel încât $f_1(a_k) < f_2(a_k)$. Se spune că $f_3 = (f_3(a_1), \ldots, f_3(a_m))$ este *suma* fluxurilor f_1 şi f_2 dacă $f_3(a_i) = f_1(a_i) + f_2(a_i)$ pentru $i = 1, \ldots, m$. Analog se defineşte $f_4 = f_2 - f_1$.

Cele mai simple fluxuri nenule în reţeaua $G = (N, A, c)$ sunt fluxul $g(D) = (g(a_1), \ldots, g(a_m))$ de-a lungul unui drum elementar D de la un nod sursă s la un nod stoc t şi fluxul $g(\overset{\circ}{D}) = \left(\overset{\circ}{g}(a_1), \ldots, \overset{\circ}{g}(a_m)\right)$ de-a lungul unui circuit elementar $\overset{\circ}{D}$. Aceste fluxuri se definesc în modul următor:

$$g(a_i) = \begin{cases} r & \text{dacă} \quad a_i \in D, \\ 0 & \text{dacă} \quad a_i \notin D, \end{cases} \quad \overset{\circ}{g}(a_i) = \begin{cases} \overset{\circ}{r} & \text{dacă} \quad a_i \in \overset{\circ}{D}, \\ 0 & \text{dacă} \quad a_i \notin \overset{\circ}{D}. \end{cases}$$

Definiţia 5.2. Un flux f în reţeaua $G = (N, A, c)$ se numeşte *flux elementar* dacă $f = g(D)$ sau $f = g(\overset{\circ}{D})$.

Fie $\mathcal{D}$ mulţimea drumurilor de la nodurile sursă la nodurile stoc şi $\overset{\circ}{\mathcal{D}}$ mulţimea circuitelor în reţeaua $G = (N, A, c)$. Un flux în reţeaua $G = (N, A, c)$ poate fi

definit în două moduri: putem defini fluxurile pe arce şi putem defini fluxurile elementare pe drumuri şi circuite. În formularea flux arc variabilele sunt fluxurile $f(x,y)$ care verifică constrângerile (5.1). În formularea drum - circuit variabilele sunt fluxurile elementare $g(D), D \in \mathcal{D}$ şi $g(\overset{\circ}{D}), \overset{\circ}{D} \in \overset{\circ}{\mathcal{D}}$.

Orice flux nenul în formularea flux arc se poate descompune în fluxuri elementare din formularea flux drum - circuit, aşa cum arată următoarea teoremă.

Teorema 5.1. (Teorema descompunerii fluxului). *Fiecare flux drum-circuit poate fi reprezentat unic ca un flux arc. Invers, fiecare flux arc poate fi reprezentat (nu necesar unic) ca un flux drum-circuit cu următoarele două proprietăţi:*
(a) fiecare drum cu flux nenul conectează un nod sursă la un nod stoc;
(b) cel mult $n+m$ drumuri şi circuite au flux nenul; dintre acestea cel mult m circuite au flux nenul.

Demonstraţie. Pentru afirmaţia directă definim:

$$\delta_{xy}(D) = \begin{cases} 1 & \text{dacă} \quad (x,y) \in D, \\ 0 & \text{dacă} \quad (x,y) \notin D, \end{cases} \quad \delta_{xy}(\overset{\circ}{D}) = \begin{cases} 1 & \text{dacă} \quad (x,y) \in \overset{\circ}{D}, \\ 0 & \text{dacă} \quad (x,y) \notin \overset{\circ}{D} . \end{cases}$$

Pentru fiecare arc $(x,y) \in A$ definim:

$$f(x,y) = \sum_{\mathcal{D}} \delta_{xy}(D) r(D) + \sum_{\overset{\circ}{\mathcal{D}}} \delta_{xy}(\overset{\circ}{D}) \overset{\circ}{r} (\overset{\circ}{D}).$$

Astfel fiecare flux drum-circuit determină fluxul arc unic.

Pentru afirmaţia inversă dăm o demonstraţie algoritmică pentru a arăta cum descompunem un flux arc f în fluxuri elementare. Algoritmul este următorul:

```
(1)   PROGRAM DESCOMPUNERE-FLUX;
(2)   BEGIN
(3)      k := 1; f_1 := f;
(4)      WHILE f_k > 0 DO
(5)      BEGIN
(6)         IF există nod sursă
(7)            THEN BEGIN
(8)                    se selectează un nod sursă x_0; D_k := {x_0}; i := 0;
(9)                 END
(10)           ELSE BEGIN
(11)                   se determină un arc (x_0, x_1) ∈ A cu f_k(x_0, x_1) > 0;
                       D_k := {x_0, x_1}; i := 1;
(12)                END;
(13)        WHILE x_i nu este nod stoc DO
(14)        BEGIN
(15)           se determină un arc (x_i, x_{i+1}) ∈ A cu f_k(x_i, x_{i+1}) > 0;
```

```
                i := i + 1;
(16)            IF x_i ∉ D_k
(17)               THEN D_k := D_k ∪ {x_i}
(18)               ELSE EXIT
(19)         END;
(20)         IF x_i este nod stoc
(21)            THEN BEGIN
(22)                    r := min{v(x_0), −v(x_i), min{f_k(x,y)|(x,y) ∈ D_k}};
(23)                    v(x_0) := v(x_0) − r; v(x_i) := v(x_i) + r;
(24)                    FOR (x,y) ∈ A DO
(25)                       IF (x,y) ∈ D_k
(26)                          THEN g_k(x,y) := r
(27)                          ELSE g_k(x,y) := 0;
(28)                    f_{k+1} := f_k − g(D_k);
(29)                 END;
(30)            ELSE BEGIN
(31)                    D°_k := {x_i, ..., x_i}; r° := min{f_k(x,y) | (x,y) ∈ D°_k};
(32)                    FOR (x,y) ∈ A DO
(33)                    IF (x,y) ∈ D°_k
(34)                          THEN g°_k (x,y) := r°
(35)                          ELSE g°_k (x,y) := 0;
(36)                    f_{k+1} := f_k − g(D°_k);
(37)                 END;
(38)         k := k + 1;
(39)      END;
(40)   END.
```

Iniţial algoritmul porneşte cu $f_1 := f$. Cât timp $f_k > 0$ şi există nod sursă atunci algoritmul determină fie un flux drum $g(D_k)$, fie un flux circuit $g(\overset{\circ}{D}_k)$. Cât timp $f_k > 0$ şi nu există nod sursă atunci algoritmul determină un flux circuit $g(\overset{\circ}{D}_k)$. Se observă că de fiecare dată când se determină un flux drum, valoarea $v(x)$ a unui nod x devine zero sau fluxul pe anumite arce devine zero şi de fiecare dată când se determină un flux circuit, reducem fluxul pe anumite arce la zero. Prin urmare după un număr finit de iteraţii nu mai există noduri x cu $v(x) \neq 0$ şi $f_k = 0$. Deci algoritmul este convergent şi este evident că, la terminarea algoritmului, fluxul originar este suma fluxurilor pe drumurile şi circuitele identificate în timpul execuţiei algoritmului. De asemenea, rezultă din cele spuse mai sus că reprezentarea drum - circuit a fluxului dat f conţine cel mult $n+m$ drumuri şi circuite cu flux nenul şi dintre acestea cel mult m circuite

au flux nenul.■

Să considerăm un flux f cu $v(x) = 0$ pentru toate nodurile x din N. Un astfel de flux se numeşte *circulaţie.*

Teorema 5.2. (Teorema descompunerii circulaţiei). *Oricare circulaţie f poate fi reprezentată ca flux circuit de-a lungul a cel mult m circuite cu flux nenul.*

Demonstraţie. Dacă $v(x) = 0$ pentru toate nodurile x din N, atunci algoritmul descompunerii fluxului determină la fiecare iteraţie un flux circuit şi conform Teoremei 5.1, cel mult m circuite au flux nenul.■

Teorema 5.3. (Teorema de complexitate a algoritmului descompunerii fluxului). *Algoritmul descompunerii fluxului are complexitatea $O(m^2)$.*

Demonstraţie. Algoritmul execută cel mult $n+m$ iteraţii.Complexitatea oricărei iteraţii este $O(m)$. Deci algoritmul are complexitatea $O(m^2.)$■

Dacă se utilizează o structură de date adecvată şi se fac unele modificări în algoritm atunci complexitatea algoritmului scade la $O(m \cdot n)$.

Exemplul 5.1. Să ilustrăm algoritmul descompunerii fluxului pentru fluxul din reţeaua reprezentată în figura 5.1**Fig. 5.1.**

Deci $f = (f(1,2), f(1,3), f(2,4), f(3,2), f(3,4), f(4,5), f(4,6), f(5,3), f(5,6)) = (2,0,5,3,4,6,3,7,2)$ şi $v = (v(1), v(2), v(3), v(4), v(5), v(6)) = (2,0,0,0,3,-5)$. Iniţial $f_1 = f$. Dacă algoritmul selectează nodul sursă 1 şi determină drumul $D_1 = (1,2,4,5,6)$ cu $g(D_1) = (2,0,2,0,0,2,0,0,2)$, atunci $f_2 = (0,0,3,3,4,4,3,7,0)$ şi $v(1) = 0, v(6) = -3$. La următoarea iteraţie algoritmul selectează nodul sursă 5 şi determină drumul $D_2 = (5,3,2,4,6)$ cu $g(D_2) = (0,0,3,3,0,0,3,3,0)$, atunci $f_3 = (0,0,0,0,4,4,0,4,0), v(5) = 0, v(6) = 0$. În a treia iteraţie nu mai există noduri sursă ($v(x) = 0$) şi presupunem că algoritmul determină arcul (3,4) cu $f_3(3,4) > 0$ şi circuitul $\overset{\circ}{D}_3 = (3,4,5,3)$ cu $g(\overset{\circ}{D}_3) = (0,0,0,0,4,4,0,4,0)$. Atunci $f_4 = (0,0,0,0,0,0,0,0,0)$ şi algoritmul se termină. Această descompunere nu este unică.

Fie $S = \{x \mid x \in N, v(x) > 0\}$ mulţimea nodurilor sursă, $T = \{x \mid x \in N, v(x) < 0\}$ mulţimea nodurilor stoc şi $R = \{x \mid x \in N, v(x) = 0\}$ mulţimea nodurilor de transfer (intermediare). Valoarea fluxului de la S la T este

$$v = \sum_S v(x) = \sum_T |v(x)|.$$

Dacă mulţimea nodurilor sursă S are un singur nod îl notăm cu s şi dacă mulţimea nodurilor stoc T are un singur nod îl notăm cu t. Evident că $N = S \cup R \cup T$.

Problema fluxului maxim de la nodurile sursă S la nodurile stoc T în reţeaua $G = (N, A, c)$ constă în a determina funcţia flux f care verifică constrângerile:

$$\text{Maximizează } v \tag{5.4.a}$$

$$\sum_y f(x,y) - \sum_y f(y,x) = \begin{cases} v(x), & x \in S, \\ 0, & x \in R \\ v(x) & x \in T, \end{cases} \tag{5.4.b}$$

$$0 \leq f(x,y) \leq c(x,y),\ (x,y) \in A. \tag{5.4.c}$$

Problema fluxului maxim de la nodul sursă s la nodul stoc t constă în a determina funcţia flux f care verifică constrângerile:

$$\text{Maximizează } v \tag{5.5.a}$$

$$\sum_y f(x,y) - \sum_y f(y,x) = \begin{cases} v, & x = s, \\ 0, & x \neq s,t \\ -v & x = t, \end{cases} \tag{5.5.b}$$

$$0 \leq f(x,y) \leq c(x,y),\ (x,y) \in A. \tag{5.5.c}$$

În acest caz am considerat $v(s) = v$ şi $v(t) = -v$.

Se construieşte reţeaua extinsă $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{c}, \widetilde{s}, \widetilde{t})$ plecând de la reţeaua $G = (N, A, c, S, T)$ în modul următor:

$$\widetilde{N} = \widetilde{N}_1 \cup \widetilde{N}_2 \cup \widetilde{N}_3,\ \widetilde{N}_1 = \{\widetilde{s}\},\ \widetilde{N}_2 = N,\ \widetilde{N}_3 = \{\widetilde{t}\},$$

$$\widetilde{A} = \widetilde{A}_1 \cup \widetilde{A}_2 \cup \widetilde{A}_3,\ \widetilde{A}_1 = \{(\widetilde{s}, x) \mid x \in S\},\ \widetilde{A}_2 = A,\ \widetilde{A}_3 = \{(x, \widetilde{t}) \mid x \in T\},$$

$$\widetilde{c}(\widetilde{s}, x) = \infty,\ (\widetilde{s}, x) \in \widetilde{A}_1,\ \widetilde{c}(x,y) = c(x,y),\ (x,y) \in \widetilde{A}_2,\ \widetilde{c}(x, \widetilde{t}) = \infty,\ (x, \widetilde{t}) \in \widetilde{A}_3.$$

Teorema 5.4. (Caracterizarea problemei fluxului într-o reţea cu surse şi stocuri multiple). *Problema fluxului maxim de la S la T în reţeaua $G = (N, A, c, S, T)$ este echivalentă cu problema fluxului maxim de la $\widetilde{s}$ la $\widetilde{t}$ în reţeaua $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{c}, \widetilde{s}, \widetilde{t})$.*

Demonstraţie. Dacă f este un flux maxim cu valoarea v de la S la T în reţeaua G, atunci se poate defini aplicaţia $\widetilde{f} : \widetilde{A} \to \Re$ în modul următor:
$\widetilde{f}(\widetilde{s}, x) = v(x),\ (\widetilde{s}, x) \in \widetilde{A}_1,\ \widetilde{f}(x,y) = f(x,y),\ (x,y) \in \widetilde{A}_2,\ \widetilde{f}(x, \widetilde{t}) = -v(x)$,
$(x, \widetilde{t}) \in \widetilde{A}_3$.
Astfel $\widetilde{f}$ verifică constrângerile

$$\text{Maximizează } \widetilde{v}$$

$$\sum_y \widetilde{f}(x,y) - \sum_y \widetilde{f}(y,x) = \begin{cases} \widetilde{v}, & x = \widetilde{s}, \\ 0, & x \neq \widetilde{s}, \widetilde{t} \\ -\widetilde{v} & x = \widetilde{t}, \end{cases}$$

$$0 \leq \widetilde{f}(x,y) \leq \widetilde{c}(x,y),\ (x,y) \in \widetilde{A}.$$

cu $\widetilde{v} = v$. Rezultă că $\widetilde{f}$ este un flux maxim în reţeaua $\widetilde{G}$.

Reciproc, dacă $\widetilde{f}$ este un flux maxim de valoare $\widetilde{v}$ în reţeaua $\widetilde{G}$, atunci restricţia sa $f : A \to \Re$ este evident un flux maxim cu valoarea $v = \widetilde{v}$ în reţeaua G. ■

Problema determinării unui flux f care verifică (5.4.b) şi (5.4.c) cu valorile $v(x)$ precizate se numeşte *problema ofertei şi cererii.* În acest caz un nod $x \in S$ se numeşte *nod ofertă* şi un nod $x \in T$ se numeşte *nod cerere.*

În cazul problemei ofertei şi cererii se construieşte reţeaua extinsă $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{c}, \widetilde{s}, \widetilde{t})$ plecând de la reţeaua $G = (N, A, c, S, T)$, cu S mulţimea nodurilor ofertă şi T mulţimea nodurilor cerere, ca mai sus, cu deosebirea că funcţia capacitate $\widetilde{c}$ se defineşte în modul următor:

$c(\widetilde{s}, x) = v(x),\ \ (\widetilde{s}, x) \in \widetilde{A}_1,\ \ \widetilde{c}(x,y) = c(x,y),\ (x,y) \in \widetilde{A}_2,\ \ \widetilde{c}(x,\widetilde{t}) = -v(x),$
$(x, \widetilde{t}) \in \widetilde{A}_3.$

Un arc $(x,y) \in A$ cu proprietatea că $f(x,y) = c(x,y)$ se va numi *arc saturat.*

Teorema 5.5. (Caracterizarea problemei ofertă-cerere). *În reţeaua $G = (N, A, c, S, T)$ există un flux care verifică* (5.4 b) *şi* (5.4 c) *dacă şi numai dacă în reţeaua $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{c}, \widetilde{s}, \widetilde{t})$ există un flux maxim care saturează toate arcele din $\widetilde{A}_1$ şi $\widetilde{A}_3$.*

Demonstraţie. Se demonstrează analog ca Teorema 5.4.■

Exemplul 5.2. Se consideră reţeaua $G = (N, A, c, S, T)$ din figura 5.2, unde $S = \{1, 2\},\ R = \{3, 4, 5\},\ T = \{6, 7\}$ şi pe fiecare arc s-a trecut capacitatea.

Fig. 5.2.

Reţeaua extinsă $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{c}, \widetilde{s}, \widetilde{t})$ este reprezentată în figura 5.3, unde $\widetilde{s} = 0,\ \widetilde{t} = 8$. Fluxul maxim, reprezentat de primul număr de pe fiecare arc al reţelei $\widetilde{G}$, saturează arcele din $\widetilde{A}_1 = \{(0,1), (0,2)\}$ şi $\widetilde{A}_3 = \{(6,8), (7,8)\}$. Deci problema ofertă-cerere are soluţie şi este restricţia funcţiei flux $\widetilde{f} : \widetilde{A} \to \Re$ la mulţimea A.

Fig. 5.3.

5.2 Ipoteze asupra problemei fluxului maxim

În continuare definim valoarea $\overline{c}$ prin $\overline{c} = \max\{c(x,y) \mid (x,y \in A)\}$.

Ipoteza 1. *Reţeaua este orientată.*

Unele probleme de flux sunt formulate pe reţele neorientate sau mixte (care conţin arce şi muchii). În acest caz, problema poate fi rezolvată pe o reţea orientată echivalentă în modul următor.

O muchie $[x, y]$ cu capacitatea $c[x, y]$ permite flux de la nodul x la nodul y şi de asemenea de la nodul y la nodul x. Astfel muchia $[x, y]$ are constrângerea de mărginire: $f[x, y] + f[y, x] \leq c[x, y]$ şi considerăm că fluxul este permis numai în unul din cele două sensuri sau $f[x, y] \cdot f[y, x] = 0$. Pentru a transforma cazul neorientat în cazul orientat, se înlocuieşte fiecare muchie $[x, y]$ prin două arce (x, y) şi (y, x) cu $c(x, y) = c(y, x) = c[x, y]$. Fie f un flux în reţeaua originară (neorientată sau mixtă). Definim fluxul f' în reţeaua transformată (orientată) în modul următor:
- dacă fluxul este de la x la y atunci $f'(x, y) = f[x, y]$ şi $f'(y, x) = 0$;
- dacă fluxul este de la y la x atunci $f'(x, y) = 0$ şi $f'(y, x) = f[y, x]$.
Invers, fie f' un flux în reţeaua transformată. Definim fluxul f în reţeaua originară în modul următor:
- dacă $f'(x, y) - f'(y, x) > 0$, atunci $f[x, y] = f'(x, y) - f'(y, x)$ şi $f[y, x] = 0$;
- dacă $f'(x, y) - f'(y, x) = 0$, atunci $f[x, y] = 0$ şi $f[y, x] = 0$;
- dacă $f'(x, y) - f'(y, x) < 0$, atunci $f[x, y] = 0$ şi $f[y, x] = f'(y, x) - f'(x, y)$.
Evident că fluxul f verifică constrângerile de mărginire pe muchii şi condiţia că fluxul este permis numai în unul din cele două sensuri.

Exemplul 5.3. Se consideră reţeaua mixtă $G = (N, A, c)$ reprezentată în figura 5.4, unde $s = 1, t = 6$ şi al doilea număr de pe fiecare arc sau muchie reprezintă capacitatea.

Fig. 5.4.

Fig. 5.5.

Presupunem că fluxul iniţial în reţeaua mixtă G este fluxul nul şi dorim să determinăm un flux maxim. Corespunzător fluxului nul şi capacităţilor arc din reţeaua mixtă G se obţine reţeaua orientată $G' = (N', A', c')$ cu fluxul nul şi capacităţile pe fiecare arc sunt reprezentate de al doilea număr. Reţeaua G' este reprezentată în figura 5.5. Utilizând orice algoritm pentru determinarea unui flux maxim într-o reţea orientată se obţine fluxul maxim f' reprezentat de primul număr de pe fiecare arc al reţelei G'. Fluxul maxim f din reţeaua mixtă G este următorul: $f[2, 4] = 0, f[4, 2] = 0, f[2, 5] = 1, f[5, 2] = 0, f[3, 5] = 4, f[5, 3] = 0$ şi $f(x, y) = f'(x, y)$ pentru toate arcele (x, y) din A. Acest flux este reprezentat de primul număr de pe fiecare arc sau muchie din reţeaua mixtă G.

Ipoteza 2. *Toate capacităţile sunt întregi nenegativi.*

Această ipoteză este cunoscută şi sub numele de ipoteza integralităţii pentru capacităţile de arc. Ipoteza integralităţii este necesară numai pentru algoritmii a căror complexitate depinde de $\overline{c}$. În realitate, acestă ipoteză nu este restrictivă, deoarece toate calculatoarele memorează capacităţile ca numere raţionale şi putem întotdeauna transforma numerele raţionale, prin multiplicarea lor printr-

un număr corespunzător de mare, în numere întregi.

Ipoteza 3. *Reţeaua nu conţine nici un drum de la nodul sursă s la nodul stoc t alcătuit numai din arce cu capacităţi infinite.*

Când orice arc al unui drum de la nodul sursă s la nodul stoc t are capacitate infinită, putem trimite o cantitate infinită de flux de-a lungul acestui drum şi deci valoarea fluxului maxim este nelimitată.

Ipoteza 4. *Dacă arcul $(x, y) \in A$, atunci şi arcul $(y, x) \in A$.*

Această ipoteză nu este restrictivă deoarece dacă $(x, y) \in A$ şi $(y, x) \notin A$, atunci se consideră că arcul $(y, x) \in A$ cu $c(y, x) = 0$.

Ipoteza 5. *Reţeaua nu conţine arce paralele.*

Această ipoteză este necesară pentru convenţia notaţională şi nu micşorează generalitatea, deoarece arcele paralele $a_1 = (x, y), \ldots, a_k = (x, y)$ se pot înlocui cu un singur arc $a = (x, y)$ cu $c(a) = c(a_1) + \ldots + c(a_k)$.

5.3 Fluxuri şi tăieturi

Conceptul de *reţea reziduală* joacă un rol central în dezvoltarea tuturor algoritmilor de flux maxim pe care îi vom prezenta. De aceea, în continuare vom defini noţiunea de reţea reziduală. Dat un flux f, capacitatea reziduală $r(x, y)$ a oricărui arc $(x, y) \in A$ este fluxul adiţional maxim care poate fi trimis de la nodul x la nodul y utilizând arcele (x, y) şi (y, x). Reamintim ipoteza din paragraful 5.2: dacă arcul (x, y) aparţine reţelei atunci şi arcul (y, x) aparţine reţelei. Capacitatea reziduală $r(x, y)$ are două componente:
(1) $c(x, y) - f(x, y)$ care reprezintă capacitatea nefolosită a arcului (x, y);
(2) $f(y, x)$ care reprezintă fluxul pe arcul (y, x) şi pe care îl putem micşora pentru a creşte fluxul de la nodul x la nodul y. În consecinţă $r(x, y) = c(x, y) - f(x, y) + f(y, x)$. Reţeaua reziduală $\widetilde{G}(f)$ este formată din arcele cu capacităţile reziduale pozitive, adică din arcele (x, y) cu $r(x, y) > 0$. În figura 5.6(b) se prezintă reţeaua reziduală $\widetilde{G}(f)$ corespunzătoare fluxului f din reţeaua prezentată în figura 5.6(a).

Fig. 5.6.

Dacă X şi Y sunt două submulţimi ale mulţimii nodurilor N, nu în mod necesar disjuncte, definim mulţimea de arce:

$$(X, Y) = \{(x, y) | (x, y) \in A,\ x \in X, y \in Y\}.$$

Pentru orice funcţie $g : N \times N \to \Re^+$ şi orice funcţie $h : N \to \Re^+$ definim

$$g(X, Y) = \sum_{(X,Y)} g(x, y),\ h(X) = \sum_{X} h(x).$$

Dacă $X = \{x\}$ sau $Y = \{y\}$ atunci notăm $g(X, Y)$ prin $g(x, Y)$, respectiv $g(X, y)$.

Cu aceste notaţii constrângerile de conservare (5.5 b) se scriu:

$$f(x, N) - f(N, x) = \begin{cases} v, & x = s, \\ 0, & x \neq s, t \\ -v & x = t \end{cases} \qquad (5.5\ b')$$

În continuare, în funcţie de context, vom utiliza constrângerile de conservare a fluxului sub forma (5.5 b) sau (5.5 b').

Definiţia 5.3. Fie $G = (N, A, c)$ o reţea şi $X \subset N$, $\overline{X} = N - X$ cu $X, \overline{X}$ nevide. O *tăietură* este mulţimea de arce $[X, \overline{X}] = (X, \overline{X}) \cup (\overline{X}, X)$, unde $(X, \overline{X})$ reprezintă mulţimea arcelor directe şi $(\overline{X}, X)$ reprezintă mulţimea arcelor inverse ale acestei tăieturi.

Notăm că *ordinea* în care mulţimile $X, \overline{X}$ sunt înregistrate în $[X, \overline{X}]$ este importantă, schimbarea ordinei schimbă mulţimea arcelor directe şi mulţimea arcelor inverse între ele. În reţeaua obţinută, după eliminarea mulţimii arcelor directe $(X, \overline{X})$ ale tăieturii $[X, \overline{X}]$, nu există drum de la orice nod $x \in X$ la orice nod $\overline{x} \in \overline{X}$, şi dacă ambele mulţimi de arce, directe $(X, \overline{X})$ şi inverse $(\overline{X}, X)$, sunt eliminate atunci nu există lanţ de la orice nod $x \in X$ la orice nod $\overline{x} \in \overline{X}$. Aceste proprietăţi sunt cunoscute ca proprietatea de *blocare a drumului*, respectiv proprietatea de *blocare a lanţului.* Proprietatea de blocare a lanţului produce deconexarea reţelei.

Definiţia 5.4. O tăietură $[X, \overline{X}]$ cu $s \in X$ şi $t \in \overline{X}$ se numeşte *tăietură* $s - t$. *Capacitatea* $c[X, \overline{X}]$ a tăieturii $s-t$ $[X, \overline{X}]$ este definită a fi $c[X, \overline{X}] = c(X, \overline{X})$. O tăietură $s-t$ a cărei capacitate este minimă printre toate tăieturile $s-t$ se numeşte *tăietură* $s - t$ *minimă.Capacitatea reziduală* $r[X, \overline{X}]$ a tăieturii $s - t$ $[X, \overline{X}]$ este definită a fi $r[X, \overline{X}] = r(X, \overline{X})$, $r(X, \overline{X}) = c(X, \overline{X}) - f(X, \overline{X}) + f(\overline{X}, X)$.

Dacă în reţeaua $G = (N, A, c)$ f este un flux şi $[X, \overline{X}]$ este o tăietură $s - t$, atunci $f[X, \overline{X}] = f(X, \overline{X}) - f(\overline{X}, X)$ este fluxul net peste această tăietură.

Teorema 5.6. (Teorema valorii fluxului). *În reţeaua $G = (N, A, c)$ dacă f este un flux de valoare v şi $[X, \overline{X}]$ o tăietură $s - t$, atunci f verifică relaţiile*

$$v = f[X, \overline{X}] \leq c[X, \overline{X}].$$

Demonstraţie. Deoarece f este un flux în reţeaua G el satisface constrângerile de conservare (5.5. b'). Sumând aceste ecuaţii pentru $x \in X$ se obţine $v = f(X, N) - f(N, X)$. Înlocuind $N = X \cup \overline{X}$ în această relaţie şi dezvoltând rezultă $v = f(X, X \cup \overline{X}) - f(X \cup \overline{X}, X) = f(X, X) + f(X, \overline{X}) - f(X, X) - f(\overline{X}, X) = f(X, \overline{X}) - f(\overline{X}, X) = f[X, \overline{X}]$. De asemenea f satisface constrângerile de mărginire a fluxului (5.5 c) şi se obţine $f(X, \overline{X}) \leq c(X, \overline{X})$, $f(\overline{X}, X) \geq 0$. Rezultă $v = f[X, \overline{X}] = f(X, \overline{X}) - f(\overline{X}, X) \leq c(X, \overline{X}) = c[X, \overline{X}]$.■

Teorema 5.7. (Teorema capacităţii reziduale). *În reţeaua $G = (N, A, c)$ dacă f este un flux de valoare $v, [X, \overline{X}]$ o tăietură $s - t$ şi f' este un flux de valoare v' cu $v' \geq v$, atunci*

$$v' - v \leq r[X, \overline{X}].$$

Demonstraţie. Din Teorema 5.6 rezultă că $v' \leq c(X, \overline{X})$. Scăzând din această relaţie, relaţia $v = f(X, \overline{X}) - f(\overline{X}, X)$ se obţine $v' - v \leq c(X, \overline{X}) - f(X, \overline{X}) + f(\overline{X}, X) = r(X, \overline{X}) = r[X, \overline{X}]$.■

Teorema 5.7 exprimă faptul că într-o reţea $G = (N, A, c)$ dacă f este un flux admisibil de valoare v, atunci valoarea $\Delta v = v' - v$ a fluxului adiţional care poate fi trimis de la nodul sursă s la nodul stoc t este mai mică decât sau egală cu capacitatea reziduală a oricărei tăieturi $s - t$.

5.4 Algoritmi pseudopolinomiali pentru fluxul maxim

5.4.1 Algoritmul generic

Fie f un flux de valoare v, L un lanţ de la nodul s la nodul t, cu L^+ mulţimea arcelor directe şi L^- mulţimea arcelor inverse, în reţeaua $G = (N, A, c)$. Un lanţ L de la nodul sursă s la nodul stoc t se numeşte *lanţ de mărire a fluxului* (LMF) în raport cu fluxul f dacă $f(x, y) < c(x, y)$ pentru $(x, y) \in L^+$ şi $f(y, x) > 0$ pentru $(y, x) \in L^-$.

Un LMF L în raport cu fluxul f în reţeaua originară G corespunde la un *drum $\widetilde{D}$ de mărire a fluxului* (DMF) în reţeaua reziduală $\widetilde{G}(f)$ şi vice versa. Definim *capacitatea reziduală a unui* LMF L în modul următor: $r(L^+) = \min\{c(x, y) - f(x, y) \mid (x, y) \in L^+\}$, $r(L^-) = \min\{f(x, y) \mid (x, y) \in L^-\}, r(L) = \min\{r(L^+), r(L^-)\}$. Evident că $r(L) > 0$. Atunci se poate face *mărirea de flux*: $f'(x, y) = f(x, y) + r(L)$, $(x, y) \in L^+, f'(x, y) = f(x, y) - r(L)$, $(x, y) \in L^-, f'(x, y) = f(x, y)$, $(x, y) \notin L$ şi fluxul f' are valoarea $v' = v + r(L)$.

Cât timp reţeaua reziduală $\widetilde{G}(f)$ conţine un DMF putem trimite flux de la nodul sursă s la nodul stoc t. Algoritmul generic este bazat pe această proprietate.

```
(1)   PROGRAM GENERIC-FD;
(2)   BEGIN
(3)     f := f_0;
(4)     se construieşte G̃(f);
(5)     WHILE G̃(f) conţine un DMF DO
(6)     BEGIN
(7)       se identifică un DMF D̃;
(8)       r(D̃) := min{r(x, y) | (x, y) ∈ D̃};
(9)       se execută mărirea de flux;
(10)      se actualizează G̃(f);
```

(11) END;
(12) END.

În implementarea oricărei versiuni a algoritmului generic avem opţiunea să lucrăm fie pe reţeaua originară cu fluxurile $f(x,y)$, fie pe reţeaua reziduală cu capacităţile reziduale $r(x,y)$ şi când algoritmul se termină să calculăm fluxurile $f(x,y)$. Versiunea prezentată lucrează pe reţeaua reziduală. Să vedem cum putem determina fluxurile $f(x,y)$ la terminarea algoritmului. În general, mai multe combinaţii ale lui $f(x,y)$ şi $f(y,x)$ corespund pentru a verifica relaţia $r(x,y) = c(x,y) - f(x,y) + f(y,x)$ şi constrângerile de mărginire (5.5 c). Pentru o alegere rescriem $r(x,y) = c(x,y) - f(x,y) + f(y,x)$ ca $c(x,y) - r(x,y) = f(x,y) - f(y,x)$. Acum, dacă $c(x,y) - r(x,y) \geq 0$, atunci $f(x,y) = c(x,y) - r(x,y)$ şi $f(y,x) = 0$, altfel stabilim $f(x,y) = 0, f(y,x) = r(x,y) - c(x,y)$. Deci pentru orice arc $(x,y) \in A$ avem $f(x,y) = \max\{0,\ c(x,y) - r(x,y)\}$.

Exemplul 5.4. Ilustrăm algoritmul generic pe reţelele din figura 5.7, în care $s = 1, t = 4$.

Fig. 5.7.

Iniţial $f_0 = 0$. Reţeaua reziduală este reţeaua originară din figura 2.4 (a). Presupunem că algoritmul generic selectează DMF $\widetilde{D} = (1,3,4)$ cu $r(\widetilde{D}) = \min\{r(1,3), r(3,4)\} = \min\{4,5\} = 4$. Mărirea de flux reduce $r(1,3) = 4$ la 0 şi se elimină arcul (1,3) din reţeaua reziduală, creşte $r(3,1)$ de la 0 la 4 şi se adaugă arcul (3,1) la reţeaua reziduală. De asemenea, mărirea de flux descreşte $r(3,4)$ de la 5 la 1, astfel în noua reţea reziduală $r(3,4) = 1$ şi $r(4,3) = 4$.

Figura 5.7 (b) arată reţeaua reziduală după prima iteraţie. În a doua iteraţie, presupunem că algoritmul selectează DMF $\widetilde{D} = (1,2,3,4)$ cu $r(\widetilde{D}) = \min\{2,3,1\} = 1$. Mărirea de flux conduce la reţeaua reziduală arătată în figura 5.7 (c). În a treia iteraţie, algoritmul generic selectează unicul DMF $\widetilde{D} = (1,2,4)$ cu $r(\widetilde{D}) = \min\{1,1\} = 1$. Mărirea de flux conduce la reţeaua reziduală arătată în figura 5.7 (d), care nu mai conţine DMF.

În continuare se calculează fluxurile pe arce:

$$c(1,2) - r(1,2) = 2 - 0 = 2, \text{ deci } f(1,2) = 2,\ f(2,1) = 0,$$
$$c(1,3) - r(1,3) = 4 - 0 = 4, \text{ deci } f(1,3) = 4,\ f(3,1) = 0,$$
$$c(2,3) - r(2,3) = 3 - 2 = 1, \text{ deci } f(2,3) = 1,\ f(3,2) = 0,$$
$$c(2,4) - r(2,4) = 1 - 0 = 1, \text{ deci } f(2,4) = 1,\ f(4,2) = 0,$$
$$c(3,4) - r(3,4) = 5 - 0 = 5, \text{ deci } f(3,4) = 5,\ f(4,3) = 0.$$

Teorema 5.8. (Teorema valorilor întregi) *Dacă capacitatea c şi fluxul iniţial f_0 sunt cu valori întregi atunci algoritmul generic converge într-un număr finit de iteraţii şi la terminarea execuţiei se obţine un flux maxim cu valori întregi.*

Demonstraţie. Dacă c şi f_0 sunt cu valori întregi atunci $r(\widetilde{D})$ a oricărui DMF va fi întotdeauna o valoare întreagă. Deci pentru fiecare DMF valoarea fluxului va creşte cu $r(\widetilde{D}) \geq 1$. Rezultă că după un număr finit de iteraţii valoarea fluxului va ajunge la valoarea maximă şi în reţeaua reziduală nu vor mai exista DMF. Deci algoritmul generic converge într-un număr finit de iteraţii. Deoarece c şi f_0 sunt cu valori întregi este evident că după fiecare mărire de flux, noul flux şi capacitate reziduală sunt cu valori întregi. Deci fluxul maxim este cu valori întregi.■

În cazul când capacitatea c şi fluxul admisibil iniţial f_0 sunt cu valori raţionale algoritmul generic converge de asemenea într-un număr finit de iteraţii, deoarece în acest caz problema fluxului maxim poate fi transformată într-o problemă echivalentă în care capacitatea şi fluxul admisibil iniţial sunt cu valori întregi.

Algoritmul generic este cel mai simplu algoritm pentru rezolvarea problemei fluxului maxim. Empiric, algoritmul generic se prezintă rezonabil de bun. Totuşi, în cazul cel mai defavorabil limita pe numărul de iteraţii nu este în totalitate satisfăcătoare. Edmonds şi Karp (1972) au dat un exemplu patologic pentru algoritmul generic ilustrat pe reţeaua din figura 5.8.

Fig. 5.8.

Iniţial $f_0 = (f_0(1,2), f_0(1,3), f_0(2,3), f_0(2,4), f_0(3,4)) = (0,0,0,0,0)$. Evident că fluxul admisibil maxim în această reţea este $f = (u,u,0,u,u)$ cu valoarea $v = 2u$. Definim secvenţa de fluxuri admisibile $f_0, f_1, f_2, \ldots, f_{2i-2}, f_{2i-1}, f_{2i}, \ldots, f_{2u}$ cu $f_{2i-1} = (i, i-1, 1, i-1, i)$, $f_{2i} = (i,i,0,i,i)$ pentru $i = 1, \ldots, u$. În această secvenţă f_k are valoarea $v_k = k$ pentru $k = 0, \ldots, 2u$. Pentru $i = 1$ la u, f_{2i-1} este obţinut prin mărirea lui f_{2i-2} utilizând DMF $\widetilde{D}_{2i-1} = (1,2,3,4)$ cu $r(\widetilde{D}_{2i-1}) = 1$, din reţeaua reziduală arătată în figura 5.9 (a). Pentru $i = 1$ la u, f_{2i} este obţinut prin mărirea lui f_{2i-1} utilizând DMF $\widetilde{D}_{2i-1} = (1,3,2,4)$ cu $r(\widetilde{D}_{2i-1}) = 1$ din reţeaua arătată în figura 5.9 (b).

Fig. 5.9. (a) Reţeaua reziduală în raport cu fluxul f_{2i-2}
(b) Reţeaua reziduală în raport cu fluxul f_{2i-1}

Această secvenţă a fluxurilor admisibile are $2u+1$ fluxuri. Presupunând că u este un întreg pozitiv, *dimensiunea problemei* fluxului maxim pe reţeaua din figura 5.8 (adică numărul de cifre binare necesare la memorarea datelor) este $\log_2(u+1) + w$, unde w este o constantă. Efortul de calcul cerut prin algoritmul generic la rezolvarea aceastor probleme, dacă el urmează această secvenţă, este de $2u$ iteraţii. Deci algoritmul generic are o complexitate exponenţială în raport cu dimensiunea problemei.

Ford şi Fulkerson (1962) au prezentat un exemplu pentru a ilustra faptul că

dacă capacităţile sunt numere iraţionale şi fluxul admisibil iniţial este cu valori întregi, algoritmul generic poate să nu conveargă într-un număr finit de iteraţii şi la limită poate să conveargă la un flux a cărui valoare este strict mai mică decât valoarea fluxului maxim.

5.4.2 Algoritmul Ford-Fulkerson de etichetare

Algoritmul de etichetare este o versiune îmbunătăţită a algoritmului generic. În algoritmul de etichetare se identifică un DMF $\widetilde{D}$ în reţeaua reziduală $\widetilde{G}(f)$ cu ajutorul unei *operaţii de etichetare* a nodurilor. Dacă un nod x este etichetat atunci se pot eticheta toate nodurile neetichetate y pentru care $(x, y) \in \widetilde{A}$. Această operaţie de etichetare se numeşte *analizarea nodului etichetat* x. Algoritmul de etichetare se datorează lui Ford şi Fulkerson (1956), de aceea se numeşte algoritmul *Ford-Fulkerson de etichetare.* În timpul execuţiei algoritmului de etichetare un nod x poate fi *etichetat* sau *neetichetat.* Un nod x este etichetat dacă algoritmul, prin operaţia de etichetare, a identificat un drum $\widetilde{D}$ de la nodul sursă s la nodul x în reţeaua reziduală, altfel nodul x este neetichetat. Un nod etichetat x este *analizat* dacă sunt etichetate toate nodurile y pentru care $(x, y) \in \widetilde{A}$, altfel nodul x este *neanalizat.* În algoritmul de etichetare utilizăm *lista nodurilor etichetate şi neanalizate* notată $\widetilde{V}$ şi *vectorul predecesor* notat $\widetilde{p}$. Operaţia $\widetilde{p}(y) := x$ înseamnă că nodul y este etichetat plecând de la nodul x.

```
(1)   PROGRAM FFE;
(2)   BEGIN
(3)      f := f_0;
(4)      se construieşte G̃(f);
(5)      p̃(t) := s;
(6)      WHILE p̃(t) ≠ 0 DO
(7)      BEGIN
(8)         FOR y ∈ Ñ DO p̃(y) := 0;
(9)         Ṽ := {s}; p̃(s) := t;
(10)        WHILE Ṽ ≠ ∅ şi p̃(t) = 0 DO
(11)        BEGIN
(12)           se extrage un nod x din Ṽ;
(13)           FOR (x, y) din Ã DO
(14)              IF p̃(y) = 0
(15)                 THEN BEGIN p̃(y) := x; Ṽ := Ṽ ∪ {y}; END;
(16)        END;
(17)        IF p̃(t) ≠ 0
(18)           THEN MĂRIRE
(19)     END;
```

(20) END.

(1) PROCEDURA MĂRIRE;
(2) BEGIN
(3) se determină DMF $\widetilde{D}$ cu vectorul predecesor $\widetilde{p}$;
(4) se determină $r(\widetilde{D}) = \min\{r(x,y) \mid (x,y) \in \widetilde{D}\}$;
(5) se efectuează mărirea de flux de-a lungul lui $\widetilde{D}$;
(6) se actualizează $\widetilde{G}(f)$;
(7) END;

Exemplul 5.5. Să ilustrăm algoritmul de etichetare pe reţeaua din figura 5.10 (a). Fluxul iniţial este $f_0 = (f(1,2), f(1,3), f(2,3), f(2,4), f(3,4)) = (1,0,1,0,1)$. Reţeaua reziduală $\widetilde{G}(f_0)$ este reţeaua arătată în figura 5.10 (b). La prima iteraţie vectorul predecesor $\widetilde{p}$ poate fi $\widetilde{p} = (4,1,1,2)$ sau $\widetilde{p} = (4,1,1,3)$. Presupunem că $\widetilde{p} = (4,1,1,2)$, atunci $\widetilde{D} = (1,2,4)$ cu $r(\widetilde{D}) = u-1$. Reţeaua reziduală actualizată după mărirea de flux este arătată în figura 5.10 (c). În a doua iteraţie se obţine $\widetilde{p} = (4,3,1,3), \widetilde{D} = (1,3,4), r(\widetilde{D}) = u-1$.

Fig. 5.10. Ilustrarea algoritmului de etichetare

Reţeaua reziduală actualizată după mărirea de flux este arătată în figura 5.10 (d). În a treia iteraţie se obţine $\widetilde{p} = (4,3,1,2), \widetilde{D} = (1,3,2,4), r(\widetilde{D}) = 1$. După mărirea de flux se obţine reţeaua reziduală din figura 5.10 (e). În această reţea iniţial $\widetilde{V} = \{s\}$, obţinem $\widetilde{V} = \emptyset, \widetilde{p}(4) = 0$ şi algoritmul se termină cu un flux maxim.

Teorema 5.9. (Teorema fluxului maxim şi a tăieturii minime). *Într-o reţea $G = (N, A, c)$ valoarea fluxului maxim de la nodul sursă s la nodul stoc t este egală cu capacitatea tăieturii $s-t$ minime.*

Demonstraţie. În reţeaua $G = (N, A, c)$, dacă f este un flux admisibil de valoare $v, [X, \overline{X}]$ o tăietură $s-t$, atunci conform Teoremei 5.6. avem $v = f[X, \overline{X}] \leq c[X, \overline{X}]$. Rezultă că, dacă f^* este un flux admisibil maxim de valoare v^* şi $[X^*, \overline{X}^*]$ este o tăietură $s-t$ minimă atunci $v^* = c[X^*, \overline{X}^*]$.■

Teorema 5.10. (Teorema drumului de mărirea fluxului). *În reţeaua $G = (N, A, c)$ un flux f^* este un flux maxim dacă şi numai dacă reţeaua reziduală $\widetilde{G}(f)$ nu conţine drum de mărire a fluxului.*

Demonstraţie. Dacă f^* este un flux maxim atunci $\widetilde{G}(f^*)$ nu conţine DMF, altfel f^* nu ar fi flux maxim. Reciproc, dacă reţeaua reziduală $\widetilde{G}(f^*)$ nu conţine DMF rezultă că nu poate fi etichetat nodul stoc t plecând de la nodul sursă s. Fie X^* mulţimea nodurilor etichetate şi $\overline{X}^* = N - X^*$ mulţimea nodurilor neetichetate. Evident că $[X^*, \overline{X}^*]$ este o tăietură $s-t$. Deoarece algoritmul de

etichetare nu a putut eticheta nodurile din $\overline{X}^*$ de la nodurile din X^* rezultă că $r^*(x,\overline{x}) = 0$ pentru oricare arc $(x,\overline{x}) \in (X^*, \overline{X}^*)$. Dar $r^*(x,\overline{x}) = c(x,\overline{x}) - f^*(x,\overline{x}) + f^*(\overline{x},x),\ c(x,\overline{x}) - f^*(x,\overline{x}) \geq 0, f^*(\overline{x},x) \geq 0$ şi condiţiile $r^*(x,\overline{x}) = 0$ implică faptul că $f^*(x,\overline{x}) = c(x,\overline{x})$ pentru oricare arc $(x,\overline{x}) \in (X^*, \overline{X}^*)$ şi $f^*(\overline{x},x) = 0$ pentru fiecare arc $(\overline{x},x) \in (\overline{X}^*, X^*)$. Rezultă că $v^* = f^*[X^*, \overline{X}^*] = f^*(X^*, \overline{X}^*) - f^*(\overline{X}^*, X^*) = f^*(X^*, \overline{X}^*) = c(X^*, \overline{X}^*) = c[X^*, \overline{X}^*]$, adică fluxul f^* de valoare v^* obţinut la terminarea algoritmului de etichetare este un flux maxim. ■

Teorema 5.11. (Teorema de corectitudine a algoritmului de etichetare.) *În reţeaua $G = (N, A, c)$, algoritmul de etichetare determină un flux maxim de la nodul sursă s la nodul stoc t.*

Demonstraţie. Execuţia algoritmului de etichetare se termină când nodul stoc t nu mai poate fi etichetat, prin operaţia de etichetare, pornind de la nodul sursă s. Deci în reţeaua reziduală $\widetilde{G}(f^*)$ nu există DMF şi conform Teoremei 5.10 f^* este un flux maxim. ■

Demonstraţia acestei teoreme arată că atunci când algoritmul de etichetare se termină, el a determinat şi o tăietură minimă.

Pentru a studia complexitatea, în cazul cel mai defavorabil, a algoritmului de etichetare să analizăm dacă putem utiliza tehnica de construire a secvenţei de fluxuri admisibile $f_0, f_1, f_2, \ldots, f_{2i-2}, f_{2i-1}, f_{2i}, \ldots, f_{2u}$ de la algoritmul generic. Evident, dacă se pleacă cu $f_0 = 0$, atunci $\widetilde{D} = (1,2,4)$ sau $\widetilde{D} = (1,3,4)$ şi $\widetilde{D} = (1,2,3,4)$ nu poate fi obţinut în primele două iteraţii. Deci, cel puţin nu în toate cazurile, putem construi secvenţa de fluxuri admisibile de la algoritmul generic. Totuşi dacă se dă reţeaua originară $G = (N, A, c)$, atunci se poate construi reţeaua transformată $G' = (N', A', c')$, astfel încât comportarea algoritmului generic pe reţeaua reziduală $\widetilde{G}(f)$ este aceeaşi cu comportarea algoritmului de etichetare pe reţeaua reziduală $\widetilde{G}'(f)$. Reţeaua G' se construieşte în modul următor:
$N' = N \cup V,\ V = \{x.y \mid x.y \text{ se află pe arcul} (x,y) \in A\},\ |N'| = n + m,$
$A' = \{(x, x.y),\ (x.y, y) \mid (x,y) \in A\},\ |A'| = 2m,$
$c' : A' \to \Re_+,\ c'(x, x.y) = c'(x.y, y) = c(x,y),\ (x,y) \in A.$

În aplicarea algoritmului de etichetare pe reţeaua $\widetilde{G}'(f)$ următoarele afirmaţii sunt adevărate:
(a_1) Dacă un nod $x \in N$ este analizat, atunci numai noduri $x.y \in V$ pot fi etichetate.
(a_2) Dacă un nod $x.y \in V$ este analizat, atunci numai nodul y poate fi etichetat.

Din aceste afirmaţii rezultă că dacă algoritmul de etichetare se aplică pe $\widetilde{G}'(f)$ şi $\widetilde{D}'$ este un DMF de la s la t, atunci este posibilă analizarea nodurilor din $\widetilde{D}'$ în ordinea în care ele apar în $\widetilde{D}'$. Dacă $\widetilde{D}'$ este dat, atunci operaţia de etichetare se va termina prin determinarea DMF $\widetilde{D}'$. Reţelei reziduale $\widetilde{G}(f)$ din figura 5.10

(b) îi corespunde reţeaua reziduală $\widetilde{G}'(f)$ arătată în figura 5.11.

Fig. 5.11. Reţeaua reziduală $\widetilde{G}'(f)$

Fie DMF $\widetilde{D} = (1, 3, 2, 4)$ în reţeaua $\widetilde{G}(f)$ din figura 5.10 (b). Am văzut că acest DMF nu poate fi obţinut la prima iteraţie a algoritmului de etichetare. Drumul corespunzător în reţeaua $\widetilde{G}'(f)$ din figura 5.11 este $\widetilde{D}' = (1, 1.3, 3, 3.2, 2, 2.4, 4)$. Se poate verifica că nodurile din $\widetilde{D}'$ se pot analiza în ordinea în care apar în $\widetilde{D}'$ şi algoritmul de etichetare determină acest drum.
Rezultă următoarele concluzii:
(c_1) Dacă capacitatea şi fluxul admisibil iniţial sunt cu valori întregi, atunci aplicarea algoritmului de etichetare se termină cu un flux admisibil maxim după un număr finit de iteraţii şi fluxul maxim este cu valori întregi.
(c_2) Dacă capacităţile sunt iraţionale, atunci aplicarea algoritmului de etichetare poate produce o secvenţă infinită de fluxuri admisibile ale căror valori converg la o valoare strict mai mică decât valoarea fluxului maxim.
(c_3) Complexitatea algoritmului de etichetare este aceeaşi cu a algoritmului generic, adică exponenţială în raport cu dimensiunea problemei.

Teorema 5.12. (Teorema de complexitate a algoritmului de etichetare). *Dacă capacitatea şi fluxul admisibil iniţial sunt cu valori întregi atunci algoritmul de etichetare are complexitatea* $O(mn\overline{c})$, $\overline{c} = \max\{c(x, y) \mid (x, y) \in A\}$.

Demonstraţie. Pentru determinarea unui DMF sunt necesare cel mult m analizări de arce. Dacă capacitatea şi fluxul admisibil iniţial sunt cu valori întregi atunci sunt necesare cel mult v^* DMF, unde v^* este valoarea fluxului maxim. Rezultă că algoritmul de etichetare are complexitatea $O(mv^*)$. Dacă $\overline{c}$ este o margine superioară a capacităţilor arcelor atunci $v^* \leq (n-1)\overline{c}$ deoarece $(n-1)\overline{c}$ este o margine superioară a capacităţii tăieturii $s-t$ $[s, \overline{X}]$. Deci algoritmul de etichetare are complexitatea $O(mn\overline{c})$. ■

Deoarece complexitatea algoritmului de etichetare depinde de $\overline{c}$ el are o complexitate pseudopolinomială. Reamintim că algoritmii pseudopolinomiali constituie o subclasă importantă a clasei algoritmilor exponenţiali.

5.5 Aplicaţii ale teoremei fluxului maxim şi tăieturii minime

5.5.1 Conexitatea reţelei

Într-o reţea $G = (N, A, c)$, două drumuri se numesc *drumuri disjuncte arc* dacă ele nu au nici un arc în comun şi *drumuri disjuncte nod* dacă ele nu au nici un nod în comun, eventual cu excepţia nodurilor iniţial şi final.

Teorema 5.13. (Teorema deconexării drumurilor $s-t$ prin arce). *Numărul maxim al drumurilor disjuncte arc de la nodul sursă s la nodul stoc t este egal cu numărul minim al arcelor care eliminate din reţea deconexează toate drumurile de la nodul s la nodul $t((s,t) \notin A)$.*

Demonstraţie. Definim capacitatea $c : A \to \Re_+$ prin $c(x,y) = 1$ pentru fiecare arc $(x,y) \in A$. Fie f^* fluxul maxim de valoare v^*. Teorema descompunerii fluxului (Teorema 5.1) implică faptul că putem descompune fluxul f^* de-a lungul drumurilor de la s la t şi de-a lungul circuitelor. Deoarece fluxurile de-a lungul circuitelor nu afectează valoarea fluxului, suma valorilor fluxurilor de-a lungul drumurilor este v^*. Cum capacitatea fiecărui arc este 1, aceste drumuri care transportă flux sunt disjuncte arc şi fiecare transportă o unitate de flux. Deci numărul maxim de drumuri disjuncte arc de la s la t este v^*. Fie $[X, \overline{X}]$ tăietura $s-t$ minimă. Conform definiţiei capacităţii oricărei tăieturi avem $c[X, \overline{X}] = c(X, \overline{X})$. Deoarece capacitatea fiecărui arc este 1 rezultă $c(X, \overline{X}) = |(X, \overline{X})|$. Fiecare drum de la nodul sursă s la nodul stoc t conţine un arc din $(X, \overline{X})$ şi eliminarea acestor arce deconexează toate drumurile de la s la t. Tăietura $s-t$ $[X, \overline{X}]$ fiind minimă şi $c[X, \overline{X}] = c(X, \overline{X}) = |(X, \overline{X})|$ rezultă că numărul arcelor care deconexează toate drumurile de la s la t este minim. Conform Teoremei fluxului maxim şi tăieturii minime (Teorema 2.4) avem $v^* = c[X, \overline{X}] = c(X, \overline{X}) = |(X, \overline{X})|$ şi rezultă afirmaţia din teoremă. ■

Teorema 5.14. (Teorema deconexării drumurilor $s-t$ prin noduri) *Numărul maxim al drumurilor disjuncte nod de la nodul sursă s la nodul stoc t este egal cu numărul minim al nodurilor care eliminate din reţea deconexează toate drumurile de la nodul s la nodul t $((s,t) \notin A)$.*

Demonstraţie. Definim reţeaua $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{c})$ cu $N_0 = \{s,t\}$, $\widetilde{N} = \{x', x'' \mid x \in N - N_0\} \cup \widetilde{N}_0$, $\widetilde{N}_0 = \{s'', t'\}$, $s'' = s$, $t' = t$, $\widetilde{A} = \widetilde{A}_1 \cup \widetilde{A}_2$, $\widetilde{A}_1 = \{(x'', y') \mid (x,y) \in A\}, \widetilde{A}_2 = \{(x', x'') \mid x \in N - N_0\}, \widetilde{c} : \widetilde{A} \to \Re_+, \widetilde{c}(x'', y') = \infty$, $(x'', y') \in \widetilde{A}_1$, $\widetilde{c}(x', x'') = 1$, $(x', x'') \in \widetilde{A}_2$. Evident că există o corespondenţă biunivocă între mulţimea drumurilor disjuncte nod din reţeaua originară G şi mulţimea drumurilor disjuncte arc din reţeaua transformată $\widetilde{G}$. Deoarece $\widetilde{c}(x'', y') = \infty, (x'', y') \in \widetilde{A}_1$, rezultă că în $\widetilde{G}$ tăietura $s'' - t'$ minimă are ca arce directe numai arce (x', x'') din $\widetilde{A}_2$. Aplicând Teorema 5.13 şi utilizând observaţiile de mai sus rezultă afirmaţia din teoremă. ■

5.5.2 Cuplaje şi acoperiri

Fie $G = (N, A, c), N = N_1 \cup N_2$ o reţea bipartită. O submulţime $A' \subset A$ se numeşte *cuplaj* dacă oricare două arce din A' nu sunt adiacente. O submulţime $N' \subset N$ se numeşte *acoperire nod* dacă fiecare arc din A este incident la unul din nodurile din N'.

Exemplul 5.6. Pentru ilustrarea acestor definiţii se consideră reţeaua bipar-

tită arătată în figura 5.12. Mulţimea $A' = \{(1,1'),(3,3'),(4,5'),(5,2')\}$ este un cuplaj, dar $A'' = \{(1,2'),(3,1'),(3,4')\}$ nu este deoarece arcele $(3,1')$ şi $(3,4')$ sunt adiacente fiind incidente la acelaşi nod 3. Mulţimea $N' = \{1,2',3,5'\}$ este o acoperire nod, dar mulţimea $N'' = \{2',3',4,5\}$ nu este o acoperire nod, deoarece arcele $(1,1'),(3,1')$ şi $(3,4')$ nu sunt incidente la nici un nod din mulţimea N''.

Fig. 5.12. Reţeaua bipartită

Teorema 5.15. (Teorema cuplajului maxim şi a acoperirii minime.) *Într-o reţea bipartită $G = (N, A)$, $N = N_1 \cup N_2$ cardinalitatea maximă a oricărui cuplaj este egală cu cardinalitatea minimă a oricărei acoperiri nod din G.*
Demonstraţie. Definim reţeaua $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{c})$, unde $\widetilde{N} = \widetilde{N}_1 \cup \widetilde{N}_2 \cup \widetilde{N}_3, \widetilde{N}_1 = N_1, \widetilde{N}_2 = N_2, \widetilde{N}_3 = \{s,t\}$, $\widetilde{A} = \widetilde{A}_1 \cup \widetilde{A}_2, \widetilde{A}_1 = A, \widetilde{A}_2 = \{(s,x),\ (y,t) \mid x \in \widetilde{N}_1, y \in \widetilde{N}_2\}$, $\widetilde{c} : \widetilde{A} \to \Re_+, \widetilde{c}(x,y) = \infty,\ (x,y) \in \widetilde{A}_1, \widetilde{c}(x,y) = 1,\ (x,y) \in \widetilde{A}_2$. Fie $\widetilde{f}^*$ fluxul maxim de valoare $\widetilde{v}^*$ în reţeaua modificată $\widetilde{G}$. Putem descompune fluxul $\widetilde{f}^*$ în fluxuri de-a lungul a $\widetilde{v}^*$ drumuri de forma $\widetilde{D} = (s,x,y,t)$, fiecare drum transportând o unitate de flux. Astfel $\widetilde{v}^*$ arce $(x,y) \in \widetilde{A}_1 = A$ au $\widetilde{f}^*(x,y) = 1$. Deoarece aceste drumuri sunt disjuncte nod, mulţimea arcelor $(x,y) \in \widetilde{A}_1 = A$ cu $\widetilde{f}^*(x,y) = 1$ constituie un cuplaj în G. Din faptul că fluxul $\widetilde{f}^*$ este maxim rezultă că, cuplajul definit de acest flux este de cardinalitate maximă egală cu $\widetilde{v}^*$. Invers, un cuplaj de cardinalitate maximă $\widetilde{v}^*$ defineşte un flux maxim $\widetilde{f}^*$ în reţeaua $\widetilde{G}$.

Fie $[\widetilde{X}^*, \overline{\widetilde{X}^*}]$ o tăietură $s-t$ minimă din reţeaua $\widetilde{G}$. Deoarece $\widetilde{c}(x,y) = \infty$ pentru $(x,y) \in \widetilde{A}_1 = A$, rezultă că mulţimea arcelor directe $(\widetilde{X}^*, \overline{\widetilde{X}^*})$ conţine numai arce $(x,y) \in \widetilde{A}_2$ cu $\widetilde{c}(x,y) = 1$. Deci $c[\widetilde{X}^*, \overline{\widetilde{X}^*}] = c(\widetilde{X}^*, \overline{\widetilde{X}^*}) = |(\widetilde{X}^*, \overline{\widetilde{X}^*})|$. Din mulţimea arcelor directe $(\widetilde{X}^*, \overline{\widetilde{X}^*})$ construim mulţimea de noduri Y^* în modul următor. Fiecare arc $(x,y) \in \widetilde{A}_1 = A$ defineşte un drum $\widetilde{D} = (s,x,y,t)$. Deoarece $[\widetilde{X}^*, \overline{\widetilde{X}^*}]$ este o tăietură $s-t$, fie $(s,x) \in (\widetilde{X}^*, \overline{\widetilde{X}^*})$, fie $(y,t) \in (\widetilde{X}^*, \overline{\widetilde{X}^*})$, fie amândouă. Atunci fie îl adăugăm pe x la Y^*, fie pe y, fie pe amândouă. Evident, că Y^* construit astfel este o acoperire nod cu $|Y^*| = |(\widetilde{X}^*, \overline{\widetilde{X}^*})|$ şi deoarece $[\widetilde{X}^*, \overline{\widetilde{X}^*}]$ este o tăietură $s-t$ minimă acoperirea nod Y^* este de cardinalitate minimă. Reciproc, dacă Y^* este o acoperire nod din reţeaua G se poate defini o tăietură $[\widetilde{X}^*, \overline{\widetilde{X}^*}]$ în reţeaua $\widetilde{G}$ cu capacitatea $|Y^*|$ în modul următor. Fie $y \in Y^*$. Dacă $y \in N_1$, atunci adăugăm arcul (s,y) la $(\widetilde{X}^*, \overline{\widetilde{X}^*})$, iar dacă $y \in N_2$ atuci adăugăm arcul (y,t) la $(\widetilde{X}^*, \overline{\widetilde{X}^*})$ şi evident că $c[\widetilde{X}^*, \overline{\widetilde{X}^*}] = c(\widetilde{X}^*, \overline{\widetilde{X}^*}) = |(\widetilde{X}^*, \widetilde{X}^*)| = |Y^*|$.

Conform teoremei fluxului maxim şi tăieturii minime avem $\widetilde{v}^* = c[\widetilde{X}^*, \overline{\widetilde{X}^*}] = |(\widetilde{X}^*, \overline{\widetilde{X}^*})| = |Y^*|$ şi utilizând concluziile de mai sus rezultă afirmaţia teoremei.■
Exemplul 5.7. Ilustrăm Teorema 5.15 în exemplul prezentat în figura 5.13.

Fig. 5.13. Tăietura minimă pentru problema fluxului maxim corespunzătoare figurii 5.12

Problema de cuplaj din figura 5.12 am transformat-o într-o problemă de flux maxim în reţeaua din figura 5.13. Mulţimea arcelor directe ale tăieturii minime este $(\widetilde{X}^*, \overline{\widetilde{X}^*}) = \{(s,1),(s,3),(2',t),(5',t)\}$. Corespunzător, mulţimea $Y^* = \{1,3,2',5'\}$ este o acoperire nod de cardinalitate minimă şi un cuplaj de cardinalitate maximă este $\{(1,1'),(2,2'),(3,3'),(5,5')\}$.

5.6 Fluxuri cu margini inferioare pozitive

În acest caz problema fluxului maxim constă în a determina un flux f care verifică condiţiile

$$\text{Maximizează } v \tag{5.6.a}$$

$$f(x,N) - f(N,x) = \begin{cases} v, & x = s, \\ 0, & x \neq s,t, \\ -v, & x = t, \end{cases} \tag{5.6.b}$$

$$l(x,y) \leq f(x,y) \leq c(x,y),\ (x,y) \in A. \tag{5.6.c}$$

Fie $[X, \overline{X}] = (X, \overline{X}) \cup (\overline{X}, X)$ o tăietură $s-t$. În acest caz *capacitatea tăieturii* $s-t$ este definită prin

$$c[X, \overline{X}] = c(X, \overline{X}) - l(\overline{X}, X).$$

Evident, dacă $l(x,y) = 0$ pentru $(x,y) \in A$ atunci $c[X, \overline{X}] = c(X, \overline{X})$ şi se obţine relaţia prin care s-a definit capacitatea unei tăieturi în cazul când marginile inferioare sunt zero.

Problema fluxului maxim cu margini inferioare zero are întotdeauna o soluţie admisibilă şi anume fluxul zero. Problema fluxului maxim cu margini inferioare pozitive poate fi inadmisibilă. De exemplu, considerăm problema fluxului maxim în reţeaua reprezentată în figura 5.14. Această problemă nu are soluţie admisibilă

Fig. 5.14. Problema fluxului maxim cu soluţie inadmisibilă

deoarece pe arcul (1,2) trebuie să transportăm cel puţin 5 unităţi de flux şi pe arcul (2,3) putem să transportăm cel mult 4 unităţi de flux şi pentru nodul 2 nu se verifică constrângerea din (5.6 b).

Deci problema fluxului maxim cu margini inferioare pozitive se rezolvă în două faze. În prima fază se determină un flux admisibil dacă el există. În a doua fază, plecând cu fluxul admisibil determinat în prima fază, se determină un flux maxim. Problema din fiecare fază se reduce la a rezolva o problemă de flux maxim cu margini inferioare zero. Mai întâi prezentăm determinarea unui flux maxim.

Presupunem că avem un flux admisibil f de valoare v în reţeaua $G = (N, A, l, c)$. În acest caz definim capacitatea reziduală a oricărui arc (x, y) ca $r(x, y) = (c(x, y) - f(x, y)) + (f(y, x) - l(y, x))$; termenul $c(x, y) - f(x, y)$ reprezintă creşterea maximă de flux pe arcul (x, y) utilizând capacitatea rămasă şi termenul $f(y, x) - l(y, x)$ reprezintă creşterea maximă de flux pe arcul (x, y) prin eliminarea fluxului excedent pe arcul (y, x). Deoarece algoritmii care au fost şi vor fi prezentaţi lucrează numai cu capacităţi reziduale, putem utiliza oricare dintre aceşti algoritmi pentru a determina un flux maxim cu margini inferioare pozitive. Oricare din aceşti algoritmi se termină cu capacităţi reziduale optime. Din aceste capacităţi reziduale optime se poate construi fluxul maxim prin mai multe metode. Prezentăm următoarea metodă. Se face schimbarea de variabile $c'(x, y) = c(x, y) - l(x, y)$, $f'(x, y) = f(x, y) - l(x, y)$, $r'(x, y) = r(x, y)$. Capacitatea reziduală pe arcul (x, y) este $r(x, y) = (c(x, y) - f(x, y)) + (f(y, x) - l(y, x))$. Echivalent se obţine $r'(x, y) = c'(x, y) - f'(x, y) + f'(y, x)$. Aceste capacităţi reziduale sunt similare cu capacităţile reziduale pentru cazul când marginile inferioare sunt zero. Calculăm fluxul f' în funcţie de r' şi c' ca în cazul când marginile inferioare sunt zero şi obţinem $f'(x, y) = \max\{0, c'(x, y) - r'(x, y)\}$ şi $f'(y, x) = \max\{0, c'(y, x) - r'(y, x)\}$. Revenind la variabilele originare obţinem $f(x, y) = l(x, y) + \max\{0, c(x, y) - r(x, y) - l(x, y)\}$ şi $f(y, x) = l(y, x) + \max\{0, c(y, x) - r(y, x) - l(y, x)\}$.

Teorema 5.16. (Teorema generalizată a fluxului maxim şi tăieturii minime). *Într-o reţea $G = (N, A, l, c)$, valoarea fluxului maxim este egală cu capacitatea tăieturii $s - t$ minime.*

Demonstraţie. Fie f un flux admisibil în reţeaua $G = (N, A, l, c)$ şi $[X, \overline{X}] = (X, \overline{X}) \cup (\overline{X}, X)$ o tăietură $s - t$. Fluxul f verifică constrângerile de conservare (5.6 b). Sumând aceste ecuaţii pentru $x \in X$ se obţine $v = f(X, N) - f(N, X)$. Înlocuind $N = X \cup \overline{X}$ şi dezvoltând rezultă $v = f(X, \overline{X}) - f(\overline{X}, X) = f[X, \overline{X}]$. De asemenea f verifică constrângerile de mărginire a fluxului (5.6 c) şi se obţine $f(X, \overline{X}) \leq c(X, \overline{X})$, $l(\overline{X}, X) \leq f(\overline{X}, X)$. Rezultă $v = f[X, \overline{X}] = f(X, \overline{X}) - f(\overline{X}, X) \leq c(X, \overline{X}) - l(\overline{X}, X) = c[X, \overline{X}]$. Deci, dacă f^* este un flux maxim de valoare v^* şi $[X^*, \overline{X}^*]$ este o tăietură $s - t$ minimă atunci $v^* = f^*[X^*, \overline{X}^*] = c[X^*, \overline{X}^*]$. ■

În continuare prezentăm problema determinării unui flux admisibil dacă el există. Transformăm problema fluxului admisibil f din reţeaua $G = (N, A, l, c)$

în problema circulaţiei admisibile $\widetilde{f}$ în reţeaua transformată $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{l}, \widetilde{c})$ unde $\widetilde{N} = N,\ \widetilde{A} = A \cup \{(t,s)\}, \widetilde{l}(x,y) = l(x,y),\ \widetilde{c}(x,y) = c(x,y),\ (x,y) \in A,\ \widetilde{l}(t,s) = 0,\ \widetilde{c}(t,s) = \infty$. Problema circulaţiei admisibile constă în a determina un flux $\widetilde{f}$, dacă există, satisfăcând constrângerile:

$$\widetilde{f}(x,\widetilde{N}) - \widetilde{f}(\widetilde{N},x) = 0,\ x \in \widetilde{N} \tag{5.7.a}$$

$$\widetilde{l}(x,y) \le \widetilde{f}(x,y) \le \widetilde{c}(x,y),\ (x,y) \in \widetilde{A} \tag{5.7.b}$$

Înlocuim $\widetilde{f}(x,y)$ prin $\widetilde{f}'(x,y) + \widetilde{l}(x,y)$ şi se obţine

$$\widetilde{f}'(x,\widetilde{N}) - \widetilde{f}'(\widetilde{N},x) = \widetilde{v}'(x),\ x \in \widetilde{N} \tag{5.8.a}$$

$$0 \le \widetilde{f}'(x,y) \le \widetilde{c}'(x,y),\ (x,y) \in \widetilde{A} \tag{5.8.b}$$

cu $\widetilde{c}'(x,y) = \widetilde{c}(x,y) - \widetilde{l}(x,y)$ şi ofertele / cererile $\widetilde{v}'(x)$ la noduri definite prin

$$\widetilde{v}'(x) = \widetilde{l}(\widetilde{N},x) - \widetilde{l}(x,\widetilde{N}),\ x \in N \tag{5.8.c}$$

Se observă că $\widetilde{v}'(\widetilde{N}) = 0$. Astfel problema circulaţiei admisibile este echivalentă cu a determina dacă problema transformată are o soluţie $\widetilde{f}'$ care verifică (5.8). În problema ofertei şi cererii prezentată în paragraful 5.1 am arătat că prin rezolvarea unei probleme de flux maxim, fie determinăm o soluţie care satisface (5.8), fie arătăm că soluţia nu satisface (5.8). Dacă $\widetilde{f}'$ este o soluţie admisibilă pentru constrângerile (5.8) atunci $\widetilde{f}$ cu $\widetilde{f}(x,y) = \widetilde{f}'(x,y) + \widetilde{l}(x,y)$ este o soluţie admisibilă pentru constrângerile (5.7).

Exemplul 5.8. Fie reţeaua $G = (N, A, l, c)$ reprezentată în figura 5.15 unde pe fiecare arc (x,y) sunt trecute $l(x,y)$ şi $c(x,y)$ în această ordine. Dacă există un flux admisibil, atunci să se determine un flux maxim.

Fig. 5.15.

Pentru a rezolva problema ofertă-cerere (5.8) se construieşte reţeaua $\widetilde{G}'_0 = (\widetilde{N}'_0, \widetilde{A}'_0, \widetilde{c}'_0)$ unde: $\widetilde{N}'_0 = \widetilde{N}'_1 \cup \widetilde{N}'_2 \cup \widetilde{N}'_3,\ \widetilde{N}'_1 = \{\widetilde{s}\}, \widetilde{N}'_2 = \widetilde{N}, \widetilde{N}'_3 = \{\widetilde{t}\};\ \widetilde{A}'_0 = \widetilde{A}'_1 \cup \widetilde{A}'_2 \cup \widetilde{A}'_3, \widetilde{A}'_1 = \{(\widetilde{s},x) \mid \widetilde{v}'(x) > 0\}, \widetilde{A}'_2 = \widetilde{A}, \widetilde{A}'_3 = \{(x,\widetilde{t}) \mid \widetilde{v}'(x) < 0\};\ \widetilde{c}'_0(\widetilde{s},x) = \widetilde{v}'(x),\ (\widetilde{s},x) \in \widetilde{A}'_1,\ \widetilde{c}'_0(x,y) = \widetilde{c}'(x,y), (x,y) \in \widetilde{A}'_2, \widetilde{c}'_0(x,\widetilde{t}) = -\widetilde{v}'(x), (x,\widetilde{t}) \in \widetilde{A}'_3$. Reţeaua $\widetilde{G}'_0$ este reprezentată în figura 5.16, unde capacitatea $\widetilde{c}'_0$ este reprezentată de al doilea număr asociat arcului (x,y).

Primul număr asociat arcului (x,y) din reţeaua $\widetilde{G}'_0$ reprezintă fluxul maxim $\widetilde{f}'_0$. Deoarece $\widetilde{f}'_0$ saturează toate arcele din $\widetilde{A}'_1$ şi $\widetilde{A}'_3$ rezultă că $\widetilde{f}'$, restricţia lui

Fig. 5.16.

$\widetilde{f}'_0$ la $\widetilde{A}'_2 = \widetilde{A}$, este o soluţie pentru problema ofertă - cerere (5.8). Marginea inferioară $l(x,y)$, fluxul admisibil $f(x,y) = l(x,y) + \widetilde{f}'(x,y)$ şi capacitatea $c(x,y)$ sunt asociate fiecărui arc (x,y), în această ordine, din reţeaua $G = (N, A, l, c)$ reprezentată în figura 5.17. Reţeaua reziduală $\widetilde{G}(f') = (\widetilde{N}, \widetilde{A}, r')$ cu $f'(x,y) =$

Fig. 5.17.

$f(x,y) - l(x,y),\ c'(x,y) = c(x,y) - l(x,y),\ r'(x,y) = c'(x,y) - f'(x,y) + f'(y,x)$ este reprezentată în figura 5.18. Evident că $\widetilde{G}(f') \equiv \widetilde{G}(f)$.

Fig. 5.18.

Aplicând algoritmul de etichetare Ford-Fulkerson se obţine în final reţeaua reziduală din figura 5.19.

Fig. 5.19.

Reţeaua $G = (N, A, l, c)$ cu $f(x,y) = l(x,y) + \max\{0, c(x,y) - r(x,y) - l(x,y)\}$ cu $r(x,y) = r'(x,y)$ este reprezentată în figura 5.20.

Fig. 5.20.

Tăietura minimă este $[X, \overline{X}] = [\{1,2\}, \{3,4,5,6\}] = \{(1,3),(2,4),(2,5)\} \cup \{(3,2)\}$ cu capacitatea $c[X, \overline{X}] = c(X, \overline{X}) - l(\overline{X}, X) = c(1,3) + c(2,4) + c(2,5) - l(3,2) = 2 + 6 + 4 - 2 = 10$. Valoarea fluxului maxim este $v = f(1,2) + f(1,3) = 8 + 2 = 10$. Deci se verifică teorema generalizată a fluxului maxim şi tăieturii minime $v = c[X, \overline{X}]$.

Teorema 5.17. (Teorema admisibilităţii circulaţiei). *Într-o reţea* $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{l}, \widetilde{c})$ *există o circulaţie admisibilă dacă şi numai dacă pentru orice mulţime* $\widetilde{X} \subset \widetilde{N}$ *este verificată condiţia*

$$\widetilde{l}(\overline{\widetilde{X}}, \widetilde{X}) \leq \widetilde{c}(\widetilde{X}, \overline{\widetilde{X}}) \tag{5.9}$$

Demonstraţie. *Necesitatea.* Presupunem că circulaţia $\widetilde{f}$ este admisibilă în reţeaua $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{l}, \widetilde{c})$. Aceasta înseamnă că circulaţia $\widetilde{f}$ verifică condiţiile (5.7 a) şi (5.7 b.)

Fie o mulţime oarecare $\widetilde{X} \subset \widetilde{N}$ în reţeaua $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{l}, \widetilde{c})$. Sumând condiţiile (5.7 a) pentru $x \in \widetilde{X}$ se obţine

$$\widetilde{f}(\widetilde{X}, \overline{\widetilde{X}}) - \widetilde{f}(\overline{\widetilde{X}}, \widetilde{X}) = 0. \tag{5.10}$$

Din condiţiile (5.7 b) rezultă că $\widetilde{f}(\widetilde{X}, \overline{\widetilde{X}}) \leq \widetilde{c}(\widetilde{X}, \overline{\widetilde{X}})$, $\widetilde{l}(\overline{\widetilde{X}}, \widetilde{X}) \leq \widetilde{f}(\overline{\widetilde{X}}, \widetilde{X})$. Utilizând aceste inegalităţi în (5.10) se obţine

$$\widetilde{l}(\overline{\widetilde{X}}, \widetilde{X}) \leq \widetilde{c}(\widetilde{X}, \overline{\widetilde{X}}).$$

Suficienţa. Presupunem că este verificată condiţia (5.9) pentru orice mulţime $\widetilde{X} \subset \widetilde{N}$. Pentru suficienţă se dă o demonstraţie algoritmică. Algoritmul porneşte cu o circulaţie $\widetilde{f}$ care verifică condiţiile (5.7 a) şi condiţiile pentru capacitate din (5.7 b), dar nu verifică condiţiile pentru marginile inferioare din (5.7 b). De exemplu $\widetilde{f} = 0$. Reţeaua reziduală $\widetilde{G}'(\widetilde{f})$ se defineşte conform celor precizate în acest paragraf cu deosebirea că pentru un arc (x, y) cu $\widetilde{f}(x, y) < \widetilde{l}(x, y)$ definim $\widetilde{r}'(x, y) = \widetilde{c}(x, y) - \widetilde{f}(x, y)$. Algoritmul este următorul:

```
(1)   PROGRAM CIRCULAŢIE;
(2)   BEGIN
(3)      f̃ := f̃₀;
(4)      WHILE există arc (x₀, y₀) ∈ Ã' cu f̃(x₀, y₀) < l̃(x₀, y₀) DO
(5)      BEGIN
(6)         IF există DMF de la y₀ la x₀ în G̃'(f̃)
(7)            THEN BEGIN
(8)                    se determină un DMF D̃' de la y₀ la x₀ în G̃'(f̃);
(9)                    se face mărirea de flux pe circuitul D̃' ∪ {(x₀, y₀)};
(10)                   se actualizează G̃'(f̃);
(11)                 END;
(12)           ELSE EXIT;
(13)     END;
(14)  END.
```

Dacă algoritmul se termină fără execuţia liniei (12), atunci se obţine o circulaţie $\widetilde{f}$ admisibilă. În caz contrar arătăm că circulaţia $\widetilde{f}$ este inadmisibilă. Presupunem că pentru determinarea unui DMF în $\widetilde{G}'(\widetilde{f})$ utilizăm algoritmul de etichetare Ford- Fulkerson. Linia (12) se execută când acest algoritm nu poate să determine un DMF de la y_0 la x_0 în $\widetilde{G}'(\widetilde{f})$. Fie $\widetilde{X}'$ mulţimea nodurilor etichetate. Este evident că $y_0 \in \widetilde{X}'$, $x_0 \in \overline{\widetilde{X}'}$ şi $\widetilde{r}'(\widetilde{X}', \overline{\widetilde{X}'}) = 0$. Pentru $\widetilde{N}' = \widetilde{N}$ rezultă că $\widetilde{X}' = \widetilde{X}$ şi din $\widetilde{r}'(\widetilde{X}, \overline{\widetilde{X}}) = 0$ rezultă că $\widetilde{f}(\widetilde{X}, \overline{\widetilde{X}}) = \widetilde{c}(\widetilde{X}, \overline{\widetilde{X}})$ şi $\widetilde{f}(\overline{\widetilde{X}}, \widetilde{X}) \leq \widetilde{l}(\overline{\widetilde{X}}, \widetilde{X})$. Pentru $(x_0, y_0) \in (\overline{\widetilde{X}}, \widetilde{X})$ avem $\widetilde{f}(x_0, y_0) < \widetilde{l}(x_0, y_0)$. Substituind aceste inegalităţi în (5.10) rezultă că

$$\widetilde{l}(\overline{\widetilde{X}}, \widetilde{X}) > \widetilde{c}(\widetilde{X}, \overline{\widetilde{X}}) \tag{5.11}$$

Dar condiţia (5.11) contrazice condiţia (5.9). Deci dacă este verificată condiţia (5.9) pentru orice $\widetilde{X} \subset \widetilde{N}$ atunci algoritmul se termină fără a executa linia (12) şi determină o circulaţie admisibilă $\widetilde{f}$. ■

Teorema 5.18. (Teorema admisibilităţii fluxului). *Într-o reţea $G = (N, A, l, c)$ există un flux admisibil dacă şi numai dacă pentru orice mulţime $X \subset N$ cu $s, t \in X$ sau $s, t \in \overline{X}$ este verificată condiţia*

$$l(\overline{X}, X) \leq c(X, \overline{X}) \tag{5.12}$$

Demonstraţie. Se construieşte reţeaua $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{l}, \widetilde{c})$, unde $\widetilde{N} = N$, $\widetilde{A} = A \cup \{(s,t),(t,s)\}$, $\widetilde{l}(x,y) = l(x,y)$, $\widetilde{c}(x,y) = c(x,y)$, $(x,y) \in A$, $\widetilde{l}(s,t) = \widetilde{l}(t,s) = 0, \widetilde{c}(s,t) = \widetilde{c}(t,s) = \infty$. Rezultă că există un flux admisibil în reţeaua G dacă şi numai dacă există o circulaţie admisibilă $\widetilde{f}$ în reţeaua $\widetilde{G}$. Conform Teoremei 5.17 există o circulaţie admisibilă $\widetilde{f}$ în reţeaua $\widetilde{G}$ dacă şi numai dacă pentru orice mulţime $\widetilde{X} \subset \widetilde{N}$ este verificată condiţia (5.9). Dacă $s \in \widetilde{X}$ şi $t \in \overline{\widetilde{X}}$ sau dacă $s \in \overline{\widetilde{X}}$, $t \in \widetilde{X}$, atunci $c(\widetilde{X}, \overline{\widetilde{X}}) = \infty$ şi condiţia (5.9) este verificată. Deci (5.9) este verificată pentru orice mulţime $\widetilde{X} \subset \widetilde{N}$ dacă şi numai dacă (5.12) este verificată pentru orice mulţime $X \subset N$ cu $s, t \in X$ sau $s, t \in \overline{X} (\widetilde{N} = N,\ \widetilde{X} = X)$.■

Teorema 5.19. (Teorema de admisibilitate a problemei ofertei şi cererii). *Într-o reţea $G = (N, A, c)$, problema ofertei şi cererii este admisibilă dacă şi numai dacă pentru orice mulţime $X \subset N$* este verificată condiţia

$$v(X) \leq c[X, \overline{X}] \tag{5.13}$$

Demonstraţie. Se construieşte reţeaua $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{l}, \widetilde{c})$, unde $\widetilde{N} = \widetilde{N}_1 \cup \widetilde{N}_2 \cup \widetilde{N}_3$, $\widetilde{N}_1 = \{\widetilde{s}\}, \widetilde{N}_2 = N, \widetilde{N}_3 = \{\widetilde{t}\}$; $\widetilde{A} = \widetilde{A}_1 \cup \widetilde{A}_2 \cup \widetilde{A}_3, \widetilde{A}_1 = \{(\widetilde{s}, x) \mid \widetilde{v}(x) > 0\}, \widetilde{A}_2 = A, \widetilde{A}_3 = \{(x, \widetilde{t}) \mid v(x) < 0\}$; $\widetilde{l}(\widetilde{s}, x) = \widetilde{c}(\widetilde{s}, x) = v(x)$, $(\widetilde{s}, x) \in \widetilde{A}_1$, $\widetilde{l}(x,y) = 0$; $\widetilde{c}(x,y) = c(x,y)$, $(x,y) \in \widetilde{A}_2, \widetilde{l}(x, \widetilde{t}) = \widetilde{c}(x, \widetilde{t}) = -v(x)$, $(x, \widetilde{t}) \in \widetilde{A}_3$. Este evident că, există un flux admisibil în G dacă şi numai dacă există un flux admisibil în $\widetilde{G}$. Aplicând Teorema 5.18 la reţeaua $\widetilde{G}$ rezultă Teorema 5.19 pentru reţeaua G.■

5.7 Aplicaţii şi comentarii bibliografice

5.7.1 Probleme de transport

În centrele $s_1, \ldots, s_p$ se găsesc cantităţile $u_1, \ldots, u_p$ de produse care urmează a fi transportate în centrele $t_1, \ldots, t_q$, fiecare centru având nevoie de cantităţile $v_1, \ldots, v_q$. Pe ruta de la centrul s_i la centrul t_j nu pot fi transportate mai mult de $c(s_i, t_j)$ produse. Problema constă în a determina cantităţile $f(s_i, t_j)$ astfel încât să fie verificate condiţiile:

$$\sum_{j=1}^{q} f(s_i, t_j) = u_i,\ i = 1, \ldots, p;$$

$$\sum_{i=1}^{p} f(s_i, t_j) = v_j,\ j = 1, \ldots, q;$$

$$0 \leq f(s_i, t_j) \leq c(s_i, t_j),\ i = 1, \ldots, p;\ j = 1, \ldots, q;$$

$$\sum_{i=1}^{p} u_i = \sum_{j=1}^{q} v_j.$$

Această problemă de transport constituie un caz particular (mulţimea nodurilor de tranzit $R = \emptyset$) al problemei ofertă - cerere prezentată în paragraful 5.1.

5.7.2 Un caz particular al problemei de afectare

Problema particulară de afectare constituie un caz particular al problemei de transport. Problema constă în a determina cantităţile $f(s_i, t_j)$ astfel încât să fie verificate condiţiile:

$$\sum_{j=1}^{p} f(s_i, t_j) = 1,\ i = 1, \ldots, p;$$

$$\sum_{i=1}^{p} f(s_i, t_j) = 1,\ j = 1, \ldots, p;$$

$$0 \leq f(s_i, t_j) \leq \infty,\ i = 1, \ldots, p;\ j = 1, \ldots, p.$$

Este evident că o problemă de afectare este o problemă a cuplajului maxim prezentată în paragraful 5.5.

Un exemplu care ilustrează acest tip de problemă este acela al repartiţiei solicitanţilor pe posturi. Un număr de solicitanţi, cu aptitudini multiple, urmează să fie repartizaţi la posturi. Fiecare solicitant este calificat pentru mai multe posturi. Se cere ca solicitanţii să ocupe numai posturi pentru care sunt calificaţi şi nici un post să nu rămână neocupat.

5.7.3 Planificarea lucrărilor pe maşini paralele

Considerăm că avem J lucrări şi M maşini paralele. Fiecare lucrare j, $j = 1, \cdots, J$ are un timp necesar prelucrării notat cu p_j (care reprezintă numărul de zile necesare unei maşini pentru a efectua lucrarea), o dată de apariţie a_j (reprezentând începutul zilei în care lucrarea j a devemit disponibilă pentru prelucrare) şi un termen limită t_j (reprezentând începutul zilei în care lucrarea j trebuie să fie gata). O maşină poate efectua o singură lucrare la un moment dat

şi o lucrare poate fi procesată de către o singură maşină la un moment dat, dar este permisă întreruperea procesării oricărei lucrări, care poate fi reluată într-o altă zi, de către o altă maşină. Problema planificării lucrărilor pe maşini paralele constă în a determina o planificare astfel încât toate lucrările să fie efectuate înaintea termenelor limită sau a arăta că nu este posibilă o asemenea planificare.

Problema planificării lucrărilor pe maşini paralele se poate modela ca o problemă de flux maxim în reţeaua $G = (N, A, c)$ construită în modul următor. Mai întâi sortăm crescător elementele mulţimii $\{a_j,\, t_j \mid j = 1, \cdots, J\}$ şi stabilim $P \leq 2J - 1$ intervale disjuncte determinate de elemente consecutive. Notăm cu $T_{k\ell}$ intervalul de la începutul zilei k la începutul zilei $\ell + 1$. Observăm că în intervalul $T_{k\ell}$ mulţimea lucrărilor care pot fi prelucrate nu se schimbă. În intervalul $T_{k\ell}$ putem procesa toate lucrările j cu proprietatea că $a_j \leq k$ şi $t_j \geq \ell + 1$. Reţeaua $G = (N, A, c)$ se determină astfel: $N = \{s\} \cup N_1 \cup N_2 \cup \{t\},\ N_1 = \{j \mid j = 1, \cdots J\},\ N_2 = \{T_{k\ell}^{(i)} \mid i = 1, \cdots, P\},\ A = A_1 \cup A_2 \cup A_3,\ A_1 = \{(s, j) \mid j \in N_1\},\ A_2 = \{(j, T_{k\ell}) \mid j \in N_1,\, T_{k\ell} \in N_2,\, a_j \leq k,\, t_j \geq \ell + 1\},\ A_3 = \{(T_{k\ell}, t) \mid T_{k\ell} \in N_2\},\, c(s, j) = p_j,\ (s, j) \in A_1,\, c\,(j, T_{k,\ell}) = \ell - k + 1,\, (j, T_{k\ell}) \in A_2,\ c\,(T_{k\ell}, t) = (\ell - k + 1)M$. Apoi se determină un flux maxim în reţeaua G. Este uşor de arătat că există o corespondenţă biunivocă între planificările admisibile şi fluxurile de valoare $\sum_{j=1}^{J} p_j$. Deci, dacă valoarea fluxului maxim este $\sum_{j=1}^{J} p_j$ atunci problema planificării lucrărilor pe maşini paralele are soluţii, altfel nu.

Exemplul 5.9. Pentru $J = 4$, $M = 3$ se dau timpii p_j, a_j, t_j în tabelul de mai jos.

Lucrarea (j)	1	2	3	4
Timpul de prelucrare (p_j)	1.5	1.25	2.1	3.6
Data apariţiei (a_j)	3	1	3	5
Termenul limită (t_j)	5	4	7	9

Mulţimea datelor $\{a_j,\, t_j \mid j = 1, \cdots, 4\}$ sortată crescător este $\{1, 3, 4, 5, 7, 9\}$. Deci, intervalele disjuncte vor fi T_{12}, T_{33}, T_{44}, T_{56} şi T_{78}.

Reţeaua $G = (N, A, c)$ este reprezentată în figura 5.21.

Fig.5.21

Un flux maxim este reprezentat în reţeaua din figura 5.22.

Fig.5.22

Deoarece valoarea fluxului maxim, 8.45, este egală cu $\sum_{j=1}^{4} p_j$, rezultă că există o planificare admisibilă a lucrărilor. O planificare corespunzătoare fluxului din figura 5.22 este următoarea: prima maşină procesează începând din ziua 1 lucrarea 2 până la terminarea ei, apoi începând din ziua 3 lucrarea 1 până la

terminarea ei şi apoi începând din ziua 5 lucrarea 4 până la terminarea ei. A doua maşină procesează o parte (0.1) din lucrarea 3 începând din ziua 3, iar a treia maşină va începe procesarea lucrării 3 în ziua 5 şi o va termina în ziua 7.

5.7.4 Comentarii bibliografice

Articolul seminal al lui Ford şi Fulkerson (1956) pentru problema fluxului maxim, stabileşte celebra teoremă a fluxului maxim şi tăieturii minime. Ford şi Fulkerson (1956) şi Elias, Feinstein şi Shannon (1956) rezolvă problema fluxului maxim prin algoritmi folosind drumuri de mărire a fluxului. Algoritmul de etichetare care a fost prezentat în paragraful 5.4 este datorat lui Ford şi Fulkerson (1956); cartea lor clasică, Ford şi Fulkerson (1962), oferă o tratare extinsă a acestui algoritm. Din nefericire, algoritmul de etichetare se execută în timp pseudopolinomial; mai mult, aşa cum a arătat Ford şi Fulkerson (1956), pentru reţele cu capacităţile arc numere iraţionale, algoritmul se poate executa într-o secvenţă infinită de măriri ale fluxului şi poate converge la o valoare diferită de valoarea fluxului maxim. În capitolul 6 sunt prezentaţi algoritmi pentru determinarea fluxului maxim în timp polinomial. În paragraful 5.5 sunt prezentate implicaţii combinatoriale ale teoremei fluxului maxim şi tăieturii minime. Teoremele 5.13 şi 5.14 sunt cunoscute ca teoremele lui Menger. Teorema 5.15 este cunoscută ca teorema K*ö*nig - Egerv*á*ry. Ford şi Fulkerson (1962) discută aceste teoreme şi multe rezultate adiţionale care sunt demonstrabile utilizând teorema fluxului maxim şi a tăieturii minime.

În paragraful 5.6 este prezentată admisibilitatea problemei fluxului într-o reţea cu margini inferioare pozitive impuse pe fluxurile arc. Teorema 5.17 este dată de Hoffman (1960) şi Teorema 5.19 este dată de Gale (1957).

Alte aplicaţii şi comentarii bibliografice privind fluxul maxim sunt prezentate în paragraful 6.3.

Capitolul 6

Algoritmi polinomiali pentru fluxul maxim

6.1 Algoritmi polinomiali cu drumuri de mărire a fluxului

6.1.1 Etichete distanţă

Definiţia 6.1. Într-o reţea reziduală $\widetilde{G}(f) = (\widetilde{N}, \widetilde{A}, r)$ o funcţie $d : \widetilde{N} \to \mathcal{N}$ se numeşte *funcţia distanţă*. Funcţia distanţă d se numeşte *validă* dacă satisface următoarele două condiţii:

$$d(t) = 0, \tag{6.1}$$

$$d(x) \leq d(y) + 1, \ (x, y) \in \widetilde{A}. \tag{6.2}$$

Valoarea $d(x)$ se numeşte *eticheta distanţă* a nodului x şi condiţiile (6.1), (6.2) se numesc *condiţii de validitate*.

Proprietatea 6.1. *Dacă etichetele distanţă sunt valide, atunci eticheta distanţă $d(x)$ este o margine inferioară pentru lungimea drumului cel mai scurt de la nodul x la nodul t în reţeaua reziduală $\widetilde{G}(f)$.*

Demonstraţie. Fie $D = (x_1, x_2, \ldots, x_k, x_{k+1})$ cu $x_1 = x, x_{k+1} = t$ un drum oarecare de lungime k de la nodul x la nodul t din reţeaua reziduală $\widetilde{G}(f)$. Condiţiile de validitate implică faptul că

$$d(x_k) \leq d(x_{k+1}) + 1 = d(t) + 1 = 1$$

$$d(x_{k-1}) \leq d(x_k) + 1 \leq 2$$

.....

$$d(x_1) \leq d(x_2) + 1 \leq k$$

Deci $d(x) = d(x_1) \leq k$ ce demonstrează afirmaţia proprietăţii. ■

Proprietatea 6.2. *Dacă $d(s) \geq n$, atunci reţeaua reziduală $\widetilde{G}(f)$ nu conţine drum de la nodul sursă s la nodul stoc t.*

Demonstraţie. Eticheta distanţă $d(s)$ este o margine inferioară pentru lungimea drumului cel mai scurt de la s la t în reţeaua reziduală $\widetilde{G}(f)$. Orice drum elementar de la s la t are lungimea cel mult $n-1$. Deci dacă $d(s) \geq n$ reţeaua reziduală $\widetilde{G}(f)$ nu conţine drum de la nodul s la t. ■

Definiţia 6.2. Etichetele distanţă se numesc *exacte* dacă pentru fiecare nod x, $d(x)$ este egală cu lungimea drumului cel mai scurt de la nodul x la nodul t în reţeaua reziduală $\widetilde{G}(f)$.

Exemplul 6.1. Se consideră reţeaua din figura 6.1 în care $s = 1$ şi $t = 4$. Un

Fig. 6.1.

vector valid de etichete distanţă este $d = (0, 0, 0, 0)$ şi vectorul de etichete distanţă exacte este $d = (3, 1, 2, 0)$. Etichetele distanţă exacte se pot determina cu ajutorul algoritmului parcurgeri inverse BF aplicat reţelei reziduale $\widetilde{G}(f)$ plecând de la nodul stoc t. Acest algoritm are complexitatea $O(m)$.

Definiţia 6.3. Un arc (x, y) din reţeaua reziduală $\widetilde{G}(f)$ se numeşte *admisibil* dacă el satisface condiţia $d(x) = d(y) + 1$, altfel arcul (x, y) se numeşte *inadmisibil*. Un drum de la nodul sursă s la nodul stoc t din reţeaua reziduală $\widetilde{G}(f)$ se numeşte *drum admisibil* dacă conţine numai arce admisibile, altfel se numeşte *inadmisibil.*

Proprietatea 6.3.*În reţeaua reziduală $\widetilde{G}(f)$ un drum admisibil de la nodul sursă s la nodul stoc t este un cel mai scurt drum de mărire a fluxului.*

Demonstraţie. Deoarece fiecare arc (x, y) dintr-un drum admisibil D este admisibil, capacitatea reziduală $r(x, y)$ şi etichetele distanţă $d(x), d(y)$ verifică condiţiile: $(1) r(x, y) > 0$, $(2) d(x) = d(y) + 1$. Condiţia (1) implică faptul că D este un drum de mărire a fluxului. Condiţia (2) implică faptul că dacă D conţine k arce, atunci $d(s) = k$. Deoarece $d(s)$ este o margine inferioară pentru lungimea drumului cel mai scurt de la nodul sursă s la nodul stoc t din reţeaua reziduală $\widetilde{G}(f)$, drumul D trebuie să fie un cel mai scurt drum de mărire a fluxului.■

6.1.2 Algoritmul Gabow al scalării bit a capacităţii

Teorema 6.4.*Dacă f_1^* este un flux maxim de valoare v_1^* pentru capacităţile scalate $c_1(x, y) = \lfloor c(x, y)/2 \rfloor$, $(x, y) \in A$ atunci*

(i) $v^* - 2v_1^* \leq m$ pentru orice flux maxim f^* de valoare v^* din reţeaua $G = (N, A, c)$;

(ii) în reţeaua reziduală $\widetilde{G}(2f_1^*)$ există cel mult m drumuri de mărire a fluxului şi capacităţile reziduale ale lor sunt egale cu 1.

Demonstraţie.(i) Fie $[X^*, \overline{X}^*]$o tăietură minimă $s - t$ în reţeaua $G = (N, A, c)$.

Deoarece $c(x,y) \leq 2c_1(x,y)+1$ rezultă $v^* = c\left(X^*, \overline{X}^*\right) \leq 2c_1\left(X^*, \overline{X}^*\right) + m = 2v_1^* + m$ şi deci $v^* - 2v_1^* \leq m$;
(ii) Dacă f_1^* este un flux maxim pentru capacitatea c_1 atunci în reţeaua reziduală $\widetilde{G}(2f_1^*)$ nu există drum de mărire a fluxului cu capacitatea reziduală mai mare decât 1. Deoarece $v^* - 2v_1^* \leq m$, rezultă că numărul acestor drumuri este cel mult m. ■

Dacă $c_0(x,y) = c(x,y),\ c_k(x,y) = \lfloor c_{k-1}(x,y)/2 \rfloor,\ (x,y) \in A,\ k = 1, \ldots, \lfloor \log \overline{c} \rfloor$, atunci notăm cu G_k reţeaua care are capacitatea $c_k,\ G_k = (N, A, c_k)$ şi cu f_k un flux oarecare, f_k^* un flux maxim în G_k. Algoritmul Gabow al scalării bit a capacităţii este următorul:

```
(1)   PROGRAM SCALARE BIT;
(2)   BEGIN
(3)     k := 0; c_0 = c;
(4)     WHILE max{c_k(x,y) | (x,y) ∈ A} > 1 DO
(5)     BEGIN
(6)       FOR (x,y) ∈ A DO
(7)         c_{k+1}(x,y) := ⌊c_k(x,y)/2⌋;
(8)       k := k + 1;
(9)     END;
(10)    f*_{k+1} := 0;
(11)    WHILE k ≥ 0 DO
(12)    BEGIN
(13)      fluxul iniţial este 2f*_{k+1};
(14)      se determină f*_k în G_k;
(15)      k := k − 1;
(16)    END;
(17)  END.
```

Conform definiţiei de mai sus avem $G_0 = G$. Afirmaţiile Teoremei 6.4 sunt demonstrate pentru G_0 şi G_1. Evident că aceste afirmaţii rămân adevărate pentru G_{k-1} şi $G_k,\ k = 1, \ldots, \lfloor \log \overline{c} \rfloor$.

Teorema 6.5. (Teorema de corectitudine a algoritmului) *Algoritmul Gabow al scalării bit a capacităţii determină un flux maxim f^* în reţeaua $G = (N, A, c)$.*

Demonstraţie. Iniţial algoritmul determină un flux maxim f_k^* în reţeaua $G_k = (N, A, c_k)$ pornind cu $2f_{k+1}^* = 0,\ k = \lfloor \log \overline{c} \rfloor$. Algoritmul se termină când $k < 0$. La ultima iteraţie se calculează un flux maxim f_0^* în $G_0 = G$. Deci $f^* = f_0^*$ este un flux maxim în G.■

Teorema 6.6. (Teorema de complexitate a algoritmului). *Algoritmul Gabow al scalării bit a capacităţii are complexitatea $O\left(m^2 \log \overline{c}\right)$.*

Demonstraţie. La fiecare iteraţie k algoritmul rezolvă o problemă P_k de flux

maxim în reţeaua G_k, $k = \lfloor \log \overline{c} \rfloor, \ldots, 0$. Conform Teoremei 6.4, reţeaua reziduală $\widetilde{G}(2f^*_{k+1})$ conţine cel mult m drumuri D_k de mărire a fluxului cu capacităţile reziduale $r(D_k) = 1$. Determinarea unui drum D_k cu algoritmul de etichetare are complexitatea $O(m)$. Rezultă că rezolvarea problemei P_k are complexitatea $O\left(m^2\right)$. Deoarece $k = \lfloor \log \overline{c} \rfloor, \ldots, 0$ se obţine că algoritmul are complexitatea $O\left(m^2 \log \overline{c}\right)$. ■

Algoritmul scalării bit a capacităţii se datorează lui Gabow (1985) şi de aceea se numeşte *algoritmul Gabow al sçalării bit a capacităţii.*

Exemplul 6.2. Să considerăm reţeaua din figura 6.2.

Fig. 6.2.

Tabelul 6.1. **Capacităţile c_k şi fluxurile f_k^*; $f^* = f_0^*$ este fluxul maxim**

6.1.3 Algoritmul Ahuja-Orlin al scalării maxime a capacităţii

Ideea esenţială ce fundamentează algoritmul Ahuja-Orlin al scalării maxime a capacităţii este conceptual destul de simplă. Mărim fluxul de-a lungul unui drum cu o capacitate reziduală suficient de mare, în locul unui drum cu o capacitate reziduală maximă, deoarece se poate obţine un drum de mărire a fluxului cu capacitatea reziduală suficient de mare mult mai uşor decât un drum de mărire a fluxului cu capacitatea reziduală maximă.

Definiţia 6.4. Fie f un flux oarecare de valoare v în reţeaua $G = (N, A, c)$. O reţea reziduală $\widetilde{G}(f, \overline{r}) = (N, \widetilde{A}(\overline{r}))$ cu $\widetilde{A}(\overline{r}) = \{(x, y) \mid (x, y) \in \widetilde{A},\ r(x, y) \geq \overline{r}\}$ se numeşte *reţea reziduală* $\overline{r}$.

Remarcăm faptul că $\widetilde{G}(f, 1) = \widetilde{G}(f)$.

Exemplul 6.3. Figura 6.3 (a) arată reţeaua reziduală $\widetilde{G}(f)$ şi figura 6.3 (b) arată reţeaua reziduală $\widetilde{G}(f, \overline{r})$ cu $\overline{r} = 8$.

Fig. 6.3.

Definiţia 6.5. O fază a algoritmului pe parcursul căreia $\overline{r}$ rămâne constant se numeşte *fază de scalare* şi o fază de scalare cu valoarea specificată a lui $\overline{r}$ se numeşte *fază de scalare* $\overline{r}$.

Într-o fază de scalare $\overline{r}$ fiecare drum D de mărire a fluxului are capacitatea reziduală $r(D) \geq \overline{r}$.

```
(1)   PROGRAM SCALARE MAXIMĂ;
(2)   BEGIN
(3)     f := 0;
(4)     r̄ := 2^⌊log c̄⌋;
(5)     WHILE r̄ ≥ 1 DO
(6)     BEGIN
(7)       WHILE G̃(f, r̄) conţine drum de la s la t DO
(8)       BEGIN
(9)         se identifică un drum D̃ de la s la t în G̃(f, r̄);
(10)        r(D̃) := min{r(x, y) | (x, y) ∈ D̃};
(11)        se execută mărirea de flux;
(12)        se actualizează G̃(f, r̄);
(13)      END;
(14)      r̄ := r̄/2;
(15)    END;
(16)  END.
```

Teorema 6.7. (Teorema de corectitudine a algoritmului) *Algoritmul Ahuja-Orlin al scalării maxime a capacităţii determină un flux maxim f^* în reţeaua $G = (N, A, c)$.*

Demonstraţie. Algoritmul porneşte cu $\overline{r} := 2^{\lfloor \log \overline{c} \rfloor}$ şi înjumătăţeşte valorile lui $\overline{r}$ până când $\overline{r} = 1$. În consecinţă algoritmul execută $1 + \lfloor \log \overline{c} \rfloor = O(\log \overline{c})$ faze de scalare. În ultima fază de scalare, $\overline{r} = 1$, astfel $\widetilde{G}(f, \overline{r}) = \widetilde{G}(f)$. Acest rezultat arată că algoritmul se termină cu un flux maxim f^* în reţeaua G.■

Teorema 6.8. (Teorema de complexitate a algoritmului) *Algoritmul Ahuja - Orlin al scalării maxime a capacităţii are complexitatea $O\left(m^2 \log \overline{c}\right)$.*

Demonstraţie. Fie faza de scalare $\overline{r}_k$ cu fluxul f_k^* de valoare v_k^* la sfârşitul acestei faze. Fie X_k^* mulţimea nodurilor atinse din s în $\widetilde{G}(f_k^*, \overline{r}_k)$. Deoarece $\widetilde{G}(f_k^*, \overline{r}_k)$ nu conţine drum de mărire a fluxului de la s la t, rezultă că $t \in \overline{X}_k^*$. Deci $\left[X_k^*, \overline{X}_k^*\right]$ este o tăietură $s - t$ în reţeaua $G = (N, A, c)$ cu $r\left(X_k^*, \overline{X}_k^*\right) \leq m\overline{r}_k$, deoarece $r(x, y) \leq \overline{r}_k, (x, y) \in \left(X_k^*, \overline{X}_k^*\right)$. Dacă f^* de valoare v^* este un flux maxim în reţeaua G, atunci conform Teoremei 5.7. avem $v^* - v_k^* \leq m\overline{r}_k$. În faza de scalare $\overline{r}_{k+1}$ fiecare drum D_{k+1} de mărire a fluxului are capacitatea reziduală $r(D_{k+1}) \geq \overline{r}_k/2$. Astfel această fază de scalare poate executa cel mult $2m$ măriri de flux. Determinarea unui drum de mărire a fluxului cu algoritmul de etichetare are complexitatea $O(m)$. Deci rezolvarea problemei P_k dintr-o fază de scalare $\overline{r}_k$ are complexitatea $O(m^2)$ şi deci algoritmul are complexitatea $O\left(m^2 \log \overline{c}\right)$. ■

Remarcăm faptul că este posibil de a reduce complexitatea algoritmilor scalării capacităţii la $O(mn \log \overline{c})$ dacă se determină un drum de mărire a fluxului cu algoritmul descris în secţiunea următoare. Deşi algoritmii scalării au aceeaşi complexitate, algoritmul scalării maxime a capacităţii este, din punct de vedere practic, mai performant decât algoritmul scalării bit a capacităţii.

Algoritmul scalării maxime a capacităţii se datorează lui Ahuja şi Orlin (1991) şi de aceea se numeşte *algoritmul Ahuja - Orlin al scalării maxime a capacităţii.*

6.1.4 Algoritmul Edmonds - Karp al drumului celui mai scurt

Algoritmul de etichetare Ford - Fulkerson are o complexitate $O(mn\overline{c})$, deci o complexitate timp pseudopolinomială. Edmonds şi Karp (1972) au obţinut o variantă îmbunătăţită a acestui algoritm cu o complexitate timp polinomială. În algoritmul de etichetare Ford - Fulkerson nodurile etichetate şi neanalizate din lista $\widetilde{V}$ nu sunt selectate după o anumită ordine pentru a le analiza. În algoritmul Edmonds - Karp aceste noduri sunt selectate pentru a le analiza pe principiul primul etichetat, primul analizat. Aceasta înseamnă că nodurile sunt analizate în ordinea în care ele sunt etichetate. O astfel de analizare presupune o parcurgere a reţelei reziduale mai întâi în lăţime pentru a găsi un DMF. Acest

DMF identificat prin strategia parcurgerii BF este un drum cel mai scurt de la nodul sursă s la nodul stoc t. De aceea, algoritmul se numeşte algoritmul Edmonds - Karp al drumului celui mai scurt. Acest algoritm se obţine din algoritmul de etichetare Ford - Fulkerson printr-o minoră modificare: lista nodurilor etichetate şi neanalizate $\widetilde{V}$ este organizată, sub aspectul unei structuri de date, ca o coadă.

Corectitudinea algoritmului drumului celui mai scurt Edmonds - Karp rezultă din corectitudinea algoritmului de etichetare Ford-Fulkerson. Determinarea unui DMF cu parcurgerea BF are complexitatea $O(m)$. Algoritmul drumului celui mai scurt Edmonds - Karp execută $O(mn)$ măriri de flux. Rezultă că acest algoritm are complexitatea $O\left(m^2n\right)$.

Exemplul 6.4. Să ilustrăm algoritmul Edmonds - Karp al drumului celui mai scurt pe reţeaua reprezentată în figura 6.4 (a). Fluxul iniţial este $f_0 = (f(1,2), f(1,3),$
$f(2,3), f(2,4), f(3,4)) = (1,0,1,0,1)$. Reţeaua reziduală $\widetilde{G}(f_0)$ este reprezentată în figura 6.4 (b).

(a) (b)

Fig. 6.4.

Se etichetează nodul $s = 1$ şi se stabileşte $\widetilde{V} = \{1\}$. Se scoate nodul 1 din $\widetilde{V}$ pentru analizare. Se etichetează nodurile 2, 3 şi se obţine lista $\widetilde{V} = \{2,3\}$, în această ordine. Vectorul predecesor este $\widetilde{p} = (0,1,1,0)$. Se scoate nodul 2 din $\widetilde{V}$ pentru analizare. Se etichetează nodul $t = 4$, lista $\widetilde{V}$ devine $\widetilde{V} = \{3,4\}$ şi vectorul predecesor devine $\widetilde{p} = (0,1,1,2)$. Deci $\widetilde{D} = (1,2,4)$ cu $r(\widetilde{D}) = u - 1$. Se execută mărirea de flux şi se continuă până când se determină un flux maxim.

6.1.5 Algoritmul Ahuja - Orlin al drumului celui mai scurt

Algoritmul Ahuja - Orlin al drumului celui mai scurt este o variantă îmbunătăţită a algoritmului Edmonds - Karp a drumului celui mai scurt . Algoritmul se datorează lui Ahuja şi Orlin (1991) şi se bazează pe etichetele distanţă. Algoritmul Ahuja - Orlin al drumului celui mai scurt (algoritmul AODS) măreşte fluxul de-a lungul drumurilor admisibile.

Definiţia 6.6. În reţeaua reziduală, un drum de la nodul sursă s la un nod oarecare x constând numai din arce admisibile se numeşte *drum parţial admisibil*. Nodul x se numeşte *nod curent*.

Reamintim ipoteza 4 din paragraful 5.2 potrivit căreia, dacă arcul $(x,y) \in A$ şi $(y,x) \notin A$, atunci se consideră că arcul $(y,x) \in A$ cu $c(y,x) = 0$. De asemenea, lista arcelor incidente către exterior la nodul x este $E^+(x) = \{(x,y) \mid (x,y) \in A\}$.

```
(1)  PROGRAM AODS;
(2)  BEGIN
(3)     f := 0;
(4)     se determină etichetele distanţă exacte d(x);
(5)     x := s;
(6)     WHILE d(s) < n DO
(7)     BEGIN
(8)        IF există arc (x, y) admisibil
(9)           THEN BEGIN
(10)                    INAINTARE(x);
(11)                    IF x = t
(12)                       THEN BEGIN
(13)                                MĂRIRE;
(14)                                x := s;
(15)                             END;
(16)                 END
(17)           ELSE INAPOIERE(x);
(18)     END;
(19) END.
```

```
(1)  PROCEDURA INAINTARE(x);
(2)  BEGIN
(3)     p̃(y) := x;
(4)     x := y;
(5)  END;
```

```
(1)  PROCEDURA INAPOIERE(x);
(2)  BEGIN
(3)     d(x) := min{d(y) + 1 | (x, y) ∈ E⁺(x), r(x, y) > 0}
(4)     IF x ≠ s
(5)        THEN x := p̃(x);
(6)  END;
```

```
(1)  PROCEDURA MĂRIRE;
(2)  BEGIN
(3)     se determină DMF D̃ utilizând vectorul predecesor p̃;
(4)     se determină r(D̃) := min{r(x, y) | (x, y) ∈ D̃};
(5)     se execută mărirea de flux;
(6)     se actualizează reţeaua reziduală G̃(f);
```

(7) END;

Algoritmul menţine un drum parţial admisibil şi iterativ execută operaţii de înaintare sau înapoiere de la nodul curent x. Dacă nodul curent x este extremitatea iniţială a unui arc admisibil (x, y) se execută o operaţie de înaintare şi se adaugă arcul (x, y) la drumul parţial admisibil, altfel se execută o operaţie de înapoiere. Repetăm aceste operaţii până când drumul parţial admisibil atinge nodul stoc t la care timp executăm o mărire de flux. Continuăm acest proces până când fluxul devine maxim. Operaţia $d(x) := \min\{d(y) + 1 \mid (x, y) \in E^+(x),\ r(x, y) > 0\}$ din procedura ÎNAPOIERE se numeşte *operaţia de reetichetare.*

Exemplul 6.5. Ilustrăm algoritmul drumului celui mai scurt Ahuja - Orlin pe exemplu dat în figura 6.5. Mai întâi calculăm vectorul etichetelor distanţă e-xacte prin executarea parcurgerii inverse mai întâi în lăţime pornind de la nodul stoc $t = 12$ din reţeaua reziduală iniţială dată în figura 6.5(a). Se obţine $d = (3, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 0)$. Pornim de la nodul sursă $s = 1$ cu un drum admisibil parţial vid. Executăm o operaţie de înaintare şi adăugăm arcul (1,2) la drumul admisibil parţial. Memorăm acest drum utilizând vectorul predecesor $\widetilde{p}$, astfel $\widetilde{p}(2) := 1$. Acum nodul 2 este nodul curent şi algoritmul execută o operaţie de înaintare de la nodul 2. Astfel, adăugăm arcul (2,7) la drumul parţial admisibil, care acum devine 1-2-7. De asemenea $\widetilde{p}(7) := 2$. În continuare algoritmul adaugă arcul (7,12) la drumul admisibil parţial obţinând $\widetilde{D}_1 = (1, 2, 7, 12)$, care este un drum admisibil la nodul stoc. Executăm o mărire de flux cu $r(\widetilde{D}_1) = \min\{r(1, 2), r(2, 7), r(7, 12)\} = \min\{2, 1, 2\} = 1$. Reţeaua reziduală după prima mărire de flux este arătată în figura 6.5.(b).

Fig. 6.5.

Din nou pornim de la nodul sursă $s = 1$ cu un drum admisibil parţial vid. Algoritmul adaugă arcul $(1, 2)$, $\widetilde{p}(2) := 1$ şi nodul 2 devine nodul curent. Nodul 2 nu este extremitatea iniţială a nici unui arc admisibil. Efectuăm o operaţie de înapoiere în cadrul căreia efectuăm operaţia de reetichetare $d(2) := \min\{d(y) + 1 \mid (2, y) \in E^+(2),\ r(2, y) > 0\} = \min\{d(1) + 1\} = 4$, $x := \widetilde{p}(2) = 1$, astfel arcul (1,2) este eliminat din drumul admisibil parţial deoarece după operaţia de reetichetare a nodului 2 arcul (1,2) devine inadmisibil. În iteraţiile următoare, algoritmul identifică drumurile admisibile $\widetilde{D}_2 = (1, 3, 8, 12)$, $\widetilde{D}_3 = (1, 4, 9, 12)$, $\widetilde{D}_4 = (1, 5, 10, 12)$, $\widetilde{D}_5 = (1, 6, 11, 12)$ şi mărim fluxul cu $r(\widetilde{D}_i) = 1$, $i = 2, 3, 4, 5$ pe aceste drumuri. Astfel se ajunge la un flux maxim.

Lema 6.9. *Algoritmul AODS păstrează validitatea etichetelor distanţă după fiecare operaţie de înaintare, mărire de flux sau operaţie de înapoiere. Fiecare operaţie de reetichetare creşte strict eticheta distanţă a nodului curent.*

Demonstraţie. Operaţia de înaintare nu modifică nici capacităţile reziduale nici

etichetele distanţă şi deci păstrează validitatea etichetelor distanţă.
Arătăm că algoritmul menţine valide etichetele distanţă după fiecare mărire de flux sau fiecare operaţie de înapoiere prin aplicarea inducţiei pe numărul de măriri de flux respectiv pe numărul de operaţii de înapoiere.

Mai întâi să presupunem că după k măriri de flux etichetele distanţă sunt valide. Să arătăm că etichetele distanţă rămân valide şi după $k+1$ măriri de flux. Deoarece în procedura mărire nu se fac modificări asupra etichetelor distanţă, mărirea de flux $k+1$ pe un arc (x, y) afectează condiţiile de validitate numai dacă această mărire introduce un nou arc (y, x) cu $r(y, x) > 0$. Deoarece mărirea de flux se face de-a lungul unui drum admisibil rezultă că $d(x) = d(y) + 1$. Deci $d(y) = d(x) - 1 < d(x) + 1$ şi prin urmare condiţia de validitate pentru arcul (y, x) este verificată.

Acum să presupunem că după k operaţii de înapoiere eticheta distanţă $d(x)$ verifică condiţiile de validitate: $d(x) \leq d(y) + 1$. La operaţia de înapoiere $k + 1$ eticheta distanţă $d(x)$ devine $d'(x) := \min\{d(y) + 1 \mid (x, y) \in E^+(x),\ r(x, y) > 0\}$. Deci $d'(x) \leq d(y) + 1$ şi $d'(x)$ verifică condiţiile de validitate. Algoritmul efectuează o operaţie de înapoiere de la nodul curent x dacă nu există arc (x, y) cu $d(x) = d(y) + 1$ şi $r(x, y) > 0$. Rezultă $d(x) < d(y) + 1$ şi prin urmare $d(x) < \min\{d(y) + 1 \mid (x, y) \in E^+(x),\ r(x, y) > 0\} = d'(x)$, relaţie care arată că fiecare operaţie de reetichetare creşte strict eticheta distanţă a nodului curent x. ■

Teorema 6.10. (Teorema de corectitudine a algoritmului) *Algoritmul AODS calculează un flux maxim în reţeaua $G = (N, A, c)$.*
Demonstraţie. Algoritmul Ahuja - Orlin al drumului celui mai scurt se termină când $d(s) \geq n$. Conform Proprietăţii 6.2 rezultă că reţeaua reziduală nu conţine DMF de la nodul sursă s la nodul stoc t. Deci la terminare, algoritmul drumului celui mai scurt Ahuja - Orlin determină un flux maxim. ■

În continuare se va analiza complexitatea algoritmului AODS. Mai întâi descriem o structură de date utilizată pentru a selecta un arc admisibil pornind dintr-un nod dat x. Această structură de date se numeşte *structură de date arc curent*. Reamintim că lista arcelor care au pe x ca extremitate iniţială este $E^+(x) = \{(x, y) \mid (x, y) \in A\}$. Arcele în această listă pot fi aranjate într-o ordine arbitrară, dar ordinea, odată stabilită, rămâne neschimbată pe parcursul execuţiei algoritmului. Arcul $(x, y) \in E^+(x)$ care urmează a fi testat pentru condiţia de admisibilitate se numeşte *arc curent* al nodului x. Iniţial, arcul curent al nodului x este primul arc din lista $E^+(x)$. Dacă arcul curent din $E^+(x)$ este un arc admisibil atunci algoritmul execută o operaţie, altfel algoritmul indică arcul următor din $E^+(x)$ ca arc curent. Algoritmul execută acest proces până când fie găseşte un arc admisibil, fie se ajunge la sfârşitul listei $E^+(x)$ după care execută o operaţie.
Exemplul 6.6. Să considerăm lista arc a nodului 1 din figura 6.6.

În acest caz $E^+(1) = \{(1,2),(1,3),(1,4),(1,5),(1,6)\}$. Iniţial, arcul curent al nodului 1 este arcul (1,2). Algoritmul verifică dacă arcul este admisibil. Deoarece $r(1,2) = 0$ arcul (1,2) este inadmisibil şi indică arcul (1,3) ca arc curent al nodului 1. Arcul (1,3) este de asemenea inadmisibil, deoarece nu se verifică $d(1) = d(3)+1$, astfel arcul curent devine arcul (1,4), care este admisibil. În continuare, arcul (1,4) rămâne arc curent al nodului 1 până când el devine inadmisibil, fie că $r(1,4)$ devine zero fie că $d(4)$ creşte prin operaţia de reetichetare.

Lema 6.11. *Dacă algoritmul AODS reetichetează orice nod de cel mult k ori, timpul necesar pentru determinarea arcelor admisibile şi reetichetarea nodurilor este $O\left(k\sum_N |E^+(x)|\right) = O(km)$.*

Demonstraţie. Timpul necesar pentru determinarea unui arc admisibil din x este cel mult $|E^+(x)|$. Timpul necesar pentru determinarea tuturor arcelor admisibile este cel mult $\sum_N |E^+(x)| = m$. Evident că timpul necesar pentru reetichetarea tuturor nodurilor este tot $O\left(\sum_N |E^+(x)|\right) = O(m)$. Ţinând seama de faptul că orice nod se reetichetează de cel mult k ori rezultă afirmaţia lemei. ■

Fig. 6.6

Lema 6.12 *Dacă algoritmul AODS reetichetează oricare nod de cel mult k ori, algoritmul saturează arcele (adică reduce capacitatea lor reziduală la zero) de cel mult $km/2$ ori.*

Demonstraţie. Presupunem că o mărire de flux saturează arcul (x,y). Deoarece acest arc este admisibil avem

$$d(x) = d(y) + 1 \tag{6.3}$$

Înainte ca acest algoritm să satureze arcul (x,y) din nou, mai întâi trebuie trimis flux de la nodul y la nodul x. La acest timp, etichetele distanţă $d'(x)$ şi $d'(y)$ verifică

$$d'(y) = d'(x) + 1 \tag{6.4}$$

În următoarea saturaţie a arcului (x,y), trebuie să avem

$$d''(x) = d''(y) + 1 \tag{6.5}$$

Utilizând Lema 6.9 se obţine $d''(x) = d''(y) + 1 \geq d'(y) + 1 = d'(x) + 2 \geq d(x) + 2$. Similar se arată că $d''(y) \geq d(y) + 2$. Deci, între două saturaţii consecutive ale arcului (x,y), ambele etichete distanţă $d(x)$ şi $d(y)$ cresc prin cel puţin două unităţi. Deoarece, prin ipoteză, algoritmul creşte fiecare etichetă distanţă de cel mult k ori rezultă că algoritmul poate satura fiecare arc de cel mult $k/2$ ori. Deci, numărul total de saturaţii arc va fi de cel mult $km/2$ ori.■

Lema 6.13. *În execuţia algoritmului AODS numărul operaţiilor de înapoiere este cel mult n^2, iar numărul măririlor de flux este cel mult $mn/2$.*
Demonstraţie. Fiecare operaţie de reetichetare a nodului x creşte valoarea lui $d(x)$ cu cel puţin o unitate. După ce algoritmul a reetichetat nodul x de cel mult n ori, avem $d(x) \geq n$ şi algoritmul nu va mai selecta niciodată nodul x pentru o operaţie de înaintare, deoarece pentru fiecare nod z din drumul admisibil parţial avem $d(z) < d(s) < n$. Astfel algoritmul reetichetează un nod x de cel mult n ori şi numărul total de operaţii de reetichetare este cel mult n^2. Fiecare operaţie de înapoiere reetichetează un nod, astfel numărul operaţiilor de înapoiere este $O(n^2)$. Conform Lemei 6.12 şi a faptului că un nod poate fi reetichetat de cel mult n ori rezultă că algoritmul saturează arcele de cel mult $mn/2$ ori. Deoarece fiecare mărire de flux saturează cel puţin un arc rezultă că numărul măririlor de flux este cel mult $mn/2$.■
Teorema 6.14. (Teorema de complexitate a algoritmului) *Algoritmul AODS are complexitatea $O(mn^2)$.*
Demonstraţie. Conform Lemei 6.11 şi Lemei 6.13 determinarea arcelor admisibile şi reetichetarea nodurilor are complexitatea $O(mn)$. Fiecare mărire de flux are complexitatea $O(n)$. Deci complexitatea măririlor de flux este $O(mn^2)$. Conform Lemei 6.13 numărul operaţiilor de înapoiere este $O(n^2)$. Ţinând seama că numărul măririlor de flux este $O(mn)$, un drum admisibil are lungimea cel mult n şi că numărul operaţiilor de înapoiere este $O(n^2)$ rezultă că numărul operaţiilor de înaintare este $O(mn^2 + n^2)$. Combinarea acestor margini stabileşte teorema.
■

Algoritmul AODS se termină când $d(s) \geq n$. Acest criteriu de terminare este satisfăcător pentru analiza cazului cel mai defavorabil, dar nu este eficient în practică. Algoritmul pierde prea mult timp reetichetând noduri după ce s-a obţinut un flux maxim. Acest lucru se întâmplă deoarece algoritmul nu are un criteriu de testare dacă s-a obţinut un flux maxim. În continuare descriem o tehnică de testare a prezenţei unei tăieturi minime şi deci testarea existenţei unui flux maxim mult înainte ca $d(s) \geq n$. Introducerea acestei tehnici îmbunătăţeşte performanţa practică a algoritmului AODS.

Introducem tabloul unidimensional $q = (q(0), \ldots, q(n-1))$. Valoarea $q(k)$ este numărul de noduri care au etichetele distanţă egale cu k. Algoritmul iniţializează tabloul $q := 0$ şi îl actualizează după ce calculează etichetele distanţă exacte. După fiecare operaţie de reetichetare actualizăm tabloul q în modul următor. Dacă algoritmul măreşte eticheta distanţă a unui nod x de la k_1 la k_2 atunci $q(k_1) := q(k_1) - 1, q(k_2) := q(k_2) + 1$. După aceea testăm dacă $q(k_1) = 0$. Dacă $q(k_1) = 0$, atunci algoritmul se termină. Pentru a justifica acest criteriu fie $X^* = \{x \mid x \in N,\ d(x) > k_1\}$, $\overline{X}^* = \{\overline{x} \mid \overline{x} \in N, d(\overline{x}) < k_1\}$. Evident că $s \in X^*$ şi $t \in \overline{X}^*$. Deci $[X^*, \overline{X}^*]$ este o tăietură $s-t$ şi $d(x) > d(\overline{x})+1,\ (x, \overline{x}) \in (X^*, \overline{X}^*)$.

Rezultă că $r(x,\overline{x}) = 0,\ (x,\overline{x}) \in (X^*, \overline{X}^*)$. Prin urmare $[X^*, \overline{X}^*]$ este o tăietură minimă şi fluxul curent este un flux maxim.

Exemplul 6.7. Ilustrăm această tehnică prin aplicarea ei la exemplul prezentat la ilustrarea algoritmului AODS. Figura 6.7 prezintă reţeaua reziduală imediat după ultima mărire de flux.

Fig. 6.7.

Deşi fluxul este acum un flux maxim şi cu toate că nodul sursă nu este conectat la nodul stoc printr-un drum în reţeaua reziduală, criteriul de terminare $d(1) \geq 12$ este departe de a fi satisfăcut. Se poate verifica faptul că, în continuare, algoritmul va creşte etichetele distanţă ale nodurilor 6,1,2,3,4,5, în această ordine, fiecare prin 2 unităţi. În cele din urmă $d(1) \geq 12$ şi algoritmul se termină. Dacă folosim tabloul q, atunci pentru etichetele distanţă arătate în figura 6.7 avem $q = (1,5,1,1,4,0,0,0,0,0,0,0)$. Când continuăm algoritmul după ultima mărire de flux el construieşte drumul admisibil parţial 1- 6. În continuare el reetichetează nodul 6 şi $d(6) = 2$ creşte la $d(6) = 4$. În acest caz $q(2) = 0$ şi algoritmul se termină.

6.1.6 Algoritmul Dinic al reţelelor stratificate

Mulţi algoritmi pentru problema fluxului maxim măresc fluxul de-a lungul drumurilor celor mai scurte de la nodul sursă s la nodul stoc t din reţeaua reziduală. Dinic (1970) introduce conceptul de *reţea stratificată*. Reţeaua stratificată Dinic $\overline{\widetilde{G}}'(f) = (\overline{\widetilde{N}}, \overline{\widetilde{A}}, \overline{r}')$ în raport cu fluxul f se defineşte cu ajutorul *funcţiei distanţă complementară* $\overline{d}: \widetilde{N} \to \mathcal{N}$, definită în reţeaua reziduală $\widetilde{G}(f) = (\widetilde{N}, \widetilde{A}, r)$.

Definiţia 6.7. Funcţia distanţă complementară $\overline{d}$ se numeşte *validă* dacă satisface următoarele două condiţii:

$$\overline{d}(s) = 0, \tag{6.6}$$

$$\overline{d}(y) \leq \overline{d}(x) + 1,\ (x,y) \in \widetilde{A}, \tag{6.7}$$

Valoarea $\overline{d}(x)$ se numeşte *eticheta distanţă complementară* a nodului x şi condiţiile (6.6), (6.7) se numesc *condiţii de validitate*.

Proprietatea 6.14. *Dacă etichetele distanţă complementară sunt valide, atunci eticheta distanţă complementară $\overline{d}(x)$ este o margine inferioară pentru lungimea drumului cel mai scurt de la nodul s la nodul x în reţeaua reziduală $\widetilde{G}(f)$.*

Demonstraţie. Se demonstrează la fel ca Proprietatea 6.1. ■

Proprietatea 6.15. *Dacă $\overline{d}(t) \geq n$, atunci reţeaua reziduală $\widetilde{G}(f)$ nu conţine drum de la nodul sursă s la nodul stoc t.*

Demonstraţie. Se demonstrează la fel ca Proprietatea 6.2. ■

Definiţia 6.8. Etichetele distanţă complementară se numesc *exacte* dacă pentru fiecare nod $x, \overline{d}(x)$ este egală cu lungimea drumului cel mai scurt de la nodul s la nodul x în reţeaua reziduală $\widetilde{G}(f)$.

Exemplul 6.8. Să considerăm reţeaua reziduală din figura 6.8, în care $s = 1$ şi $t = 4$. Un vector al etichetelor distanţă complementară valide este $\overline{d} = (0, 0, 0, 0)$ şi vectorul etichetelor distanţă complementară exacte este $\overline{d} = (0, 2, 1, 3)$.

Fig. 6.8

Etichetele distanţă complementară exacte se pot determina cu ajutorul algoritmului parcurgerii BF aplicat reţelei reziduale $\widetilde{G}(f)$ plecând de la nodul sursă s. Acest algoritm are complexitatea $O(m)$.

Definiţia 6.9. Un arc (x, y) din reţeaua reziduală $\widetilde{G}(f)$ se numeşte *admisibil* dacă el satisface condiţia $\overline{d}(y) = \overline{d}(x) + 1$, altfel arcul (x, y) se numeşte *inadmisibil.* Un drum de la nodul sursă s la nodul stoc t din reţeaua reziduală $\widetilde{G}(f)$ se numeşte *drum admisibil* dacă conţine numai arce admisibile, altfel se numeşte *inadmisibil.*

Proprietatea 6.16. *În reţeaua reziduală $\widetilde{G}(f)$ un drum admisibil de la nodul sursă s la nodul stoc t este un cel mai scurt drum de mărire a fluxului.*

Demonstraţie. Se demonstrează la fel ca Proprietatea 6.3. ■

Legătura dintre etichetele distanţă exacte $d(x)$ şi etichetele distanţă complementară exacte $\overline{d}(x)$ este

$$d(x) + \overline{d}(x) = d(s) = \overline{d}(t),\ x \in \widetilde{D},\ \widetilde{D} \text{ admisibil.} \tag{6.8}$$

Este evident că eticheta distanţă complementară exactă $\overline{d}(x)$ este o distanţă complementară pentru eticheta distanţă exactă $d(x)$ în drumul cel mai scurt de la nodul sursă s la nodul stoc t. Astfel se justifică denumirile specificate mai sus. În raport cu un flux dat f, reţeaua stratificată Dinic $\overline{\widetilde{G}'}(f) = (\overline{\widetilde{N}'}, \overline{\widetilde{A}'}, \overline{r}')$ se defineşte în modul următor. Se determină etichetele distanţă complementară exacte $\overline{d}(x)$ prin parcurgerea BF a reţelei reziduale $\widetilde{G}(f) = (\widetilde{N}, \widetilde{A}, r)$ pornind de la nodul sursă s. Atunci avem $\overline{\widetilde{N}'} = \widetilde{N},\ \overline{\widetilde{A}'} = \{(x, y) \mid (x, y) \in \widetilde{A},\ \overline{d}(y) = \overline{d}(x) + 1\}, \overline{r}'(x, y) = r(x, y),\ (x, y) \in \overline{\widetilde{A}'}$. Pentru $k = 0, \ldots, \overline{d}(t)$ se defineşte un strat $\overline{\widetilde{N}'}(k) = \{x \mid x \in \widetilde{N},\ \overline{d}(x) = k\}$. Evident că $\overline{\widetilde{N}'}(0) = \{s\}$ şi $\overline{\widetilde{N}'} = \cup\{\overline{\widetilde{N}'}(k) \mid k = 0, \ldots, \overline{d}(t)\}$. Dacă arcul $(x, y) \in \overline{\widetilde{A}'}$ cu $\overline{d}(x) = k$, atunci $x \in \overline{\widetilde{N}'}(k)$ şi $y \in \overline{\widetilde{N}'}(k + 1)$. Este evident că orice drum de la nodul sursă s la nodul stoc t din reţeaua stratificată Dinic $\overline{\widetilde{G}'}(f)$ are lungimea $\overline{d}(t)$.

Definiţia 6.10. Un flux $\overline{f}$ din reţeaua stratificată Dinic $\overline{\widetilde{G}'}(f)$ se numeşte *flux de blocare* dacă orice drum de la nodul sursă s la nodul stoc t din $\overline{\widetilde{G}'}(f)$ are un arc (x, y) cu $\overline{r}'(x, y) = 0$.

Din definiţie rezultă că dacă reţeaua stratificată Dinic $\overline{\widetilde{G}'}(f)$ nu conţine drum de mărire a fluxului, atunci fluxul $\overline{f}$ este un flux de blocare.
Este posibil ca anumite arce şi noduri din reţeaua stratificată Dinic $\overline{\widetilde{G}'}(f)$ să nu aparţină la nici un drum de la nodul sursă s la nodul stoc t. Aceste arce şi noduri pot fi eliminate din reţeaua stratificată Dinic.

Algoritmul reţelelor stratificate Dinic este următorul:

```
(1)   PROGRAM REŢELE-STRATIFICATE-DINIC;
(2)   BEGIN
(3)      f := 0; d̄(t) = 0;
(4)      se determină reţeaua reziduală G̃(f);
(5)      WHILE d̄(t) < n DO
(6)      BEGIN
(7)         REŢEA-STRATIFICATĂ (G̃(f), G̃̄'(f), d̄(t));
(8)         FLUX-BLOCARE (G̃̄'(f), f̄);
(9)         MĂRIRE FLUX (G̃(f), f, f̄);
(10)     END;
(11)  END.
```

```
(1)   PROCEDURA REŢEA-STRATIFICATĂ (G̃(f), G̃̄'(f), d̄(t));
(2)   BEGIN
(3)      se determină etichetele distanţă complementară d̄(x) în G̃(f);
(4)      se construieşte reţeaua stratificată G̃̄'(f);
(5)   END;
```

```
(1)   PROCEDURA FLUX-BLOCARE (G̃̄'(f), f̄);
(2)   BEGIN
(3)      f̄ := 0;
(4)      WHILE există DMF în G̃̄'(f) DO
(5)      BEGIN
(6)         se determină un DMF D̃̄ în G̃̄'(f);
(7)         se determină r(D̃̄) := min{r̄'(x, y) | (x, y) ∈ D̃̄};
(8)         se execută mărirea de flux;
(9)         se elimină arcele (x, y) cu r̄'(x, y) = 0 şi nodurile x cu
            ρ⁻(x) = 0 sau ρ⁺(x) = 0;
(10)     END;
(11)  END;
```

```
(1)   PROCEDURA MĂRIRE-FLUX (G̃(f), f, f̄);
```

(2) BEGIN
(3) $f := f + \overline{f}$;
(4) se actualizează reţeaua reziduală $\widetilde{G}(f)$;
(5) END;

Teorema 6.17. (Teorema de corectitudine a algoritmului) *Algoritmul Dinic al reţelelor stratificate determină un flux maxim în reţeaua* $G = (N, A, c)$.
Demonstraţie. Algoritmul reţelelor stratificate Dinic se termină când $\overline{d}(t) \geq n$. După un număr de iteraţii ale algoritmului condiţia $\overline{d}(t) \geq n$ va fi îndeplinită deoarece din definiţia şi proprietăţile funcţiei $\overline{d}$ rezultă că etichetele $\overline{d}(t)$, calculate pentru reţelele stratificate construite în algoritm, formează un şir strict crescător. Dacă $\overline{d}(t) \geq n$, atunci conform Proprietăţii 6.15, reţeaua reziduală $\widetilde{G}(f)$ nu conţine DMF de la nodul sursă s la nodul stoc t. Deci, la terminare, algoritmul reţelelor stratificate Dinic determină un flux maxim. ■

Teorema 6.18. (Teorema de complexitate a algoritmului). *Algoritmul reţelelor stratificate Dinic are complexitatea* $O(mn^2)$.
Demonstraţie. Procedura REŢEA-STRATIFICATĂ şi procedura MĂRIRE-FLUX au fiecare complexitatea $O(m)$. Determinarea unui DMF D în $\overline{\widetilde{G}'}$ cu parcurgerea DF are complexitatea $O(n)$. Ciclul WHILE din procedura FLUX-BLOCARE se execută de $O(m)$ ori deoarece la fiecare execuţie se elimină cel puţin un arc din $\overline{\widetilde{G}'}$ şi deci după cel mult m eliminări nodul sursă s nu mai este conectat la nodul stoc t. Deci procedura FLUX-BLOCARE are complexitatea $O(mn)$. Rezultă că ciclul WHILE din algoritmul reţelelelor stratificate Dinic are complexitatea $O(mn)$. Acest ciclu se execută de $O(n)$ ori. Prin urmare algoritmul are complexitatea $O(mn^2)$.■

6.1.7 Algoritmul Ahuja - Orlin al reţelelor stratificate

Algoritmul Ahuja - Orlin al reţelelor stratificate se datorează lui Ahuja şi Orlin (1991) şi este o versiune a algoritmului reţelelor stratificate Dinic. În acest caz reţelele stratificate sunt definite cu ajutorul etichetelor distanţă exacte $d(x)$.

În raport cu un flux dat f, reţeaua stratificată $\widetilde{G}'(f)$ se defineşte în modul următor. Se determină etichetele distanţă exacte $d(x)$ prin parcurgerea inversă BF a reţelei reziduale $\widetilde{G}(f)$ pornind de la nodul stoc t. Reţeaua stratificată este $\widetilde{G}'(f) = (\widetilde{N}', \widetilde{A}', r')$ cu $\widetilde{N}' = \widetilde{N}$, $\widetilde{A}' = \{(x, y) \mid (x, y) \in \widetilde{A}, d(x) = d(y) + 1\}$, $r'(x, y) = r(x, y)$, $(x, y) \in \widetilde{A}'$. Pentru fiecare k, $k = 0, \ldots, d(s)$, se defineşte un strat $\widetilde{N}'(k) = \{x \mid x \in \widetilde{N},\ d(x) = k\}$. Este evident că $\widetilde{N}'(0) = \{t\}$ şi $\widetilde{N}' = \cup\{\widetilde{N}'(k) \mid k = 0, \ldots, d(s)\}$. Legătura dintre straturile $\overline{\widetilde{N}'}(\overline{k})$ şi straturile $\widetilde{N}'(k)$ este $\overline{\widetilde{N}'}(\overline{d}(t) - k) = \widetilde{N}'(k)$ sau $\overline{\widetilde{N}'}(\overline{k}) = \widetilde{N}'(d(s) - \overline{k})$.

Exemplul 6.9. Se consideră reţeaua reziduală $\widetilde{G}(f)$ prezentată în figura 6.9.
Fig. 6.9.

În figura 6.10 (a) se prezintă reţeaua stratificată $\widetilde{G}'(f)$ asociată reţelei reziduale $\widetilde{G}(f)$ prezentată în figura 6.9. Deoarece arcele (5,7), (6,7) şi nodurile 5,6 nu aparţin la nici un drum de la nodul $s = 1$ la nodul $t = 7$ se elimină şi se obţine reţeaua stratificată prezentată în figura 6.10(b).
Fig. 6.10.

Evident că un arc $(x, y) \in \widetilde{A}'$ cu $d(x) = k$ are proprietatea că $x \in \widetilde{N}'(k), y \in \widetilde{N}'(k-1)$.

Algoritmul Ahuja - Orlin al reţelelor stratificate (în continuare algoritmul A - O al reţelelor stratificate) se obţine făcând în algoritmul Ahuja - Orlin al drumului celui mai scurt (algoritmul AODS) următoarele modificări:

Modificarea 1. În operaţia înapoiere (x), nu schimbăm eticheta distanţă a nodului x, dar nodul x devine nod blocat. Un nod blocat x nu are drum admisibil la nodul stoc t.
Modificarea 2. Un arc (x, y) îl numim admisibil dacă $d(x) = d(y)+1,\ r(x, y) > 0$ şi nodul y nu este blocat.

Modificarea 3. Când nodul sursă s devine blocat, se execută parcurgere înapoi BF pentru a recalcula etichetele distanţă exacte care definesc o nouă reţea stratificată.

```
(1)   PROGRAM A - O REŢELE STRATIFICATE;
(2)   BEGIN
(3)      f := 0;
(4)      se determină etichetele distanţă exacte d(x) în G̃(f);
(5)      FOR x ∈ Ñ DO B(x) := 1;
(6)      x := s;
(7)      WHILE d(s) < n DO
(8)      BEGIN
(9)         IF B(s) = 1
(10)           THEN
(11)                IF există arc (x, y) admisibil
(12)                THEN BEGIN
(13)                          INAINTARE(x);
(14)                          IF x = t
```

```
(15)                                THEN BEGIN
(16)                                        MĂRIRE;
(17)                                        x := s;
(18)                                      END;
(19)                        END
(20)                ELSE INAPOIERE(x);
(21)            ELSE BEGIN
(22)                    se determină etichetele distanţă exacte d(x) în G̃(f);
(23)                    FOR x ∈ Ñ DO B(x) := 1;
(24)                    x := s;
(25)                  END;
(26)      END;
(27)   END.
(1)    PROCEDURA INAINTARE(x);
(2)    BEGIN
(3)       p̃(y) := x;
(4)       x := y;
(5)    END;
(1)    PROCEDURA INAPOIERE(x);
(2)    BEGIN
(3)       B(x) := 0;
(4)       IF x ≠ s
(5)          THEN x := p̃(x);
(6)    END;

(1)    PROCEDURA MĂRIRE;
(2)    BEGIN
(3)       se determină DMF D̃ utilizând vectorul predecesor p̃;
(4)       se determină r(D̃) = min{r(x, y) | (x, y) ∈ D̃};
(5)       se execută mărirea de flux;
(6)       se actualizează reţeaua reziduală G̃(f);
(7)    END;
```

Algoritmul menţine un drum parţial admisibil în reţeaua stratificată $\widetilde{G}'(f)$ şi iterativ execută operaţii de înaintare sau înapoiere de la nodul curent x. Dacă nodul curent x este extremitatea iniţială a unui arc admisibil (x, y) se execută o operaţie de înaintare şi se adaugă arcul (x, y) la drumul parţial admisibil, altfel se execută o operaţie de înapoiere şi se blochează nodul x prin $B(x) := 0$. Repetăm aceste operaţii până când drumul parţial admisibil atinge nodul stoc t la care timp executăm o mărire de flux. Repetăm acest proces până când se obţine un

flux de blocare în reţeaua stratificată $\widetilde{G}'(f)$, adică până când nodul sursă s devine blocat. În acest caz se recalculează etichetele distanţă d care definesc o nouă reţea stratificată şi toate nodurile x sunt deblocate prin $B(x) := 1$. Se continuă acest proces până când fluxul devine maxim, adică până când $d(s) \geq n$.

Exemplul 6.10 Ilustrăm algoritmul A - O al reţelelor stratificate pe exemplul dat în figura 6.11 (a).

Fig. 6.11.

Iniţial $d := (3, 2, 2, 1, 1, 0)$ şi $B := (1, 1, 1, 1, 1, 1)$. Algoritmul determină DMF: $\widetilde{D}_1 = (1, 2, 4, 6)$ cu $r(\widetilde{D}_1) = 2$, $\widetilde{D}_2 = (1, 2, 5, 6)$ cu $r(\widetilde{D}_2) = 1, \widetilde{D}_3 = (1, 3, 5, 6)$ cu $r(\widetilde{D}_3) = 2$. Mărind fluxul de-a lungul acestor drumuri se obţine reţeaua reziduală din figura 6.11 (b). În continuare algoritmul determină drumul parţial admisibil $\widetilde{D}_4 = (1, 3, 5)$. Nodul 5 nu este extremitatea iniţială a unui arc admisibil şi se execută o operaţie de înapoiere blocându-se nodul 5, deci $B(5) := 0$. Analog $\widetilde{D}_4 = (1, 3), B(3) := 0, \widetilde{D}_4 = (1), B(1) := 0$. Nodul sursă 1 devine nod blocat. Deci s-a obţinut un flux de blocare. Se calculează $d := (4, 3, 3, 1, 2, 0)$ şi $B := (1, 1, 1, 1, 1, 1)$ şi reţeaua reziduală cu noile etichete care definesc o nouă reţea stratificată este arătată în figura 6.11 (c). Algoritmul determină DMF $\widetilde{D}_5 = (1, 3, 5, 4, 6)$ cu $r(\widetilde{D}_5) = 1$. Mărind fluxul pe acest drum se obţine reţeaua reziduală din figura 6.11 (d). În continuare algoritmul determină drumul parţial admisibil $\widetilde{D}_6 = (1, 3)$. Nodul 3 devine nod blocat deci $B(3) := 0$ şi $\widetilde{D}_6 = (1), B(1) := 0$. Din nou s-a obţinut un flux de blocare. Se calculează $d := (\infty, \infty, \infty, 1, \infty, 0), B := (1, 1, 1, 1, 1, 1)$. Reţeaua reziduală cu noile etichete este prezentată în figura 6.11 (e). Deoarece $d(1) \geq n$ algoritmul se opreşte. Fluxul maxim este prezentat în figura 6.11 (f).

Teorema 6.19. (Teorema de corectitudine a algoritmului) *Algoritmul A - O al reţelelor stratificate determină un flux maxim în reţeaua $G = (N, A, c)$.*

Demonstraţie. Algoritmul reţelelor stratificate se termină când $d(s) \geq n$. Conform Proprietăţii 6.2 rezultă că reţeaua reziduală nu conţine DMF de la nodul sursă s la nodul stoc t. Deci la terminare, algoritmul reţelelor stratificate determină un flux maxim.■

În continuare se va analiza complexitatea algoritmului A - O al reţelelor stratificate.

Lema 6.20. *Algoritmul A - O al reţelelor stratificate calculează cel mult n fluxuri de blocare pentru determinarea unui flux maxim.*

Demonstraţie. Când algoritmul calculează etichetele distanţă d, mulţimea arcelor admisibile defineşte o reţea stratificată. Deoarece nu actualizează etichetele distanţă în timpul calculării unui flux de blocare, fiecare drum admisibil din reţeaua stratificată are lungimea $d(s)$. Fluxul de blocare sumarizează toate flu-

xurile de-a lungul drumurilor admisibile de lungime $d(s)$. Deci când algoritmul recalculează etichetele distanţă exacte, care definesc o nouă reţea stratificată, $d(s)$ trebuie să crească strict. Algoritmul se termină când $d(s) \geq n$ ceea ce demonstrează afirmaţia lemei.■

Teorema 6.21. (Teorema de complexitate a algoritmului). *Algoritmul A - O al reţelelor stratificate are complexitatea* $O(mn^2)$.

Demonstraţie. În timpul calculării unui flux de blocare nu se introduc noi arce admisibile deoarece în acest timp algoritmul nu actualizează etichetele distanţă. Fiecare mărire de flux reduce capacitatea reziduală la zero a cel puţin unui arc. Deci algoritmul execută cel mult m măriri de flux pentru calculul unui flux de blocare. Din definiţie rezultă că orice reţea stratificată este aciclică. Determinarea unui DMF în reţeaua stratificată are complexitatea $O(n)$. Rezultă că determinarea unui flux de blocare are complexitatea $O(mn)$. Ţinând seama de Lema 3.16 obţinem că algoritmul are complexitatea $O(mn^2)$.■

6.2 Algoritmi polinomiali cu prefluxuri

6.2.1 Algoritmul preflux generic

Algoritmul preflux generic este obţinut de Goldberg şi Tarjan (1986) şi noţiunea de preflux a fost introdusă de Karzanov (1974) pentru calcularea fluxurilor de blocare.

Un *preflux* este o funcţie $f : A \to \Re^+$ care satisface constrângerile

$$f(N, x) - f(x, N) \geq 0,\ x \in N - \{s, t\} \tag{6.9.a}$$

$$0 \leq f(x, y) \leq c(x, y),\ (x, y) \in A. \tag{6.9.b}$$

Pentru un preflux dat f reţeaua reziduală $\widetilde{G}(f)$ şi capacitatea reziduală r se definesc analog ca pentru fluxuri. *Excesul* fiecărui nod $x \in N$ este

$$e(x) = f(N, x) - f(x, N) \tag{6.10}$$

Pentru fiecare nod $x \in N - \{s\}$ avem $e(x) \geq 0$, nodul sursă s fiind singurul nod cu exces negativ. Un nod $x \in N$ se numeşte *nod activ* dacă $d(x) > 0$ şi $e(x) > 0$. Deoarece întotdeauna $e(s) < 0$ şi $d(t) = 0$ rezultă că nodurile sursă s şi stoc t nu sunt active niciodată. Evident un preflux f este un flux dacă $e(x) = 0$ pentru toate nodurile $x \in N - \{s, t\}$.

Algoritmii cu drumuri de mărire trimit flux de la nodul sursă la nodul stoc de-a lungul unui drum de mărire a fluxului din reţeaua reziduală corespunzător

unui lanţ de mărire a fluxului din reţeaua originară. Această operaţie se descompune în mai multe operaţii elementare de trimitere a fluxului de-a lungul arcelor ce compun drumul. Fiecare operaţie elementară de trimitere a fluxului de-a lungul unui arc se va numi *înaintare.* Algoritmii cu prefluxuri utilizează aceste operaţii elementare (înaintări) şi deci trimit fluxul pe arcele individuale în locul drumurilor de mărire a fluxului.

Prezenţa nodurilor active indică faptul că soluţia este inadmisibilă. De aceea, operaţia de bază a algoritmului preflux înaintare este selectarea unui nod activ şi diminuarea excesului prin înaintarea fluxului la nodurile adiacente nodului activ selectat. Deoarece dorim să trimitem flux de la nodul sursă s la nodul stoc t, înaintăm fluxul de la nodul activ selectat la nodurile care sunt mai apropiate de nodul stoc. Măsurăm apropierea nodurilor de nodul stoc prin etichetele distanţă. De aceea, trimitem flux pe arcele admisibile. Dacă nodul activ curent nu este extremitatea iniţială a unui arc admisibil, îi creştem eticheta distanţă astfel încât să creem cel puţin un arc admisibil. Algoritmul se termină când reţeaua nu conţine nici un nod activ.

```
(1)  PROGRAM PREFLUX GENERIC;
(2)  BEGIN
(3)     INIŢIALIZARE;
(4)     WHILE reţeaua reziduală conţine un nod activ DO
(5)     BEGIN
(6)        se selectează un nod activ x;
(7)        INAINTARE/REETICHETARE(x);
(8)     END;
(9)  END.
```

```
(1)  PROCEDURA INIŢIALIZARE;
(2)  BEGIN
(3)     f := 0;
(4)     se calculează etichetele distanţă exacte d(x) în G̃(f);
(5)     se înaintează u := r(s, y) unităţi de flux pe fiecare arc (s, y) ∈ E⁺(s);
(6)     d(s) := n;
(7)  END;
```

```
(1)  PROCEDURA INAINTARE/REETICHETARE(x);
(2)  BEGIN
(3)     IF reţeaua reziduală conţine arc admisibil (x, y)
(4)       THEN
```

(5) se înaintează $u := \min\{e(x), r(x,y)\}$ unităţi de flux de la x la y
(6) ELSE
(7) $d(x) := \min\{d(y)+1 \mid (x,y) \in E^+(x)$ şi $r(x,y) > 0\}$;
(8) END;

O înaintare de u unităţi de flux de la nodul x la nodul y constă din următoarele operaţii elementare: $e(x) := e(x) - u$, $r(x,y) := r(x,y) - u$, $e(y) := e(y) + u$, $r(y,x) := r(y,x) + u$. Dacă $u = r(x,y)$ atunci înaintarea pe arcul (x,y) se numeşte *saturată*, iar dacă $u = e(x)$ atunci înaintarea pe arcul (x,y) se numeşte *nesaturată*. O înaintare saturată descreşte $r(x,y)$ la zero, iar o înaintare nesaturată descreşte $e(x)$ la zero. Operaţia $d(x) := \min\{d(y)+1 \mid (x,y) \in E^+(x)$ şi $r(x,y) > 0\}$ se numeşte *operaţie de reetichetare*. Scopul operaţiei de reetichetare este de a crea cel puţin un arc admisibil pe care algoritmul poate executa înaintări ulterioare.

Procedura INIŢIALIZARE realizează următoarele:
i) fiecare nod y pentru care $(s,y) \in E^+(s)$ primeşte un exces pozitiv şi astfel algoritmul poate să înceapă prin selectarea unui nod cu exces pozitiv;
ii) deoarece $r(s,y)$ devin zero rezultă că orice arc $(s,y) \in E^+(s)$ este inadmisibil şi stabilind $d(s) = n$ rămâne satisfăcută condiţia de validitate a etichetelor;
iii) stabilind $d(s) = n$, reţeaua reziduală nu va conţine drumuri de la nodul sursă s la nodul stoc t; deoarece etichetele distanţă sunt nedescrescătoare, garantăm că în iteraţiile ulterioare ale algoritmului reţeaua reziduală nu va conţine drum de la s la t niciodată şi astfel niciodată nu vom mai înainta flux de la nodul s.
Exemplul 6.11. Se consideră reţeaua dată în figura 6.12 (a). Figura 6.12 (b) specifică prefluxul după iniţializări.

Fig. 6.12.

Iteraţia 1. Presupunem că algoritmul selectează nodul 2 pentru operaţia de înaintare/reetichetare. Arcul $(2,4)$ este singurul arc admisibil şi algoritmul execută o înaintare de valoare $u := \min\{2,1\} = 1$. Această înaintare este saturată. Figura 6.12 (c) prezintă reţeaua reziduală după această iteraţie.
Iteraţia 2. Presupunem că algoritmul selectează din nou nodul 2. Deoarece nici un arc admisibil nu pleacă din nodul 2, algoritmul execută o operaţie de reetichetare şi $d(2) := \min\{d(3)+1, d(1)+1\} = \min\{2,5\} = 2$. Noua reţea reziduală este aceeaşi ca cea arătată în figura 6.12 (c) cu excepţia că $d(2) = 2$.
Iteraţia 3. Presupunem că algoritmul selectează nodul 3. Arcul $(3,4)$ este singurul arc admisibil care pleacă din nodul 3 şi algoritmul execută o înaintare cu $u = \min\{e(3), r(3,4)\} = \min\{4,5\} = 4$. Această înaintare este nesaturată. Figura 6.12 (d) prezintă reţeaua reziduală după această iteraţie.

Iteraţia 4. Algoritmul selectează nodul 2 şi execută o înaintare nesaturată pe arcul $(2,3)$ cu $u = \min\{1,3\} = 1$, obţinându-se reţeaua reziduală dată în figura 6.12 (e.)
Iteraţia 5. Algoritmul selectează nodul 3 şi execută o înaintare saturată pe arcul $(3,4)$ cu $u = \min\{1,1\} = 1$, obţinându-se reţeaua reziduală prezentată în figura 6.12 (f).

Acum reţeaua nu conţine noduri active şi algoritmul se termină. Valoarea fluxului maxim este $v^* = e(4) = 6$.

Teorema 6.21. (Teorema de corectitudine a algoritmului) *Algoritmul preflux generic determină un flux maxim în reţeaua $G = (N, A, c)$.*
Demonstraţie. Algoritmul se termină când $e(x) = 0,\ x \in N - \{s,t\}$ ceea ce implică faptul că f este un flux. Deoarece $d(s) \geq n$, reţeaua reziduală nu conţine drum de la nodul sursă s la nodul stoc t. Deci fluxul curent este un flux maxim.■

Pentru analiza complexităţii algoritmului preflux generic remarcăm faptul că rămâne valabilă Lema 6.9, deoarece, ca în cazul algoritmului AODS, algoritmul preflux generic înaintează fluxul numai pe arcele admisibile şi reetichetează un nod x numai când nu există arc admisibil care îl are pe x ca extremitate iniţială.
Lema 6.22. *La orice iteraţie a algoritmului preflux generic, fiecare nod activ x este conectat la nodul sursă s printr-un drum de la nodul x la nodul s în reţeaua reziduală.*
Demonstraţie. Notăm că pentru un preflux $f,\ e(s) \leq 0,\ e(x) \geq 0,\ x \in N - \{s\}$. Conform teoremei descompunerii fluxului putem descompune orice preflux f în raport cu reţeaua originară G în fluxuri de-a lungul (1) drumurilor de la nodul s la nodul t, (2) drumurilor de la nodul s la nodurile active x şi (3) circuitelor. Fie x un nod activ relativ la prefluxul f în G. Descompunerea flux a lui f trebuie să conţină un drum D de la nodul s la nodul x, deoarece fluxurile de-a lungul drumurilor de la nodul s la nodul t şi fluxurile în jurul circuitelor nu contribuie la excesul nodului x. Reţeaua reziduală conţine inversul lui D (D cu orientarea fiecărui arc inversă), adică un drum de la nodul x la nodul s.■

Această lemă ne asigură că operaţia de reetichetare calculează minimul elementelor unei mulţimi diferită de mulţimea vidă.
Lema 6.23. *Pentru fiecare nod $x \in N,\ d(x) < 2n$.*
Demonstraţie. Când algoritmul reetichetează ultima dată nodul x, acest nod a avut un exces pozitiv. Deci reţeaua reziduală conţine un drum D de lungime cel mult $n-2$ de la nodul x la nodul s. Fie $D = (x_1, x_2, \ldots, x_k),\ k \leq n-1,\ x_1 = x,\ x_k = s$. Avem $d(x_1) \leq d(x_2) + 1,\ d(x_2) \leq d(x_3) + 1, \ldots, d(x_{k-1}) \leq d(x_k) + 1$ sau $d(x_1) \leq d(x_3) + 2 \leq \ldots \leq d(x_k) + k - 1$. Ţinând seama că $x_1 = x,\ x_k = s,\ k \leq n-1,\ d(s) = n$ avem $d(x) < 2n$. ■
Lema 6.24. *Fiecare etichetă distanţă creşte de cel mult $2n$ ori şi numărul total*

al operaţiilor de reetichetare este cel mult $2n^2$.

Demonstraţie. Fiecare operaţie de reetichetare a nodului x creşte eticheta distanţă $d(x)$ cu cel puţin o unitate şi conform Lemei 6.23 fiecare etichetă distanţă $d(x)$ va creşte de cel mult $2n$ ori. În consecinţă numărul total al operaţiilor de reetichetare este cel mult $2n^2$.■

Lema 6.25. *Algoritmul execută cel mult* mn *înaintări saturate.*

Demonstraţie. Conform Lemei 6.12 dacă algoritmul reetichetează oricare nod de cel mult k ori, algoritmul saturează arcele de cel mult $km/2$ ori. Din Lema 6.23 rezultă că algoritmul execută cel mult mn înaintări saturate.■

Lema 6.26. *Timpul necesar pentru determinarea arcelor admisibile şi reetichetarea nodurilor este* $O(mn)$.

Demonstraţie. Conform Lemei 6.11 dacă algoritmul reetichetează orice nod de cel mult k ori, timpul necesar pentru determinarea arcelor admisibile şi reetichetarea nodurilor este $O(km)$. Ţinând seama de Lema 6.24 rezultă că timpul necesar pentru determinarea arcelor admisibile şi reetichetarea nodurilor este $O(mn)$.■

Lema 6.27. *Algoritmul preflux generic execută* $O(mn^2)$ *înaintări nesaturate.*

Demonstraţie. Pentru demonstraţie utilizăm o funcţie potenţial. Notăm cu N^+ mulţimea nodurilor active. Considerăm funcţia potenţial $P = \sum_{N^+} d(x)$. Deoarece $|N^+| < n$ şi $d(x) < 2n,\ x \in N^+$ valoarea iniţială a lui P după iniţializare este cel mult $2n^2$. La terminarea algoritmului $N^+ = \emptyset$ şi P are valoarea zero. În timpul execuţiei procedurii ÎNAINTARE/REETICHETARE(x), unul din următoarele două cazuri trebuie să apară.

Cazul 1. Reţeaua reziduală nu conţine nici un arc admisibil (x, y). În acest caz $d(x)$ creşte cu $q \geq 1$ unităţi. Funcţia potenţial P creşte cu cel mult q unităţi. Deoarece $d(x) < 2n,\ x \in N$, creşterea totală în P datorată creşterii etichetelor este limitată de $2n^2$.

Cazul 2. Reţeaua reziduală conţine un arc admisibil (x, y). Atunci algoritmul execută o înaintare saturată sau o înaintare nesaturată. O înaintare saturată pe arcul (x, y) poate crea un nou exces la nodul y şi deci creşte numărul nodurilor active cu 1. Rezultă că funcţia potenţial P creşte cu $d(y) < 2n$ per o înaintare saturată şi cu cel mult $2mn^2$ pentru toate înaintările saturate. O înaintare nesaturată pe arcul (x, y) nu creşte numărul de noduri active. Înaintarea nesaturată produce $e(x) = 0$, deci nodul x devine inactiv, dar nodul y poate să devină activ. Deci funcţia potenţial P descreşte prin $d(x)$ şi poate să crească prin $d(y) = d(x) - 1$. În consecinţă, funcţia potenţial P descreşte cu cel puţin 1 unitate per o înaintare nesaturată.

În concluzie valoarea iniţială a lui P este cel mult $2n^2$ şi creşterea maximă posibilă a lui P este $2n^2 + 2mn^2$. Fiecare înaintare nesaturată descreşte P prin cel puţin o unitate şi $P \geq 0$. Deci algoritmul poate executa cel mult $2n^2 + 2n^2 + 2n^2m$, adică $O(mn^2)$ înaintări nesaturate.■

Teorema 6.28. (Teorema de complexitate a algoritmului) *Algoritmul preflux generic are complexitatea* $O(mn^2)$.
Demonstraţie. Algoritmul se termină când mulţimea nodurilor active $N^+ = \emptyset$. Pentru $N^+ = \emptyset$, obţinem $P = 0$. Ţinând cont de Lema 6.27 rezultă afirmaţia teoremei.■
Fie v^* valoarea fluxului maxim f^*. Un preflux f cu $e(t) = v^*$ se numeşte *preflux maxim.* Algoritmul preflux generic stabileşte un preflux maxim cu mult înainte ca el să stabilească un flux maxim ($e(x) = 0,\ x \in N - \{s, t\}$). Operaţiile înaintare/reetichetare ulterioare cresc etichetele distanţă ale nodurilor active până când ele sunt mai mari decât n, astfel ele pot să trimită excesul lor înapoi la nodul sursă a cărui etichetă distanţă este n. O posibilă modificare a algoritmului constă în introducerea mulţimii N' a nodurilor cu proprietatea că reţeaua reziduală nu conţine drum de la orice nod din N' la nodul stoc t.
Iniţial $N' = \{s\}$ şi definim tabloul q ca la algoritmul AODS. Dacă $q(k') = 0$ atunci pentru orice nod x cu $d(x) > k'$ efectuăm $d(x) = n$ şi nodul x îl introducem în N'. Nu efectuăm operaţiile de înaintare/ reetichetare pentru nodurile din N' şi algoritmul se termină când $e(x) = 0,\ x \in \overline{N}' = N - N' - \{t\}$. La terminare, prefluxul curent f este un preflux maxim şi-l convertim într-un flux maxim.

6.2.2 Algoritmul preflux FIFO

Algoritmul preflux FIFO este obţinut de Goldberg(1985). Mai întâi definim conceptul de analizare a unui nod.

Algoritmul preflux generic, într-o iteraţie, selectează un nod activ x şi efectuează fie o înaintare saturată, fie o înaintare nesaturată, fie o reetichetare a nodului x. Dacă se efectuează o înaintare saturată atunci nodul x poate să rămână nod activ. Algoritmul preflux generic poate selecta, în iteraţia următoare, alt nod activ $y \neq x$. Impunem regula că algoritmul selectează un nod x consecutiv până când $e(x) = 0$ sau nodul x este reetichetat. În consecinţă, algoritmul preflux generic poate executa de la un nod x mai multe înaintări saturate urmate fie de o înaintare nesaturată ($e(x)$ devine zero), fie de o operaţie de reetichetare (nu există arc admisibil (x, y)). Numim această secvenţă de operaţii *analizarea nodului* x. În continuare presupunem că toţi algoritmii cu preflux adoptă această regulă de analizare a nodurilor.

Notăm L *lista nodurilor active.* Algoritmul preflux FIFO analizează nodurile active în ordinea primul intrat, primul ieşit (FIFO). Lista nodurilor active L este organizată, sub aspectul unei structuri de date, ca o *coadă.* Deci algoritmul preflux FIFO selectează nodul x din capul listei L, execută o înaintare de la acest nod şi adaugă noul nod activ y la urma listei L. Algoritmul selectează nodul x până când fie devine inactiv ($e(x) = 0$), fie este reetichetat. În ultimul caz, se adaugă nodul x la urma listei L. Algoritmul preflux FIFO se termină când coada

nodurilor active L este vidă.

```
(1)   PROGRAM PREFLUX-FIFO;
(2)   BEGIN
(3)      INIŢIALIZARE;
(4)      WHILE L ≠ ∅ DO
(5)      BEGIN
(6)         se scoate primul nod x din L;
(7)         INAINTARE/REETICHETARE(x);
(8)      END;
(9)   END.
```

```
(1)   PROCEDURA INIŢIALIZARE;
(2)   BEGIN
(3)      f := 0; L := ∅;
(4)      se calculează etichetele distanţă exacte d(x);
(5)      FOR (s, y) ∈ E+(s) DO
(6)      BEGIN
(7)         se înaintează r(s, y) unităţi de flux pe arcul (s, y);
(8)         IF e(y) > 0 şi y ≠ t
(9)            THEN se adaugă y la urma listei L;
(10)     END;
(11)     d(s) := n;
(12)  END;
```

```
(1)   PROCEDURA INAINTARE/REETICHETARE(x);
(2)   BEGIN
(3)      se selectează primul arc (x, y) din E+(x);
(4)      B := 1;
(5)      REPEAT
(6)         IF (x, y) este arc admisibil
(7)            THEN BEGIN
(8)                    se înaintează u := min{e(x), r(x, y)} unităţi de
                       flux de la x la y;
(9)                    IF y ∉ L şi y ≠ s, t
(10)                      THEN se adaugă y la urma listei L;
(11)                END;
(12)        IF e(x) > 0
(13)           THEN IF (x, y) nu este ultimul arc din E+(x)
(14)                   THEN se selectează arcul următor (x, y) din E+(x)
```

(15) ELSE BEGIN
(16) $d(x) := \min\{d(y) + 1 \mid (x, y) \in E^+(x)$ şi $r(x, y) > 0\}$;
(17) $B := 0$;
(18) END;
(19) UNTIL $e(x) = 0$ sau $B = 0$;
(20) IF $e(x) > 0$
(21) THEN se adaugă x la urma listei L;
(22) END;

Exemplul 6.12. Ilustrăm algoritmul preflux FIFO utilizând exemplul arătat în figura 6.13.

Se consideră reţeaua din figura 6.13 (a). După execuţia procedurii INIŢIALIZARE se obţine reţeaua prezentată în figura 6.13 (b). Algoritmul scoate nodul 2 din coada L şi-l analizează. Presupunem că el execută o înaintare saturată de 5 unităţi pe arcul $(2, 4)$ şi o înaintare nesaturată de 5 unităţi pe arcul $(2, 5)$. După analizarea nodului 2 se obţine reţeaua prezentată în figura 6.13 (c). Algoritmul în continuare scoate nodul 3 din coada L şi-l analizează. Algoritmul execută o înaintare saturată de 5 unităţi pe arcul $(3, 5)$, urmată de o operaţie de reetichetare a nodului 3. După analizarea nodului 3 se obţine reţeaua prezentată în figura 6.13 (d). În final se obţin reţelele din figura 6.13 (e) şi (f).

Fig. 6.13.

Corectitudinea algoritmului preflux FIFO rezultă din corectitudinea algoritmului preflux generic.

Pentru a analiza complexitatea algoritmului preflux FIFO, partiţionăm numărul total de examinări nod în *faze*. Prima fază constă din analizările nod pentru nodurile din coada L obţinută imediat după execuţia procedurii INIŢIALIZARE. Faza a doua constă din analizările nod ale nodurilor care sunt în coada L după ce algoritmul a analizat nodurile din prima fază. Similar, a treia fază constă din analizările nod ale nodurilor care sunt în coada L după ce algoritmul a analizat nodurile din a doua fază şi aşa mai departe. De exemplu, în ilustrarea precedentă, prima fază constă din analizările nod ale listei $L = \{2, 3\}$ şi a doua fază constă din analizările nod ale listei $L = \{4, 5, 3\}$. Se observă că algoritmul analizează orice nod cel mult o dată în timpul unei faze.

Teorema 6.29. (Teorema de complexitate a algoritmului.) *Algoritmul preflux FIFO are complexitatea* $O(n^3)$.

Demonstraţie. Considerăm funcţia potenţial $P = \max\{d(x) \mid x \in L\}$ şi schimbarea totală a lui P în timpul unei faze, adică diferenţa dintre valorile iniţială şi finală ale funcţiei potenţial în timpul fazei. Considerăm două cazuri.

Cazul 1. Algoritmul execută cel puţin o operaţie de reetichetare în timpul unei faze. Atunci P poate să crească cu cel mult creşterea maximă a etichetelor distanţă. Conform Lemei 6.24 rezultă faptul că P poate să crească de cel mult $2n^2$ ori.
Cazul 2. Algoritmul nu execută nici o operaţie de reetichetare pe durata unei faze. În acest caz excesul fiecărui nod x care a fost activ la începutul fazei se transferă la noduri y cu $d(y) = d(x) - 1$. În consecinţă P descreşte prin cel puţin o unitate.
Iniţial $P < n$ şi combinând cazurile 1 şi 2, rezultă că numărul total de faze este cel mult $2n^2 + n$, deoarece la terminarea algoritmului $L = \emptyset$, deci $P = 0$. Fiecare fază analizează orice nod cel mult o dată şi fiecare analizare nod execută cel mult o înaintare nesaturată. Deci o margine de $2n^2 + n$ pe numărul total de faze va implica o margine de $O(n^3)$ pe numărul de înaintări nesaturate. Deoarece numărul de înaintări nesaturate a implicat complexitatea algoritmului preflux generic, rezultă că algoritmul preflux FIFO are complexitatea $O(n^3)$.■

6.2.3 Algoritmul preflux cu eticheta cea mai mare

Algoritmul preflux cu eticheta cea mai mare este obţinut de Goldberg şi Tarjan (1986) şi întotdeauna înaintează fluxul de la un nod activ x cu eticheta distanţă cea mai mare. Fie $p = \max\{d(x) \mid x \text{ este nod activ }\}$. Algoritmul analizează mai întâi nodurile cu etichetele distanţă egale cu p şi înaintează fluxul la nodurile cu etichetele distanţă egale cu $p-1$, şi aceste noduri, la rândul lor, înaintează fluxul la nodurile cu etichetele distanţă egale cu $p-2$, şi aşa mai departe, până când fie algoritmul reetichetează un nod, fie a epuizat toate nodurile active. Când a fost reetichetat un nod, algoritmul repetă acelaşi proces. Corectitudinea algoritmului preflux cu eticheta cea mai mare rezultă din corectitudinea algoritmului preflux generic.

Pentru selectarea unui nod cu eticheta distanţă cea mai mare utilizăm următoarea structură de date:

$$L(i) := \{x \mid x \text{ este nod activ şi } d(x) = i\}$$

sub forma unei stive înlănţuite sau a unei cozi înlănţuite. Deoarece conform Lemei 4.3 $d(x) < 2n,\ x \in N$, rezultă faptul că variabila i trebuie să aibă valorile $i = 1, 2, \ldots, 2n-1$. Definim o *variabilă nivel* h care este o margine superioară pentru valoarea cea mai mare a lui i pentru care $L(i) \neq \emptyset$. Fie $h = k$ şi selectăm orice nod din $L(k)$. Dacă eticheta distanţă a unui nod x creşte de la $d(x)$ la $d'(x)$ în timpul analizării stabilim $h = d'(x)$. Conform Lemei 4.4 variabila nivel h creşte de cel mult $2n^2$ ori. Deoarece iniţial $h < n$ rezultă că h descreşte de cel mult $2n^2 + n$ ori.

Algoritmul preflux cu eticheta cea mai mare este următorul:

```
(1) PROGRAM PREFLUX-ETICH;
(2) BEGIN
(3)     INIŢIALIZARE;
(4)     WHILE L ≠ ∅ DO
(5)     BEGIN
(6)         se scoate din L nodul x cu prioritatea cea mai mare;
(7)         INAINTARE/REETICHETARE(x);
(8)     END;
(9) END.
(1) PROCEDURA INIŢIALIZARE;
(2) BEGIN
(3)     f := 0;
(4)     L := ∅;
(5)     se calculează etichetele exacte d(·);
(6)     FOR (s, y) ∈ E⁺(s) DO
(7)     BEGIN
(8)         se înaintează r(s, y) unităţi de flux pe arcul (s, y);
(9)         IF e(y) < 0 şi y ≠ t
(10)            THEN se adaugă y în coada L având prioritatea d(y);
(11)    END;
(12)    d(s) := n;
(13) END;
```

```
(1) PROCEDURA ÎNAINTARE/REETICHETARE(x);
(2) BEGIN
(3)     se consideră că primul arc din Ẽ⁺(x) este arcul curent;
(4)     B := 1;
(5)     REPEAT
(6)         fie (x, y) arcul curent din Ẽ⁺(x);
(7)         IF (x, y) este arc admisibil
(8)         THEN BEGIN
(9)             se înaintează u := min{e(x), r(x, y)} unităţi de flux de la x la y;
(10)            IF y ∉ L şi y ≠ s, t
(11)                THEN se adaugă y în coada L având prioritatea d(y);
(12)            END;
(13)        IF e(x) > 0
(14)            THEN IF (x, y) nu este ultimul arc din Ẽ⁺(x)
(15)                THEN se consideră că următorul arc din Ẽ⁺(x) este arcul
```

curent;
(16) ELSE BEGIN
(17) $d(x) := \min\{d(y) + 1 \mid (x, y) \in \tilde{E}^+(x)\}$;
(18) $B := 0$;
(19) END;
(20) UNTIL $e(x) = 0$ sau $B = 0$;
(21) IF $e(x) > 0$
(22) THEN se adaugă x în coada L cu prioritatea $d(x)$;
(23) END;

Teorema 6.30. (Prima teoremă de complexitate a algoritmului.) *Algoritmul preflux cu eticheta cea mai mare are complexitatea* $O(n^3$.)
Demonstraţie. Conform precizărilor de mai sus variabila nivel h descreşte de cel mult $2n^2 + n$ ori. Deci se obţine o margine de $O(n^3)$ pe numărul de analizări nod. Deoarece fiecare analizare nod necesită cel mult o înaintare nesaturată, algoritmul execută $O(n^3)$ înaintări nesaturate. Deci algoritmul are complexitatea $O(n^3)$.■

Algoritmul preflux cu eticheta cea mai mare este considerat din punct de vedere practic metoda cea mai eficientă pentru rezolvarea problemei fluxului maxim deoarece execută cel mai mic număr de înaintări nesaturate.
Exemplul 6.13. Se consideră exemplul prezentat în figura 6.14.

Fig. 6.14.

Reţeaua iniţială este prezentată în figura 6.14 (a). După execuţia procedurii INIŢIALIZARE nodurile $2, 3, \ldots, n-1$ au un exces de 1 unitate fiecare. Algoritmul preflux cu eticheta cea mai mare analizează nodurile $2, 3, \ldots, n-1$ în această ordine şi înaintează excesul la nodul stoc n. În contrast, algoritmul preflux FIFO poate executa mult mai multe înaintări. Să presupunem că după execuţia procedurii INIŢIALIZARE coada nodurilor active este $L = \{n-1, n-2, \ldots, 3, 2\}$. Soluţia este descrisă în figura 6.14 (b). Algoritmul va analiza fiecare din aceste noduri în prima fază şi va obţine soluţia descrisă în figura 6.14 (c). După această fază $L = \{n-1, n-2, \ldots, 4, 3\}$. Se poate arăta că algoritmul preflux FIFO va executa în total $n-2$ faze şi utilizează $(n-2)+(n-3)+\ldots+1 = \Omega(n^2)$ înaintări nesaturate.

Din exemplul prezentat rezultă că algoritmul preflux cu eticheta cea mai mare evită înaintări repetate pe arce transportând o cantitate mică de flux. Această caracteristică a algoritmului preflux cu eticheta cea mai mare conduce la o margine mai strânsă pe numărul de înaintări nesaturate. Acest algoritm execută de fapt $O\left(n^2 m^{\frac{1}{2}}\right)$ înaintări nesaturate. Astfel avem următoarea teoremă.

Teorema 6.31. (A doua teoremă de complexitate a algoritmului.) *Algoritmul preflux cu eticheta cea mai mare are complexitatea* $O\left(n^2 m^{\frac{1}{2}}\right)$.
Demonstraţie. Pentru demonstraţie se recomandă cititorului monografiile precizate la bibliografie.■

Algoritmul preflux cu eticheta cea mai mare poate fi implementat uşor dacă în algoritmul FIFO, lista L a nodurilor active o structurăm nu ca o coadă obişnuită, ci ca o coadă prioritară cu prioritatea funcţia distanţă d.

6.2.4 Algoritmul de scalare a excesului

Acest algoritm este o variantă îmbunătăţită a algoritmului preflux generic şi se datorează lui Ahuja şi Orlin (1989). În algoritmul generic cu prefluxuri cele mai mari consumatoare de timp sunt înaintările nesaturate de flux $(O(n^2m))$. În algoritmul de scalare a excesului, numărul acestora se reduce la $O\left(n^2 \log \overline{c}\right)$ prin folosirea tehnicii scalării, adică prin impunerea condiţiei ca să se efectueze doar înaintări nesaturate de cantităţi suficient de mari de flux.

Fie $\overline{r}$ o margine superioară pentru $e_{\max} = \max\{e(x) \mid x \text{ este nod activ}\}$. Spunem că un nod x are un *exces mare* dacă $e(x) \geq \overline{r}/2$; în caz contrar nodul x are un *exces mic.*

Algoritmul de scalare a excesului înaintează întotdeauna flux de la un nod cu exces mare la un nod cu exces mic pentru ca niciun nod să nu acumuleze exces mai mare decât $\overline{r}$.

Algoritmul de scalare a excesului este următorul:

```
(1) PROGRAM SCALARE EXCES;
(2) BEGIN
(3)     INIŢIALIZARE;
(4)     r := 2^⌈log c⌉;
(5)     WHILE r ≥ 1 DO
(6)     BEGIN
(7)         WHILE în reţeaua reziduală există un nod cu exces mare DO
(8)         BEGIN
(9)             dintre nodurile cu exces mare se selectează nodul x care are cea
                mai mică etichetă distanţă;
(10)            ÎNAINTARE/REETICHETARE(x);
(11)        END;
(12)        r := r/2;
(13)    END;
(14) END
```

PROCEDURA INIŢIALIZARE este aceeaşi ca la algoritmul preflux generic.

```
(1) PROCEDURA ÎNAINTARE/REETICHETARE(x);
```

(2) BEGIN
(3) IF în reţeaua reziduală $\tilde{G}(f)$ există un arc admisibil (x, y)
(4) THEN
(5) se înaintează $u := \min\{e(x), r(x, y), \overline{r} - e(y)\}$ unităţi de flux de la x la y;
(6) ELSE
(7) $d(x) := \min\{d(y) + 1 \mid (x, y) \in \tilde{E}^+(x)\}$
(8) END;

Numim *fază de scalare* o fază a algoritmului pe parcursul căreia $\overline{r}$ rămâne constant. O fază de scalare pentru o valoare dată a lui $\overline{r}$ se numeşte *fază de* $\overline{r}-$ *scalare*.

Exemplul 6.14. Se consideră reţeaua din figura 6.15(a). În figura 6.15(b) este reprezentată reţeaua reziduală obţinută după executarea PROCEDURII INIŢIALIZARE. Se efectuează $\overline{r} = 8$. Singurul nod cu exces mare este nodul 3, care este selectat de algoritm şi de la care se înaintează $g := \min\{e(3), r(3, 4), \overline{r} - e(4)\} = \min\{4, 5, 8\} = 4$ unităţi de flux pe arcul admisibil $(3, 4)$. În reţeaua reziduală astfel obţinută care este reprezentată în figura 6.15(c) nu mai există noduri cu excese mari. Se înjumătăţeşte valoarea lui $\overline{r}$ şi se începe o nouă fază de scalare pentru $\overline{r} = 4$. Se selectează nodul 2 care este unicul nod cu exces mare şi se înaintează o unitate de flux pe arcul admisibil $(2, 4)$, obţinându-se reţeaua din figura 6.15(d). În această reţea nu mai există noduri cu exces mare, se efectuează $\overline{r} := 2$ şi se începe o nouă fază de scalare. Există un singur nod cu exces mare, nodul 2, care va fi selectat, reetichetat $d(2) = 2$ şi apoi selectat din nou. Se înaintează o unitate de flux pe arcul admisibil $(2, 3)$ şi se obţine reţeaua reziduală din figura 6.15(e), în care există un singur nod cu exces mare: nodul 3. Acesta este selectat, se înaintează o unitate de flux pe arcul admisibil $(3, 4)$ şi se obţine reţeaua reziduală din figura 6.15(f) în care nu mai există noduri cu exces mare. Se efectuează $\overline{r} := 1$ şi, deoarece nu mai există noduri active, algoritmul se termină. În figura 6.15(g) este reprezentat fluxul maxim obţinut prin aplicarea algoritmului de scalare a excesului.

Fig. 6.15.

Teorema 6.32. *Algoritmul de scalare a excesului determină un flux maxim în reţeaua* $G = (N, A, c)$.

Demonstraţie. La începutul algoritmului $\overline{r} := 2^{\lceil \log \overline{c} \rceil}$, deci $\overline{c} \leq \overline{r} \leq 2\overline{c}$. Pe parcursul fazei de $\overline{r}-$ scalare, $e_{\max}$ poate să crească şi să descrească, dar trebuie să fie îndeplinite condiţiile $\overline{r}/2 < e_{\max} \leq \overline{r}$. Când $e_{\max} \leq \overline{r}/2$ atunci se înjumătăţeşte valoarea lui $\overline{r}$ şi se începe o nouă fază de scalare. După $\lceil \log \overline{c} \rceil + 1$ faze de scalare,

$e_{\max}$ devine nul şi fluxul astfel obţinut este un flux maxim. ■

Teorema 6.33. *Pe parcursul fiecărui faze de $\overline{r}-$ scalare sunt îndeplinite următoarele două condiţii:*
(a) *la fiecare înaintare nesaturată se înaintează cel puţin $\overline{r}/2$ unităţi de flux*
(b) $e_{\max} \leq \overline{r}$.

Demonstraţie. (a) Considerăm o înaintare nesaturată pe un arc oarecare (x, y). Deoarece arcul (x, y) este un arc admisibil, avem: $d(y) < d(x)$. Dar, nodul x a fost selectat ca fiind nodul cu cea mai mică etichetă distanţă dintre toate nodurile cu exces mare. Deci, $e(x) \geq \overline{r}/2$ şi $e(y) < \overline{r}/2$. Deoarece pe arcul (x, y) se efectuează o înaintare nesaturată, rezultă că se măreşte fluxul pe arcul (x, y) cu $\min\{e(x), \overline{r} - e(y)\} \geq \overline{r}/2$ unităţi.

(b) O înaintare de flux pe un arc oarecare (x, y) măreşte doar excesul nodului y. După înaintare, noul exces al nodului y va fi $e(y)+\min\{e(x), r(x, y), \overline{r}-e(y)\} \leq \overline{r}$. Deci, $e_{\max} \leq \overline{r}$. ■

Teorema 6.34. *Pe parcursul fiecărei faze de scalare, algoritmul de scalare a excesului efectuează $O(n^2)$ înaintări nesaturate.*

Demonstraţie. Considerăm funcţia potenţial $P = \sum_{x \in N} e(x)d(x)/\overline{r}$. Valoarea iniţială a lui P la începutul fazei de $\overline{r}-$ scalare este mărginită superior de $2n^2$ deoarece $e(x) \leq \overline{r}$ şi $d(x) \leq 2n$, $x \in N$, conform Teoremei 6.33(b) şi Lemei 6.23. Când algoritmul apelează PROCEDURA ÎNAINTARE/ REETICHETARE(x) putem avea unul din următoarele două cazuri:
Cazul 1. Nu există arc admisibil incident către exterior cu nodul x. În acest caz eticheta distanţă a nodului x creşte cu $q \geq 1$ unităţi, ceea ce duce la creşterea lui P cu cel mult q unităţi pentru că $e(x) \leq \overline{r}$. Deoarece, pentru fiecare nod x, $d(x)$ poate creşte cu cel mult $2n$ unităţi pe parcursul execuţiei algoritmului, rezultă că valoarea lui P creşte cu cel mult $2n^2$ unităţi din cauza reetichetării nodurilor.
Cazul 2. Există un arc admisibil incident către exterior cu nodul x, fie acesta (x, y), pe care algoritmul efectuează o înaintare saturată sau nesaturată. În ambele cazuri P descreşte. După o înaintare nesaturată pe arcul (x, y), fluxul de la nodul x la nodul y creşte, conform Teoremei 6.33(a) cu cel puţin $\overline{r}/2$ unităţi, deci P descreşte cu cel puţin $1/2$ unităţi pentru că $d(y) = d(x) - 1$. Deoarece valoarea iniţială a lui P la începutul fazei de $\overline{r}-$ scalare era cel mult $2n^2$ şi putea să crească cu încă cel mult $2n^2$ unităţi, rezultă că acest caz poate să apară de cel mult $8n^2$ ori. Adică, pe parcursul fiecărei faze de scalare, algoritmul de scalare a excesului efectuează $O(n^2)$ înaintări nesaturate. ■

Teorema 6.35 *Complexitatea algoritmului de scalare a excesului este $O(nm + n^2 \log \overline{c})$.*

Demonstraţie. Conform Teoremei 6.34 rezultă că numărul total de înaintări nesaturate este $O(n^2 \log \overline{c})$ deoarece algoritmul efectuează $O(\log \overline{c})$ faze de scalare. Celelalte operaţii (înaintări saturate, reetichetări de noduri şi determinări de arce

admisibile) au complexitatea $O(nm)$, ca în algoritmul preflux generic. Deci, complexitatea algoritmului de scalare a excesului este $O(nm + n^2 \log \overline{c})$. ■

6.3 Aplicaţii şi comentarii bibliografice

6.3.1 Problema reprezentanţilor

Un oraş are q locuitori, q' asociaţii şi r partide politice. Fiecare locuitor face parte din cel puţin o asociaţie şi poate să fie membru doar la un singur partid. Fiecare asociaţie trebuie să-şi desemneze un membru care să-l reprezinte la consiliul de guvernare al oraşului astfel încât numărul membrilor din consiliu care fac parte din partidul p_k este cel mult u_k. Dacă există un astfel de consiliu atunci se spune că acesta este un *consiliu echilibrat*.

Formulăm această problemă ca o problemă de flux maxim într-o reţea $G = (N, A, c)$. Mulţimea nodurilor este $N = N_1 \cup N_2 \cup N_3 \cup N_4$ unde $N_1 = \{s, t\}$, s este nodul sursă, t este nodul stoc, $N_2 = \{a_1, \ldots, a_{q'}\}$ este mulţimea nodurilor care reprezintă asociaţiile $N_3 = \{l_1, \ldots, l_q\}$ este mulţimea nodurilor care reprezintă locuitorii, $N_4 = \{p_1, \ldots, p_r\}$ este mulţimea nodurilor care reprezintă partidele. Mulţimea arcelor este $A = A_1 \cup A_2 \cup A_3 \cup A_4$, unde $A_1 = \{(s, a_i) \mid a_i \in N_2\}$, $A_2 = \{(a_i, l_j) \mid a_i \in N_2,\ l_j \in N_3$ şi l_j face parte din $a_i\}$, $A_3 = \{(l_j, p_k) \mid l_j \in N_3,\ p_k \in N_4$ şi l_j este membru la $p_k\}$, $A_4 = \{(p_k, t) \mid p_k \in N_4\}$. Funcţia capacitate $c : A \to \mathcal{N}$ este $c(x, y) = 1$ dacă (x, y) este din A_1 sau A_2 sau A_3 şi $c(p_k, t) = u_k,\ (p_k, t) \in A_4$.

Dacă valoarea fluxului maxim în reţeaua $G = (N, A, c)$ este egală cu q', atunci oraşul are un consiliu echilibrat, altfel consiliul este neechilibrat.

Exemplul 6.15. Pentru problema reprezentanţilor considerăm $q = 7, q' = 4, r = 3, u_1 = 1, u_2 = 2, u_3 = 2$. Reţeaua $G = (N, A, c)$ este reprezentată în figura 6.16. Există un flux maxim cu valoarea egală cu $q' = 4$. Deci există un consiliu echilibrat. Soluţia nu este unică.

Fig. 6.16.

6.3.2 Problema rotunjirii matricelor

Se poate rotunji orice număr real r la cel mai mare întreg $\lfloor r \rfloor$ astfel încât $\lfloor r \rfloor \leq r$ sau la cel mai mic întreg $\lceil r \rceil$ astfel încât $r \leq \lceil r \rceil$. Notăm cu $[r]$ dacă r este rotunjit prin $\lfloor r \rfloor$ sau prin $\lceil r \rceil$.

Presupunem că se dă o matrice $p \times q$ de numere reale nenegative $R = (r_{ij})$ cu

$$l_i = \sum_{j=1}^{q} r_{ij},\ i = 1, \ldots, p;\ c_j = \sum_{i=1}^{p} r_{ij},\ j = 1, \ldots, q.$$

Se pune problema determinării matricei rotunjite $[R] = ([r_{ij}])$ şi a sumelor rotunjite $[l_i],\ i = 1, \ldots, p;\ [c_j],\ j = 1, \ldots, q$, astfel încât

$$[l_i] = \sum_{j=1}^{q} [r_{ij}],\ i = 1, \ldots, p;\ [c_j] = \sum_{i=1}^{p} [r_{ij}],\ j = 1, \ldots, q.$$

O astfel de rotunjire se numeşte *rotunjire consistentă*.

Problema rotunjirii consistente a unei matrice se poate modela ca o problemă de flux admisibil într-o reţea $G = (N, A, l, c)$. Mulţimea nodurilor este $N = N_1 \cup N_2 \cup N_3$, unde $N_1 = \{s, t\}$, s este nodul sursă, t este nodul stoc, $N_2 = \{x_1, \ldots, x_p\},\ N_3 = \{y_1, \ldots, y_q\}$. Mulţimea arcelor este $A = A_1 \cup A_2 \cup A_3$, unde $A_1 = \{(s, x_i) \mid x_i \in N_2\},\ A_2 = \{(x_i, y_j) \mid x_i \in N_2,\ y_j \in N_3\},\ A_3 = \{(y_j, t) \mid y_j \in N_3\}$. Funcţiile $l : A \to \mathcal{N},\ c : A \to \mathcal{N}$ sunt definite în modul următor: $l(s, x_i) = \lfloor l_i \rfloor,\ c(s, x_i) = \lceil l_i \rceil,\ i = 1, \ldots, p;\ l(x_i, y_j) = \lfloor r_{ij} \rfloor,\ c(x_i, y_j) = \lceil r_{ij} \rceil,\ i = 1, \ldots, p,\ j = 1, \ldots, q;\ l(y_j, t) = \lfloor c_j \rfloor,\ c(y_j, t) = \lceil c_j \rceil,\ j = 1, \ldots, q$.

Problema rotunjirii matricelor apare în multe aplicaţii. De exemplu, Institutul de Statistică al României utilizează informaţii înregistrate în tabele. Informaţiile conţinute în anumite tabele trebuie protejate. Putem proteja informaţiile dintr-un tabel după cum urmează. Rotunjim fiecare înregistrare din tabel, inclusiv sumele pe linii şi coloane, la un multiplu al unei constante corespunzătoare k, astfel încât înregistrările rotunjite din tabel adunate pe linii şi coloane să dea sumele rotunjite pe linii şi coloane. Această problemă este aceeaşi ca problema rotunjirii matricei discutată mai sus cu excepţia că trebuie să facem rotunjirea fiecărui element la un multiplu de $k \geq 1$ în loc de rotunjirea la un multiplu de 1. Rezolvăm această problemă prin definirea reţelei $G = (N, A, l, c)$ asemănător ca mai sus, dar acum definim $l(x, y)$ şi $c(x, y)$ unde (x, y) este asociat unui număr real r, ca cel mai mare multiplu de k mai mic sau egal cu r şi respectiv cel mai mic multiplu de k mai mare sau egal cu r.

Exemplul 6.16. Să considerăm următoarea matrice

$$\begin{array}{c} \\ R = \end{array} \begin{array}{c} \\ \left[\begin{array}{ccc} 3.1 & 6.8 & 7.3 \\ 9.6 & 2.4 & 0.7 \\ 3.6 & 1.2 & 6.5 \end{array}\right] \\ \\ \end{array} \begin{array}{c} l \\ 17.2 \\ 12.7 \\ 11.3 \end{array}$$

$$c \quad 16.3 \quad 10.4 \quad 14.5$$

Fig. 6.17.

Reţeaua $G = (N, A, l, c)$ pentru cazul $k = 1$ este prezentată în figura 6.17 (a) şi pentru cazul $k = 2$ în figura 6.17 (b).

Rezolvând problema fluxului maxim, în cazul $k = 1$, obţinem

$$[R] = \begin{array}{c} \\ [l] \end{array} \begin{bmatrix} 4 & 6 & 8 \\ 9 & 3 & 1 \\ 4 & 2 & 6 \end{bmatrix} \begin{array}{c} \\ 18 \\ 13 \\ 12 \end{array}$$

$$[c] \quad 17 \quad 11 \quad 15$$

6.3.3 Comentarii bibliografice

Algoritmul de etichetare al lui Ford şi Fulkerson (1956) rulează în timp pseudopolinomial. Edmonds şi Karp (1972) prezintă două implementări în timp polinomial ale acestui algoritm. Prima implementare, care măreşte fluxul de-a lungul drumurilor cu capacitatea reziduală maximă, se execută în $O(m \log \overline{c})$ iteraţii. A doua implementare, care măreşte fluxul de-a lungul drumurilor celor mai scurte, se execută în $O(nm)$ iteraţii şi are complexitatea $O(nm^2)$. Independent, Dinic (1970) introduce un concept al reţelelor drum cel mai scurt (în număr de arce), numite reţele stratificate, şi obţine un algoritm cu complexitatea $O(n^2m)$. Până la acest punct toţi algoritmii fluxului maxim au fost algoritmi bazaţi pe drumuri (lanţuri) de mărire. Karzanov (1974) introduce primul algoritm bazat pe preflux utilizat pentru determinarea unui flux de blocare în reţeaua stratificată; el obţine un algoritm $O(n^3)$. Shiloach şi Vishkin (1982) descriu un alt algoritm $O(n^3)$ bazat pe preflux, care este un precursor al algoritmului preflux FIFO care este descris în paragraful 6.2.

Algoritmul scalării maxime a capacităţii este dat de Ahuja şi Orlin (1992), care este similar cu algoritmul scalării bit dat de Gabow (1985). Algoritmul drumului celui mai scurt de mărire a fluxului dat de Ahuja şi Orlin (1992) poate fi privit ca o variantă a algoritmului lui Dinic (1970) şi utilizează etichetele distanţă în locul reţelelor stratificate.

Cercetătorii au obţinut îmbunătăţiri suplimentare privind timpii de rulare ai algoritmilor pentru fluxul maxim prin utilizarea etichetelor distanţă în locul reţelelor stratificate. Goldberg (1985) introduce primul etichetele distanţă; prin incorporarea lor în algoritmul lui Shiloach şi Vishkin (1982), el obţine implementarea FIFO cu complexitatea $O(n^3)$. Algoritmul preflux generic şi implementarea preflux cu eticheta cea mai mare se datorează lui Goldberg şi Tarjan

(1988). Utilizând ca structură de date un arbore dinamic dezvoltat de Sleator şi Tarjan (1983), Goldberg şi Tarjan (1988) îmbunătăţesc timpul de rulare al implementării FIFO la $O(nm \log(n^2/m))$. Utilizând o analiză ingenioasă, Cheriyan şi Maheshwari (1989) arată că algoritmul preflux cu eticheta cea mai mare are de fapt complexitatea $O\left(n^2 m^{\frac{1}{2}}\right)$. Ahuja şi Orlin (1989) prezintă algoritmul scalării excesului care rulează în $O(nm + n^2 \log \overline{c})$. Ahuja, Orlin şi Tarjan (1989) au îmbunătăţit algoritmul scalării excesului şi au obţinut mai mulţi algoritmi: marginea timp cea mai bună a acestor algoritmi este $O\left(nm \log(n(\log m)^{\frac{1}{2}}/m + 2)\right)$ După 1990 au apărut noi implementări ale principalilor algoritmi pentru fluxul maxim, dar fără a se obţine îmbunătăţiri remarcabile. Recomandăm cititorului să consulte bibliografia acestei cărţi.

Capitolul 7

Fluxuri minime în reţele

7.1 Noţiuni introductive

Problema fluxului minim de la nodul sursă s la nodul stoc t în reţeaua $G = (N, A, \ell, c, s, t)$ constă în a determina funcţia flux $f : A \leftarrow \Re_+$ care îndeplineşte constrângerile:

$$\text{minimizează } v \tag{7.1.a}$$

$$\sum_y f(x,y) - \sum_y f(y,x) = \begin{cases} v, & \text{dacă } x = s \\ 0, & \text{dacă } x \neq s, t \\ -v, & \text{dacă } x = t \end{cases} \tag{7.1.b}$$

$$\ell(x,y) \leq f(x,y) \leq c(x,y), \quad (x,y) \in A. \tag{7.1.c}$$

Pentru problema fluxului minim, un preflux este o functie $f : A \to \Re_+$ care îndeplineşte condiţiile:

$$\sum_y f(x,y) - \sum_y f(y,x) \leq 0, \;\; x \in N \setminus \{s,t\} \tag{7.2.a}$$

$$\ell(x,y) \leq f(x,y) \leq c(x,y). \tag{7.2.b}$$

Fie f un preflux. *Deficitul* fiecărui nod $x \in N$ este definit astfel:

$$e(x) = \sum_y f(x,y) - \sum_y f(y,x). \tag{7.3}$$

Spunem că un nod x este *echilibrat* dacă $e(x) = 0$. Un preflux f care îndeplineşte condiţia

$$e(x) = 0, \ \ x \in N \setminus \{s, t\}$$

este un flux. Deci, un flux este un caz particular de preflux.

Pentru problema fluxului minim, *capacitatea reziduală* a arcului (x, y) corespunzătoare prefluxului f se defineşte astfel

$$\hat{r}(x, y) = c(y, x) - f(y, x) + f(x, y) - \ell(x, y).$$

Reamintim ipoteza 4 din paragraful 5.2 : dacă arcul (x, y) aparţine reţelei atunci şi arcul (y, x) aparţine reţelei. Capacitatea reziduală $\hat{r}(x, y)$ a arcului (x, y) reprezintă cantitatea maximă cu care se poate micşora fluxul de la nodul x la nodul y. Reţeaua reziduală $\hat{G}(f) = (\hat{N}, \hat{A})$ corespunzătoare prefluxului f este formată doar din arcele cu capacităţile reziduale pozitive, adică din arcele (x, y) cu $\hat{r}(x, y) > 0$. În figura 7.1(b) este reprezentată reţeaua reziduală $\hat{G}(f)$ corespunzătoare prefluxului f din reţeaua din figura 7.1(a).

Fig.7.1

Pentru problema fluxului minim, definim capacitatea $\hat{c}[X, \overline{X}]$ a unei tăieturi $s-t$ $[X, \overline{X}]$ ca fiind diferenţa dintre suma marginilor inferioare ale arcelor directe şi suma capacităţilor arcelor inverse, adică

$$\hat{c}[X, \overline{X}] = \ell(X, \overline{X}) - c(\overline{X}, X).$$

O tăietură $s - t$ a cărei capacitate este maximă se numeşte *tăietură maximă*.

Dacă în reţeaua $G = (N, A, \ell, c, s, t)$ f este un flux şi $[X, \overline{X}]$ este o tăietură $s - t$ atunci $f[X, \overline{X}] = f(X, \overline{X}) - f(\overline{X}, X)$ este fluxul net peste această tăietură.

Teorema 7.1. (Teorema valorii fluxului pentru problema fluxului minim). *În reţeaua $G = (N, A, \ell, c, s, t)$ dacă f este un flux de valoare v şi $[X, \overline{X}]$ este o tăietură $s - t$ atunci au loc relaţiile:*

$$v = f[X, \overline{X}] \geq \hat{c}[X, \overline{X}].$$

Demonstraţie. Deoarece f este un flux în reţeaua G, rezultă că f satisface constrângerile de conservare a fluxului (7.1.b). Însumând aceste ecuaţii pentru $x \in X$ se obţine $v = f(X, N) - f(N, X)$. Înlocuind $N = X \cup \overline{X}$ în această relaţie şi dezvoltând rezultă că $v = f(X, X \cup \overline{X}) - f\left(X \cup \overline{X}, X\right) = f(X, X) + f(X, \overline{X}) - f(X, X) - f(\overline{X}, X) = f[X, \overline{X}]$. De asemenea, f satisface constrângerile de mărginire a fluxului (7.1c), deci $f(X, \overline{X}) \geq \ell(X, \overline{X})$ şi $f(\overline{X}, X) \leq c(\overline{X}, X)$. Rezultă că $v = f[X, \overline{X}] = f(X, \overline{X}) - f(\overline{X}, X) \geq \ell(X, \overline{X}) - c(\overline{X}, X) = \hat{c}[X, \overline{X}]$. ■

Teorema 7.2. (Teorema fluxului minim şi a tăieturii maxime). *Dacă există un flux admisibil în reţeaua $G = (N, A, \ell, c, s, t)$ atunci valoarea fluxului minim de la nodul sursă s la nodul stoc t este egală cu capacitatea tăieturii $s - t$ maxime.*

Demonstraţie. Rezultă din Teorema 7.1.

Determinarea unui flux minim într-o reţea $G = (N, A, \ell, c, s, t)$ se face în două etape:

Etapa 1. Se determină un flux admisibil (a se vedea paragraful 5.6).

Etapa 2. Pornind de la un flux admisibil, se determină un flux minim.

În continuare vom prezenta algoritmi pentru Etapa 2.

7.2 Algoritmi pseudopolinomiali pentru fluxul minim

7.2.1 Algoritmul generic

Fie f un flux de valoare v, L un lanţ de la nodul s la nodul t, cu L^+ mulţimea arcelor directe şi L^- mulţimea arcelor inverse, în reţeaua $G = (N, A, \ell, c, s, t)$. Un lanţ L de la nodul sursă s la nodul stoc t se numeşte *lanţ de micşorare a fluxului* (LmF) în raport cu fluxul f dacă $f(x, y) > \ell(x, y)$ pentru $(x, y) \in L^+$ şi $f(y, x) < c(y, x)$ pentru $(y, x) \in L^-$.

Un LmF L în raport cu fluxul f în reţeaua G corespunde unui drum $\hat{D}$ de micşorare a fluxului (DmF) în reţeaua reziduală $\hat{G}(f)$ şi reciproc. Definim capacitatea reziduală a unui LmF L în modul următor:

$$r(L^+) = \min\{f(x, y) - \ell(x, y) \,|\, (x, y) \in L^+\},\ r(L^-) =$$

$$= \min\{c(y, x) - f(y, x) \mid (y, x) \in L^-\},\ r(L) = \min\{r(L^+),\, r(L^-)\}.$$

Evident că $r(L) > 0$. Se poate efectua micşorarea de flux : $f'(x, y) = f(x, y) - r(L)$, $(x, y) \in L^+$, $f'(y, x) = f(y, x) + r(L)$, $(y, x) \in L^-$, $f'(x, y) = f(x, y)$, $(x, y) \notin L$. Evident, fluxul f' are valoarea $v' = v - r(L)$.

Teorema 7.3 (Teorema drumului de micşorare a fluxului). *Un flux f^* este un flux minim dacă şi numai dacă reţeaua reziduală $\hat{G}(f^*)$ nu conţine nici un drum de micşorare a fluxului.*

Demonstraţie. Dacă f^* este un flux minim atunci în reţeaua reziduală $\hat{G}(f^*)$ nu există nici un drum de micşorare a fluxului, altfel f^* nu ar fi un flux minim. Reciproc, dacă reţeaua reziduală $\hat{G}(f^*)$ nu conţine nici un drum de micşorare a fluxului rezultă că nodul t nu este accesibil din nodul s în $\hat{G}(f^*)$. Fie X^* mulţimea nodurilor accesibile din nodul s în $\hat{G}(f^*)$. Rezultă că $[X^*, \overline{X}^*]$ este o

tăietură $s-t$ cu $\hat{c}[X^*, \overline{X}^*] = v^*$, unde v^* este valoarea fluxului f^*. Deci, f^* este un flux minim.■

Cât timp reţeaua reziduală $\hat{G}(f)$ conţine un DmF putem micşora fluxul de la nodul sursă s la nodul stoc t. Algoritmul generic pentru fluxul minim se bazează pe această proprietate.

(1) PROGRAM GENERIC − Fm;
(2) BEGIN
(3) fie f un flux admisibil în G;
(4) se determină reţeaua reziduală $\hat{G}(f)$;
(5) WHILE $\hat{G}(f)$ conţine un DmF DO
(6) BEGIN
(7) se determină un DmF $\hat{D}$;
(8) $\hat{r}(\hat{D}) := \min\{\hat{r}(x,y) \mid (x,y) \in \hat{D}\}$;
(9) se efectuează micşorarea de flux;
(10) se actualizează $\hat{G}(f)$;
(11) END
(12) END.

Algoritmul generic pentru fluxul minim lucrează pe reţeaua reziduală şi, la terminarea lui, se obţin capacităţi reziduale optime. Pornind de la aceste capacităţi reziduale putem determina un flux minim în modul următor: efectuăm schimbările de variabile: $c'(x,y) = c(x,y) - \ell(x,y)$, $\hat{r}'(x,y) = \hat{r}(x,y)$, $f'(x,y) = f(x,y) - \ell(x,y)$, $(x,y) \in A$. Capacitatea reziduală a arcului (x,y) este

$$\hat{r}(x,y) = c(y,x) - f(y,x) + f(x,y) - \ell(x,y)$$

sau, echivalent

$$\hat{r}'(x,y) = c'(y,x) - f'(y,x) + f'(x,y).$$

Analog

$$\hat{r}'(y,x) = c'(x,y) - f'(x,y) + f'(y,x).$$

Putem determina valorile lui f' în mai multe moduri. De exemplu,

$$f'(x,y) = \max\left(\hat{r}'(x,y) - c'(y,x),\, 0\right)$$

şi

$$f'(y,x) = \max\left(\hat{r}'(y,x) - c'(x,y),\, 0\right).$$

Revenind la variabilele initiale, obtinem

$$f(x,y) = \ell(x,y) + \max(\hat{r}\,(x,y) - c(y,x) + \ell(y,x),\, 0) \qquad (7.4.a)$$

şi

$$f(y,x) = \ell(y,x) + \max(\hat{r}\,(y,x) - c(x,y) + \ell(x,y),\, 0). \qquad (7.4.b)$$

Exemplul 7.1. Ilustrăm algoritmul generic pentru fluxul minim pe reţeaua din figura 7.2(a). În figura 7.2(b) este reprezentată reţeaua reziduală corespunzătoare fluxului admisibil ilustrat în figura 7.2(a). Se determină DmF $\hat{D}= (1,2,4)$ a cărui capacitate reziduală este $\hat{r}\,(\hat{D}) = \min\{\hat{r}\,(1,2), \hat{r}\,(2,4)\} = \min\{3,6\} = 3$ şi se micşorează cu trei unităţi fluxul de-a lungul drumului $\hat{D}$. Reţeaua reziduală obţinută după micşorarea fluxului de-a lungul drumului $\hat{D}= (1,2,4)$ este reprezentată în figura 7.2(c). Se determină DmF $\hat{D}= (1,3,2,4)$, $\hat{r}\,(\hat{D}) = \min\{\hat{r}\,(1,3), \hat{r}\,(3,2), \hat{r}\,(2,4)\} = \min\{6,2,3\} = 2$ şi se micşorează cu două unităţi fluxul de-a lungul drumului $\hat{D}= (1,3,2,4)$, obţinându-se reţeaua reziduală ilustrată în figura 7.2(d). Se determină DmF $\hat{D}= (1,3,4)$, $\hat{r}\,(\hat{D}) = \min\{\hat{r}\,(1,3), \hat{r}\,(3,4)\} = \min\{4,4\} = 4$ şi după ce se micşorează cu patru unităţi fluxul de-a lungul drumului $\hat{D}= (1,3,4)$ se obţine reţeaua reziduală reprezentată în figura 7.2(e), în care nu mai există DmF şi algoritmul se termină. Pornind de la capacităţile reziduale optime determinate cu ajutorul algoritmului generic pentru fluxul minim şi folosind relaţiile (7.4) se determină fluxul minim care este ilustrat în figura 7.2(f).

Fig.7.2

Teorema 7.4. (Teorema valorilor întregi). *Dacă funcţia capacitate c şi funcţia margine inferioară ℓ sunt cu valori întregi şi dacă există un flux admisibil atunci algoritmul generic pentru fluxul minim converge într-un număr finit de iteraţii şi la terminarea execuţiei lui se obţine un flux minim cu valori întregi.*
Demonstraţie. Dacă funcţia capacitate c şi funcţia limită inferioară ℓ sunt cu valori întregi atunci fluxul admisibil f care este determinat rezolvând o problemă de flux maxim într-o reţea extinsă (a se vedea paragraful 5.6) are valori întregi, conform Teoremei 5.8. Dacă c, ℓ şi f sunt cu valori întregi atunci capacitatea reziduală $\hat{r}(\hat{D})$ a oricărui DmF $\hat{D}$ va fi întotdeauna o valoare întreagă. Deci, pentru fiecare DmF $\hat{D}$ valoarea fluxului va scădea cu $\hat{r}(\hat{D}) \geq 1$. Rezultă că după un număr finit de iteraţii valoarea fluxului va ajunge la valoarea minimă şi în reţeaua reziduală nu vor mai exista DmF. Deci, algoritmul generic pentru fluxul minim converge într-un număr finit de iteraţii. Deoarece c, ℓ şi f sunt cu valori întregi este evident că după fiecare micşorare de flux, atât fluxul cât şi capacitatea reziduală sunt cu valori întregi. Deci, fluxul minim este cu valori întregi. ■
Teorema 7.5. *Dacă funcţia capacitate c, funcţia margine inferioară ℓ şi fluxul admisibil iniţial f sunt cu valori întregi, atunci complexitatea algoritmului generic pentru fluxul minim este $O(nm\bar{c})$.*
Demonstraţie. La fiecare iteraţie algoritmul efectuează o micşorare de flux în urma căreia valoarea fluxului scade cu cel puţin o unitate. Complexitatea unei micşorări de flux este $O(m)$, iar numărul de micşorări este $O(n\bar{c})$, pentru că valoarea fluxului este cuprinsă între 0 şi $n\bar{c}$. Deci, complexitatea algoritmului generic pentru fluxul minim este $O(nm\bar{c})$. ■

7.2.2 Varianta algoritmului Ford-Fulkerson

Varianta algoritmului Ford-Fulkerson este o versiune îmbunătăţită a algoritmului generic pentru fluxul minim. Acest algoritm determină un DmF $\hat{D}$ în reţeaua reziduală cu ajutorul unui vector n-dimensional numit predecesor şi notat cu $\hat{p}$. Dacă, pe parcursul execuţiei algoritmului, un nod $x \neq s$ are $\hat{p}(x) \neq 0$ atunci înseamnă că s-a identificat un drum $\hat{D}_{sx}$ de la nodul s la nodul x în reţeaua reziduală. În plus, pentru fiecare succesor y al lui x către care nu s-a identificat deocamdată un drum care porneşte din nodul s, se poate determina drumul $\hat{D}_{sy}=\hat{D}_{sx} \cup\{(x,y)\}$ şi se atribuie $\hat{p}(y) = x$. Varianta algoritmului Ford-Fulkerson este următoarea:
(1) PROGRAM FF− Fm;

(2) BEGIN
(3) fie f un flux admisibil în G;
(4) se determină reţeaua reziduală $\hat{G}(f)$;
(5) REPEAT
(6) FOR toate nodurile $y \in \hat{N}$ DO $\hat{p}(y) := 0$;
(7) $\hat{V} := \{s\}$;
(8) WHILE $\hat{V} \neq \emptyset$ şi $\hat{p}(t) = 0$ DO
(9) BEGIN
(10) se scoate un nod x din $\hat{V}$;
(11) FOR fiecare arc (x, y) din $\hat{G}(f)$ DO
(12) IF $\hat{p}(y) = 0$ şi $y \neq s$
(13) THEN BEGIN
(14) $\hat{p}(y) := x$;
(15) $\hat{V} = \hat{V} \cup \{y\}$;
(16) END
(17) END;
(18) IF $\hat{p}(t) \neq 0$ THEN MICŞORARE;
(19) UNTIL $\hat{p}(t) = 0$;
(20) END.

(1) PROCEDURA MICŞORARE;
(2) BEGIN
(3) se determină un DmF $\hat{D}$ cu ajutorul vectorului predecesor $\hat{p}$;
(4) se determină $\hat{r}(\hat{D}) = \min\{\hat{r}(x, y) \mid (x, y) \in \hat{D}\}$;
(5) se efectuează micşorarea de flux de-a lungul drumului $\hat{D}$;
(6) se actualizeaza $\hat{G}(f)$;
(7) END;

Exemplul 7.2. Ilustrăm varianta algoritmului Ford-Fulkerson pe reţeaua din figura 7.3 (a). În figura 7.3(b) este reprezentată reţeaua reziduală corespunzătoare fluxului admisibil ilustrat în figura 7.3(a). La prima iteraţie vectorul predecesor $\hat{p}$ poate fi $(0, 1, 1, 2)$. Rezultă că $\hat{D} = (1, 2, 4)$ şi $\hat{r}(\hat{D}) = 3$. Reţeaua reziduală obţinută după micşorarea fluxului de-a lungul DmF $\hat{D} = (1, 2, 4)$ este reprezentată în figura 7.3(c). La a doua iteraţie vectorul predecesor este $\hat{p} = (0, 3, 1, 3)$, iar DmF este $\hat{D} = (1, 3, 4)$ cu $\hat{r}(\hat{D}) = 4$. Se micşorează cu patru unităţi fluxul

de-a lungul DmF $\hat{D}= (1,3,4)$ şi se obţine reţeaua reziduală din figura 7.3(d). La a treia iteraţie se determină $\hat{p}= (0,3,1,2)$, $\hat{D}= (1,3,2,4)$, $\hat{r}\ (\hat{D}) = 2$ şi se micşorează cu două unităţi fluxul de-a lungul drumului $\hat{D}$, obţinându-se reţeaua reziduală reprezentată în figura 7.3(e). În această reţea, se iniţializează $\hat{V}= \{s\}$, se obţine $\hat{V}= \phi$ şi $\hat{p}\ (t) = 0$ şi algoritmul se termină. Pornind de la capacităţile reziduale optime determinate cu ajutorul variantei algoritmului Ford-Fulkerson şi folosind relaţiile (7.4) se determină fluxul minim care este ilustrat în figura 7.3(f).

Fig.7.3

Teorema 7.6. *Varianta algoritmului Ford-Fulkerson determină un flux minim de la nodul sursă s la nodul stoc t.*
Demonstraţie. Execuţia algoritmului se termină atunci când nodului stoc t nu i se mai poate atribui un predecesor nenul.

Fie $X^* = \{x \in N \mid \hat{p}\ (x) \neq 0\} \cup \{s\}$ şi $\overline{X}^* = N \setminus X^*$. Evident că $\left[X^*, \overline{X}^*\right]$ este o tăietură $s-t$. Deoarece $\hat{p}\ (\overline{x}) = 0$ pentru oricare $\overline{x} \in \overline{X}^*$ rezultă că $\hat{r}\ (x,\overline{x}) = 0$ pentru orice arc $(x,\overline{x}) \in (X^*, \overline{X}^*)$. Dar $\hat{r}\ (x,\overline{x}) = c(\overline{x},x) - f(\overline{x},x) + f(x,\overline{x}) - \ell(x,\overline{x})$, $c(\overline{x},x) - f(\overline{x},x) \geq 0$, $f(x,\overline{x}) - \ell(x,\overline{x}) \geq 0$ şi condiţiile $\hat{r}\ (x,\overline{x}) = 0$ implică faptul că $f(x,\overline{x}) = \ell(x,\overline{x})$ pentru orice arc $(x,\overline{x}) \in (X^*, \overline{X}^*)$ şi $f(\overline{x},x) = c(\overline{x},x)$ pentru orice arc $(\overline{x},x) \in (\overline{X}^*, X^*)$. Rezultă că $v = f\left[X^*, \overline{X}^*\right] = f(X^*, \overline{X}^*) - f(\overline{X}^*, X) = \ell(X^*, \overline{X}^*) - c(\overline{X}^*, X^*) = \hat{c}\ [X^*, \overline{X}^*]$, adică fluxul f de valoare v obţinut la terminarea algoritmului este un flux minim. ■

Teorema 7.7. *Dacă funcţia capacitate c, funcţia margine inferioară ℓ şi fluxul admisibil iniţial f sunt cu valori întregi atunci varianta algoritmului Ford-Fulkerson pentru fluxul minim are complexitatea $O(nm\overline{c})$.*
Demonstraţie. La fiecare iteraţie algoritmul efectuează o micşorare de flux în urma căreia valoarea fluxului scade cu cel puţin o unitate. Cum valoarea fluxului este cuprinsă între 0 şi $n\overline{c}$, rezultă că algoritmul efectuează $O(n\overline{c})$ iteraţii. Deoarece pentru determinarea unui DmF sunt necesare cel mult m examinări de arce, rezultă că complexitatea unei iteraţii este $O(m)$, iar complexitatea algoritmului este $O(nm\overline{c})$. ■

7.3 Algoritmi polinomiali pentru fluxul minim

7.3.1 Noţiuni introductive

Vom defini etichetele distanţa pentru problema fluxului minim.

Definiţia 7.1. Într-o reţea reziduală $\hat{G}(f) = (\hat{N}, \hat{A}, \hat{r})$ o funcţie $d : \hat{N} \rightarrow \mathcal{N}$ se numeşte funcţie *distanţă.* Funcţia distanţă d se numeşte *validă* dacă satisface următoarele două condiţii:

$$d(s) = 0 \tag{7.5}$$

$$d(y) \leq d(x) + 1,\ (x, y) \in \hat{A} \tag{7.6}$$

Valoarea $d(x)$ se numeşte *eticheta distanţă* a nodului x, iar condiţiile (7.5) şi (7.6) se numesc *condiţii de validitate.*

Proprietatea 7.8. (a) *Dacă etichetele distanţă sunt valide, atunci eticheta distanţă $d(x)$ este o margine inferioară pentru lungimea drumului celui mai scurt de la nodul s la nodul x în reţeaua reziduală $\hat{G}(f)$.*

(b) *Dacă $d(t) \geq n$ atunci în reţeaua reziduală $\hat{G}(f)$ nu există drum de la nodul sursă s la nodul stoc t.*

Demonstraţie. (a) Fie $D = (x_1, x_2, \ldots, x_k, x_{k+1})$ cu $x_1 = s$, $x_{k+1} = x$ un drum oarecare de lungime k de la nodul s la nodul x în reţeaua reziduală. Condiţiile de validitate implică faptul că:

$$d(x_2) \leq d(x_1) + 1 = d(s) + 1 = 1$$
$$d(x_3) \leq d(x_2) + 1 \leq 2$$
$$\ldots$$
$$d(x_{k+1}) \leq d(x_k) + 1 \leq k.$$

Deci $d(x) = d(x_{k+1}) \leq k$, ceea ce demonstrează prima parte a proprietăţii.

(b) Eticheta distanţă $d(t)$ este o margine inferioară pentru lungimea drumului celui mai scurt de la nodul s la nodul t în reţeaua reziduală $\hat{G}(f)$. Orice drum elementar de la s la t are lungimea cel mult $n - 1$. Deci, dacă $d(t) \geq n$ atunci în reţeaua reziduală $\hat{G}(f)$ nu există drum de la nodul s la nodul t.

Definiţia 7.2. Etichetele distanţă se numesc *exacte* dacă pentru fiecare nod x, $d(x)$ este egală cu lungimea drumului celui mai scurt de la nodul s la nodul x în reţeaua reziduală $\hat{G}(f)$.

Etichetele distanţă exacte se pot determina cu ajutorul algoritmului de parcurgere BF aplicat reţelei reziduale $\hat{G}(f)$ plecând de la nodul sursă s.

Definiţia 7.3. Un arc (x, y) din reţeaua reziduală $\hat{G}(f)$ se numeşte *admisibil* dacă satisface condiţia $d(y) = d(x) + 1$; în caz contrar arcul (x, y) se numeşte *inadmisibil.* Un drum de la nodul sursă s la nodul stoc t în reţeaua reziduală $\hat{G}(f)$ se numeşte *drum admisibil* dacă este format doar din arce admisibile, altfel

se numeşte *inadmisibil.*

Proprietatea 7.9. *În reţeaua reziduală* $\hat{G}(f)$ *un drum admisibil de la nodul sursă* s *la nodul stoc* t *este un celui mai scurt drum de micşorare a fluxului.*

Demonstraţie. Deoarece fiecare arc (x, y) dintr-un drum admisibil D este admisibil, capacitatea reziduală $\hat{r}(x, y)$ şi etichetele distanţă $d(x)$ şi $d(y)$ îndeplinesc următoarele condiţii: (1) $\hat{r}(x, y) > 0$ şi (2) $d(y) = d(x) + 1$. Condiţia (1) implică faptul că D este un drum de micşorare a fluxului, iar condiţia (2) implică faptul că dacă D conţine k arce atunci $d(t) = k$. Deoarece $d(t)$ este o margine inferioară pentru lungimea drumului celui mai scurt de la nodul sursă s la nodul stoc t în reţeaua reziduală $\hat{G}(f)$, rezultă că D trebuie să fie un celui mai scurt drum de micşorare a fluxului. ■

7.3.2 Algoritmi polinomiali cu drumuri de micşorare

Algoritmii pseudopolinomiali cu drumuri de micşorare prezentaţi în paragraful 7.2 au fost obţinuţi din algoritmii pentru fluxul maxim în modul următor: s-a construit reţeaua reziduală $\hat{G}(f)$ pentru problema fluxului minim în locul reţelei reziduale $\tilde{G}(f)$ pentru problema fluxului maxim, s-au determinat drumuri de la nodul sursă s la nodul stoc t în $\hat{G}(f)$ după aceleaşi reguli după care se determină drumuri de la s la t în $\tilde{G}(f)$ în algoritmii corespunzători pentru fluxul maxim şi s-a micşorat fluxul de-a lungul acestor drumuri. Această transformare poate fi aplicată oricărui algoritm cu drumuri de mărire pentru fluxul maxim, obţinându-se astfel un algoritm similar pentru fluxul minim, care va avea aceeaşi complexitate ca algoritmul pentru fluxul maxim din care a fost obţinut. Algoritmii polinomiali cu drumuri de mărire din paragraful 6.1 sunt prezentaţi în Tabelul 7.1.

Tabelul 7.1

Algoritm polinomial cu drumuri de mărire	Complexitate
Algoritmul Gabow al scalării bit a capacităţii	$O(nm \log \overline{c})$
Algoritmul Ahuja - Orlin al scalării maxime a capacităţii	$O(nm \log \overline{c})$
Algoritmul Edmons - Karp al drumului celui mai scurt	$O(nm^2)$
Algoritmul Ahuja - Orlin al drumului celui mai scurt	$O(n^2m)$
Algoritmul Dinic al reţelelor startificate	$O(n^2m)$
Algoritmul Ahuja - Orlin al reţelelor stratificate	$O(n^2m)$

În continuare vom prezenta, pentru exemplifcare, varianta algoritmului Ahuja - Orlin al drumului celui mai scurt. Acest algoritm foloseşte etichetele de distanţă

exacte şi micşorează fluxul de-a lungul drumurilor admisibile.

Definiţia 7.4 În reţeaua reziduală, un drum de la un nod oarecare x la nodul stoc t format doar din arce admisibile se numeşte drum *parţial admisibil.* Nodul x se numeşte nod curent.

Varianta algoritmului Ahuja - Orlin al drumului celui mai scurt este următoarea:

```
(1) PROGRAM AODS - Fm;
(2) BEGIN
(3)     fie f un flux admisibil în reţeaua G;
(4)     se determină reţeaua reziduală Ĝ (f);
(5)     se calculează etichetele distanţă exacte d(·) în Ĝ (f);
(6)     y := t;
(7)     WHILE d(t) < n DO
(8)     BEGIN
(9)       IF există arc admisibil (x, y)
(10)        THEN BEGIN
(11)                 ÎNAINTARE (x, y);
(12)                 IF y = s
(13)                   THEN BEGIN
(14)                          MICŞORARE;
(15)                          y := t;
(16)                        END;
(17)             END
(18)        ELSE ÎNAPOIERE (y);
(19)    END;
(20) END.
(1) PROCEDURA ÎNAINTARE (x, y);
(2) BEGIN
(3)    û (x) := y
(4)    y := x;
(5) END;
(1) PROCEDURA ÎNAPOIERE (y);
(2) BEGIN
(3)    d(y) := min{d(x) | (x, y) ∈Â} + 1;
(4)    IF y ≠ t
(5)       THEN y :=û (y);
(6) END;
(1) PROCEDURA MICŞORARE;
(2) BEGIN
(3)    se determină un DmF D̂ utilizând vectorul succesor û;
```

(4) se determină $g(\hat{D}) = \min\{\hat{r}\ (x,y) \mid (x,y) \in \hat{D}\}$;
(5) se execută micşorarea de flux;
(6) se actualizează reţeaua reziduală $\hat{G}\ (f)$;
(7) END;

Algoritmul menţine un drum parţial admisibil memorat cu ajutorul vectorului succesor $\hat{u}$ şi execută operaţii de înaintare sau înapoiere de la nodul curent y. Dacă nodul curent y este extremitatea finală a unui arc admisibil (x,y) atunci se execută o operaţie de înaintare: se adaugă arcul (x,y) la începutul drumului parţial admisibil şi se reţine prin $\hat{u}\ (x) = y$, altfel se execută o operaţie de înapoiere. Operaţia de înapoiere constă în reetichetarea nodului y, $d(y) = \min\{d(x) \mid (x,y) \in \hat{A}\} + 1$ şi eliminarea arcului $\left(y, \hat{u}\ (y)\right)$ din drumul parţial admisibil dacă $y \neq t$. Repetăm aceste operaţii până când drumul parţial admisibil devine un drum admisibil şi atunci executăm o micşorare de flux. Continuăm acest proces până când fluxul devine minim.

Exemplul 7.3. Ilustrăm varianta algoritmului Ahuja-Orlin al drumului celui mai scurt pe reţeaua din figura 7.4(a). În figura 7.4(b) este reprezentată reţeaua reziduală corespunzătoare fluxului admisibil ilustrat în figura 7.4(a). Pornim de la nodul stoc $t = 4$ cu un drum admisibil parţial vid. Executăm o operaţie de înaintare şi adaugăm arcul $(2,4)$ la drumul parţial admisibil. Memorăm acest drum utilizând vectorul succesor $\hat{u}$, astfel $\hat{u}\ (2) = 4$. Acum nodul 2 este nodul curent şi algoritmul execută o operaţie de înaintare de la nodul 2. Se efectuează $\hat{u}\ (1) = 2$ şi se adaugă arcul $(1,2)$ la începutul drumului admisibil parţial, obţinându-se drumul admisibil $\hat{D}= (1,2,4)$. Se determină $g(\hat{D}) = \min\{\hat{r}\ (1,2), \hat{r}\ (2,4)\} = \min\{3,4\} = 3$ şi se micşorează cu 3 unităţi fluxul de-a lungul drumului $\hat{D}= (1,2,4)$ obţinându-se reţeaua reprezentată în figura 7.4(c). Din nou, pornim de la nodul stoc $t = 4$ cu un drum admisibil parţial vid. Executăm o operaţie de înaintare şi adaugăm arcul $(2,4)$ la drumul parţial admisibil şi efectuăm $\hat{u}\ (2) = 4$. Deoarece nodul curent, care este nodul 2, nu este extremitatea finală a nici unui arc admisibil, efectuăm o operaţie de înapoiere: $d(2) = \min\{d(3), d(4)\} + 1 = 2$, nodul curent devine nodul $\hat{u}\ (2) = 4$ şi eliminăm arcul $(2,4)$ din drumul parţial admisibil. La următoarele iteraţii, se vor determina drumurile admisibile $\hat{D}= (1,3,4)$, apoi $\hat{D}= (1,3,2,4)$. Reţeaua obţinută după micşorarea cu 4 unităţi a fluxului de-a lungul drumului $\hat{D}= (1,3,4)$ este reprezentată în figura 7.4(d). După micşorarea cu 2 unităţi a fluxului de-a lungul drumului $\hat{D}= (1,3,2,4)$ se obţine reţeaua reziduală corespunzătoare fluxului minim care este ilustrată în figura 7.4(e). Pornind de la capacităţile reziduale

optime şi folosind relaţiile (7.4) se determină fluxul minim care este reprezentat în figura 7.4(f).

Fig.7.4

Teorema 7.10. *Varianta algoritmului Ahuja - Orlin al drumului celui mai scurt determină un flux minim în reţeaua* $G = (N, A, \ell, c, s, t)$.
Demonstraţie. Analog cu demonstraţia Teoremei 6.10 de corectitudine a algoritmului Ahuja - Orlin al drumului celui mai scurt pentru fluxul maxim (a se vedea paragraful 6.1.5.). ■
Teorema 7.11. *Varianta algoritmului Ahuja - Orlin al drumului celui mai scurt are complexitatea* $O(n^2m)$.
Demonstraţie. Analiza complexităţii algoritmului se bazează pe următoarele două rezultate:
(1) complexitatea micşorărilor de flux este $O(n^2m)$.
(2) numărul operaţiilor de înapoiere este $O(n^2)$.

Din aceste rezultate, care se demonstrează analog cu cele similare de la algoritmul Ahuja - Orlin al drumului celui mai scurt pentru fluxul maxim (a se vedea paragraful 6.1.5.), rezultă că varianta algoritmului Ahuja - Orlin are complexitatea $O(n^2m)$. ■

7.3.3 Algoritmi polinomiali cu prefluxuri

7.3.3.1. Algoritmul preflux generic

Fie f un preflux în reţeaua $G = (N, A, \ell, c, s, t)$. Un nod $x \in N \setminus \{s, t\}$ se numeşte *nod activ* dacă $e(x) < 0$. Prin convenţie nodurile s şi t nu sunt active niciodată. Existenţa nodurilor active indică faptul că prefluxul nu este flux. De aceea, operaţia de bază a algoritmului constă în a selecta un nod activ şi a-i diminua deficitul. Pentru a diminua deficitul unui nod activ se retrage fluxul către nodurile adiacente lui care se găsesc mai "aproape" de nodul sursă. "Apropierea" nodurilor faţă de nodul sursă se măsoară cu ajutorul etichetelor distanţă exacte. De aceea, se retrage flux doar pe arce admisibile. În cazul în care nodul activ selectat nu este extremitatea finală a nici unui arc admisibil i se va mări eticheta distanţă astfel încât să se creeze cel puţin un arc admisibil. Această mărire a etichetei distanţă va fi numită operaţie de reetichetare. Algoritmul se termină atunci când în reţea nu mai există noduri active, ceea ce înseamnă că prefluxul este de fapt un flux.

Algoritmul preflux generic pentru fluxul minim este o variantă a algoritmului preflux generic pentru fluxul maxim.

(1) PROGRAM PREFLUX-GENERIC-Fm;
(2) BEGIN
(3) fie f un flux admisibil în reţeaua G;
(4) se determină reţeaua reziduală $\hat{G}(f)$;

(5) se calculează etichetele distanţă exacte $d(\cdot)$ în reţeaua reziduală $\hat{G}(f)$;
(6) IF t nu este etichetat
(7) THEN f este un flux minim
(8) ELSE BEGIN
(9) FOR $(x,t) \in E^-(t)$ DO $f(x,t) := \ell(x,t)$
(10) $d(t) := n$;
(11) WHILE în reţeaua reziduală $\hat{G}(f)$ există un nod activ DO
(12) BEGIN
(13) se selectează un nod activ y;
(14) IF în reţeaua reziduală $\hat{G}(f)$ există un arc admisibil (x,y)
(15) THEN se retrag $g = \min(-e(y), \hat{r}(x,y))$ unităţi de flux de la nodul y la nodul x;
(16) ELSE $d(y) := \min\{d(x) \mid (x,y) \in \hat{A}\} + 1$;
(17) END;
(18) END;
(19)END.

Exemplul 7.4. Ilustrăm algoritmul preflux generic pentru fluxul minim pe reţeaua din figura 7.5(a). În figura 7.5(b) este reprezentată reţeaua reziduală corespunzătoare fluxului admisibil ilustrat în figura 7.5(a). În reţeaua reziduală obţinută după iniţializările din liniile (9) şi (10) şi reprezentată în figura 7.5(c) există două noduri active: 2 şi 3. Presupunem că algoritmul selectează nodul activ 3. Rezultă că va retrage $g = \min(-e(3), \hat{r}(1,3)) = \min(4,6) = 4$ unităţi de flux pe arcul admisibil (1,3). În urma acestei operaţii, nodul 3 nu mai este activ. Apoi algorit-mul va selecta singurul nod activ, nodul 2 şi va retrage $g = \min(-e(2), \hat{r}(1,2)) = 3$ unităţi de flux pe arcul admisibil (1,2). Nodul 2 a rămas activ şi nu este extremitatea finală a nici unui arc admisibil, deci va fi reetichetat: $d(2) = \min\{d(3), d(4)\} + 1 = 4$. Reţeaua reziduală astfel obţinută este reprezentată în figura 7.5(d). Prin operaţia de reetichetare s-a creat arcul admisibil (3,2), pe care se vor retrage $g = 2$ unităţi de flux. Cum nodul 2 este în continuare activ şi nu mai există arce admisibile cu extremitatea finală nodul 2, rezultă că va fi reetichetat: $d(2) = 5$. În reţeaua reziduală obţinută, care este reprezentată în figura 7.5(e) există două noduri active: 2 şi 3. Dacă se selectează din nou nodul 2 se va retrage o unitate de flux pe arcul admisibil (4,2). Singurul nod rămas activ este 3, deci se vor retrage 2 unităţi de flux pe arcul admisibil (1,3). Astfel s-a obţinut reţeaua din figura 7.5(f), în care nu mai există noduri

active şi algoritmul se termină. Pornind de la capacităţile reziduale optime determinate cu algoritmul preflux generic pentru fluxul minim şi folosind relaţiile (7.4) se determină fluxul minim care este ilustrat în figura 7.5(g).

Fig.7.5.

Teorema 7.12. *Algoritmul preflux generic pentru fluxul minim determină un flux minim în reţeaua* $G = (N, A, \ell, c, s, t)$.

Demonstraţie. Algoritmul se termină atunci când $e(x) = 0,\ x \in N \setminus \{s, t\}$, ceea ce implică faptul că f este un flux. Deoarece $d(t) \geq n$, în reţeaua reziduală nu există drum de la nodul sursă s la nodul stoc t. Deci, fluxul curent este un flux minim. ■

Spunem că o retragere de flux de la nodul y la nodul x pe arcul (x, y) este *completă* dacă duce la eliminarea arcului (x, y) din reţeaua reziduală; în caz contrar spunem că este o retragere de flux *incompletă.*

Teorema 7.13. *Complexitatea algoritmului preflux generic pentru fluxul minim este* $O(n^2m)$.

Demonstraţie. Analiza complexităţii algoritmului preflux generic pentru fluxul minim se bazează pe următoarele trei rezultate:

(1) numărul total de operaţii de reetichetare este cel mult $2n^2$.

(2) algoritmul efectuează cel mult nm retrageri complete de flux.

(3) algoritmul efectuează $O(n^2m)$ retrageri incomplete de flux.

Aceste rezultate se demonstrează analog cu cele similare lor de la algoritmul preflux generic pentru fluxul maxim (a se vedea paragraful 6.2.1.).

Dacă nodurile active sunt păstrate într-o listă înlănţuită atunci complexitatea operaţiilor de adăugare, ştergere şi selectare a unui nod activ este $O(1)$. Deci, complexitatea algoritmului generic pentru fluxul minim este $O(n^2m)$. ■

În continuare sugerăm o îmbunătăţire practică a algoritmului preflux generic pentru fluxul minim. Fie v^* valoarea fluxului minim. Un preflux f cu $e(s) = v^*$ se numeşte *preflux minim.* Algoritmul preflux generic pentru fluxul minim poate să determine un preflux minim cu mult înainte de a determina un flux minim. După stabilirea prefluxului minim, algoritmul efectuează operaţii de reetichetare până când etichetele distanţă ale nodurilor active devin mai mari decât n şi se poate retrage flux de la nodurile active către nodul stoc t, a cărui etichetă distanţă este n. O posibilă modificare a algoritmului constă în introducerea mulţimii N' a nodurilor x cu proprietatea că în reţeaua reziduală nu există drum de la s la x. Iniţial $N' = \{t\}$. Folosim vectorul q, cu $q(k)$ egal cu numărul de noduri care au etichetele distanţă egale cu k. După fiecare operaţie de reetichetare actualizăm vectorul q în modul următor: Dacă algoritmul măreşte eticheta distanţă a unui nod y de la k_1 la k_2 atunci $q(k_1) := q(k_1) - 1$, $q(k_2) := q(k_2) + 1$. Dacă $q(k_1) = 0$ atunci în reţeaua reziduală nu există drum de la nodurile x cu $d(x) < k_1$ la nodurile y cu $d(y) > k_1$. Deci, vom adăuga toate nodurile y cu $d(y) > k_1$ la mulţimea N'. În continuare nu mai selectăm decât noduri active care nu sunt în N', iar algoritmul se termină când toate nodurile active sunt în N'. Prefluxul astfel obţinut este un preflux minim. Conform Teoremei 5.1, orice preflux poate fi descompus într-o sumă de cel mult $m+n$ fluxuri nenule de-a lungul unor drumuri şi circuite. Fie $M-$ mulţimea acestor drumuri şi circuite şi $M_0 \subseteq M-$ mulţimea drumurilor de la noduri cu deficit la nodul stoc t. Fie f_0- fluxul de-a lungul drumurilor din M_0. Rezultă că $f^* = f + f_0$ va fi un flux minim.

7.3.3.2. Algoritmul preflux FIFO

La fiecare iteraţie, algoritmul preflux generic pentru fluxul minim selectează un nod activ y şi efectuează fie o retragere de flux completă, fie o retragere incompletă, fie o reetichetare a nodului y. Dacă se efectuează o retragere de flux completă, atunci nodul y poate să rămână activ, dar nu este obligatoriu ca algoritmul să-l selecteze din nou la iteraţia următoare. Algoritmul preflux FIFO lucrează după următoare regulă: un nod y este selectat la iteraţii consecutive până când $e(y) = 0$ sau y este reetichetat. În consecinţă, algoritmul preflux FIFO pentru fluxul minim va executa mai multe retrageri complete de flux de la nodul y, urmate fie de o retragere incompletă ($e(y)$ devine zero), fie de o operaţie de reetichetare (nu există arc admisibil (x, y)). Numim această secvenţă de operaţii *examinarea nodului* y.

Notăm cu L lista nodurilor active. Algoritmul preflux FIFO pentru fluxul minim examinează nodurile active în ordinea primul intrat, primul ieşit (FIFO). Deci, lista nodurilor active L va fi organizată ca o coadă. Algoritmul se termină atunci când coada nodurilor active devine vidă.

Algoritmul preflux FIFO pentru fluxul minim este următorul:

```
(1) PROGRAM PREFLUX FIFO-Fm;
(2) BEGIN
(3)     fie f un flux admisibil în reţeaua G;
(4)     se determină reţeaua reziduală Ĝ(f);
(5)     se calculează etichetele distanţă exacte d(·) în reţeaua reziduală
        Ĝ(f);
(6)     IF t nu este etichetat
(7)         THEN f este un flux minim
(8)         ELSE BEGIN
(9)                  L := ∅;
(10)                 FOR (x,t) ∈ E⁻(t) DO
(11)                 BEGIN
(12)                    f(x,t) := ℓ(x,t);
(13)                    IF e(x) < 0 şi x ≠ s
(14)                        THEN se adaugă x în vârful cozii L;
(15)                 END;
(16)                 d(t) := n;
(17)                 WHILE L ≠ ∅ DO
(18)                 BEGIN
(19)                    se scoate primul nod y din L;
(20)                    RETRAGERE / REETICHETARE(y);
(21)                 END;
```

(22) END;
(23) END.

(1) PROCEDURA RETRAGERE / REETICHETARE(y);
(2) BEGIN
(3) se selectează primul arc (x, y) cu extremitatea finală y din $\hat{G}(f)$;
(4) B:=1;
(5) REPEAT
(6) IF (x, y) este arc admisibil
(7) THEN BEGIN
(8) se retrag $g := \min(-e(y), \hat{r}(x, y))$ unităţi de flux de la nodul y la nodul x;
(9) IF $x \notin L$ şi $x \neq s$ şi $x \neq t$
(10) THEN se adaugă x în vârful cozii L;
(11) END;
(12) IF $e(y) < 0$
(13) THEN IF (x, y) nu este ultimul arc cu extremitatea finală y din $\hat{G}(f)$;
(14) THEN se selectează următorul arc cu extremitatea finală y din $\hat{G}(f)$;
(15) ELSE BEGIN
(16) $d(y) := \min\{d(x) \mid (x, y) \in \hat{A}\} + 1$;
(17) $B := 0$;
(18) END;
(19) UNTIL $e(y) = 0$ sau $B = 0$;
(20) IF $e(y) < 0$
(21) THEN se adaugă y în vârful cozii L;
(22) END;

Exemplul 7.5. Ilustrăm algoritmul preflux FIFO pentru fluxul minim pe reţeaua din figura 7.6(a), în care este ilustrat şi un flux admisibil. După iniţializările din liniile (9) - (16) se obţine reţeaua reziduală din figura 7.6(b) şi $L = \{2, 3\}$. Algoritmul soate nodul 2 din coada L şi îl examinează: retrage 3 unităţi de flux pe arcul admisibil (1,2), iar apoi, pentru că nu mai există arce admisibile cu extremitatea finală 2 îl reeticheteză şi-l adaugă la sfârşitul cozii L. Deci, $L = \{3, 2\}$, iar reţeaua reziduală este reprezentată în figura 7.6(c). Apoi algoritmul scoate nodul 3 din coada L şi-l examinează retrăgând 4 unităţi de flux pe arcul (1,3). Astfel nodul 3 nu mai este activ, $L = \{2\}$. Reţeaua reziduală obţinută după examinarea nodului 3 este reprezentată în figura 7.6(d). În continuare algoritmul scoate din

coada L nodul 2, retrage 2 unităţi de flux pe arcul admisibil (3,2), adaugă nodul 3 în coadă deoarece a redevenit activ, reetichetează nodul 2: $d(2) = 5$ şi-l adaugă la sfârşitul cozii L. După examinarea nodului 2 se obţine reţeaua reziduală din figura 7.6(e), iar $L = \{3, 2\}$. Apoi algoritmul scoate nodul 3 din coada L şi-l examinează. Se retrag 2 unităţi de flux pe arcul admisibil (1,3) şi nodul 3 nu mai este activ. Reţeaua reziduală astfel obţinută este reprezentată în figura 7.6(f). Algoritmul va scoate nodul 2 din coadă şi va retrage o unitate de flux pe arcul (4,2). Astfel nodul 2 nu mai este activ, $L = \emptyset$ şi algoritmul se termină. Pornind de la capacităţile reziduale optime determinate cu algoritmul preflux FIFO pentru fluxul minim, care sunt ilustrate în figura 7.6(g), se determină fluxul minim din figura 7.6(h).

Fig.7.6.

Teorema 7.14. *Algoritmul preflux FIFO pentru fluxul minim determină un flux minim în reţeaua* $G = (N, A, \ell, c, s, t)$.
Demonstraţie. Rezultă din Teorema 7.12. ■
Teorema 7.15. *Complexitatea algoritmului preflux FIFO pentru fluxul minim este* $O(n^3)$.
Demonstraţie. Analog cu demonstraţia complexităţii algoritmului preflux FIFO pentru fluxul maxim. ■

7.3.3.3. Algoritmul preflux cu eticheta cea mai mare

Algoritmul preflux cu eticheta cea mai mare pentru fluxul minim retrage întotdeauna fluxul de la nodul activ y cu eticheta distanţă cea mai mare. Fie $p = \max\{d(x) \mid x \text{ este nod activ}\}$. Algoritmul examinează mai întâi nodurile active cu etichetele distanţă egale cu p şi retrage flux la nodurile cu etichetele distanţă egale cu $p-1$. Apoi de la aceste noduri retrage fluxul la nodurile cu etichetele distanţă egale cu $p-2$ şi aşa mai departe până când fie algoritmul reetichetează un nod, fie a epuizat toate nodurile active. După reetichetarea unui nod, algoritmul repetă acelaşi proces. Dacă algoritmul nu efectuează nici o reetichetare pe parcursul a n examinări succesive de noduri atunci tot deficitul ajunge în nodul sursă sau înapoi în nodul stoc şi algoritmul se termină.

Corectitudinea algoritmului preflux cu eticheta cea mai mare pentru fluxul minim rezultă din corectitudinea algoritmului preflux generic pentru fluxul minim.

Algoritmul preflux cu eticheta cea mai mare pentru fluxul minim poate fi implementat uşor dacă în algoritmul preflux FIFO pentru fluxul minim lista L a nodurilor active este organizată nu ca o coadă obişnuită, ci ca o coadă prioritară cu prioritatea funcţia distanţă d.

```
(1) PROGRAM PREFLUX ETICH-Fm;
(2) BEGIN
(3)     fie f un flux admisibil în reţeaua G;
(4)     se determină reţeaua reziduală Ĝ(f);
(5)     se calculează etichetele distanţă exacte d(·) în reţeaua reziduală Ĝ(f);
(6)     IF t nu este etichetat THEN f este un flux minim
(7)         ELSE BEGIN
(8)                  L := ∅;
(9)                  FOR (x,t) ∈ E⁻(t) DO
(10)                 BEGIN
(11)                     f(x,t) := ℓ(x,t);
(12)                     IF e(x) < 0 şi x ≠ s THEN
(13)                         se adaugă x în coada L având prioritatea d(x);
(14)                 END;
```

(15) $d(t) := n$;
(16) WHILE $L \neq \emptyset$ DO
(17) BEGIN
(18) se scoate din L nodul y cu prioritatea cea mai mare;
(19) RETRAGERE / REETICHETARE(y);
(20) END;
(21) END;
(22) END.

(1) PROCEDURA RETRAGERE / REETICHETARE(y);
(2) BEGIN
(3) arcul curent este primul arc din $\hat{E}^{-}(y)$;
(4) B:=1;
(5) REPEAT
(6) fie (x, y) arcul curent din $\hat{E}^{-}(y)$;
(7) IF (x, y) este arc admisibil THEN
(8) BEGIN
(9) se retrag $g := \min(-e(y), \hat{r}(x, y))$ unităţi de flux de la nodul y la nodul x;
(10) IF $x \notin L$ şi $x \neq s$ şi $x \neq t$ THEN
(11) se adaugă x în coada L având prioritatea $d(x)$;
(12) END;
(13) IF $e(y) < 0$ THEN
(14) IF (x, y) nu este ultimul arc din $\hat{E}^{-}(y)$
(15) THEN arcul curent este următorul arc din $\hat{E}^{-}(y)$;
(16) ELSE BEGIN
(17) $d(y) := \min\{d(x) \mid (x, y) \in \hat{A}\} + 1$;
(18) $B := 0$;
(19) END;
(20) UNTIL $e(y) = 0$ sau $B = 0$;
(21) IF $e(y) < 0$ THEN se adaugă y în coada L având prioritatea $d(y)$;
(22) END;

Teorema 7.16. *Dacă există un flux admisibil în reţeaua G, atunci algoritmul preflux cu eticheta cea mai mare pentru fluxul minim determină un flux minim în reţeaua $G = (N, A, \ell, c, s, t)$.*

Demonstraţie. Rezultă din Teorema 7.12. ■

Teorema 7.17. *Complexitatea algoritmului preflux cu eticheta cea mai mare pentru fluxul minim este $O(n^2 m^{1/2})$.*

Demonstraţie. Analog cu demonstraţia complexităţii algoritmului preflux cu eticheta cea mai mare pentru fluxul maxim, pentru care se recomandă cititorului

monografiile precizate la bibliografie ■

Exemplul 7.6. Ilustrăm algoritmul preflux cu eticheta cea mai mare pentru fluxul minim pe reţeaua din figura 7.7(a). După iniţializările din liniile (8) - (15) se obţine reţeaua reziduală din figura 7.7(b), iar în coada L se găsesc nodurile 2 cu prioritatea 1 şi 3 cu prioritatea 1. Algoritmul scoate nodul 2 din coada L şi îl examinează, adică retrage $g = \min\{-e(2), \hat{r}(1,2)\} = 3$ unităţi pe arcul (1,2), apoi reetichetează nodul 2: $d(2) = 2$ şi-l adaugă în coada L cu prioritatea 2. Reţeaua reziduală obţinută după examinarea nodului 2 este reprezentată în figura 7.7(c), iar coada L va conţine nodul 2 cu prioritatea 2 şi nodul 3 cu prioritatea 1. La următoarea iteraţie, algoritmul va scoate din nou nodul 2 din coadă şi-l va examina, ceea ce înseamnă că va retrage 2 unităţi de flux pe arcul (3,2) şi apoi va efectua o reetichetare: $d(2) = 5$ şi va adăuga nodul 2 cu prioritatea 5 în coada L. Deci coada L va conţine nodul 2 cu prioritatea 5 şi nodul 3 cu prioritatea 1, iar reţeaua reziduală este cea din figura 7.7(d). Din nou nodul 2 este scos din coadă şi examinat. Se retrage o unitate de flux pe arcul (4,2), obţinându-se reţeaua din figura 7.7(e). La următoarea iteraţie singurul nod din coada L, nodul 3, va fi scos şi examinat. Se retrag 6 unităţi de flux pe arcul (1,3) şi algoritmul se termină, coada L rămânând vidă. Reţeaua reziduală obţinută la terminarea algoritmului este ilustrată în figura 7.7(f), iar fluxul minim este reprezentat în figura 7.7(g).

(c) (d)

Fig.7.7.

7.3.3.4. Algoritmul de scalare a deficitului

Acest algoritm este o variantă îmbunătăţită a algoritmului preflux generic pentru fluxul minim. În algoritmul generic cu prefluxuri cele mai mari consumatoare de timp sunt retragerile incomplete de flux $\left(O(n^2m)\right)$. În algoritmul de scalare a deficitului, vom reduce numărul acestora la $O(n^2 \log \overline{c})$ folosind tehnica scalării, adică impunând condiţia ca să se efectueze doar retrageri incomplete de cantităţi suficient de mari de flux.

Fie $\overline{r}$ o margine superioară pentru $e_{\max} = \max\{|e(x)| \;\; x \text{ este nod activ}\}$. Spunem că un nod y are un *deficit mare* dacă $e(y) \leq -\overline{r}/2$; în caz contrar nodul y are un *deficit mic.*

Algoritmul de scalare a deficitului retrage întotdeauna flux de la un nod cu deficit mic pentru că niciun nod să nu acumuleze deficit prea mare. Algoritmul de scalare a deficitului pentru fluxul minim este următorul:

(1) PROGRAM SCALARE DEFICIT-Fm;
(2) BEGIN
(3) fie f un flux admisibil în reţeaua G;
(4) se determină reţeaua reziduală $\hat{G}(f)$;
(5) se calculează etichetele distanţă exacte $d(\cdot)$ în reţeaua reziduală $\hat{G}(f)$;
(6) IF t nu este etichetat THEN f este un flux minim
(7) ELSE BEGIN
(8) FOR $(x,t) \in E^-(t)$ DO $f(x,t) := \ell(x,t)$;
(9) $d(t) := n$;
(10) $\overline{r} := 2^{\lceil \log \overline{c} \rceil}$;
(11) WHILE $\overline{r} \geq 1$ DO
(12) BEGIN
(13) WHILE în reţeaua reziduală $\hat{G}(f)$ există un nod activ cu deficit mare DO
(14) BEGIN
(15) se selectează nodul y astfel încât $d(y) = \min\{d(z) \,|\, z \text{ nod cu deficit mare}\}$;
(16) RETRAGERE/REETICHETARE(y);
(17) END;

(18) $\overline{r} := \overline{r}/2$;
(19) END;
(20) END;
(21) END.
(1) PROCEDURA RETRAGERE/REETICHETARE(y);
(2) BEGIN
(3) IF în reţeaua reziduală $\hat{G}(f)$ există un arc admisibil (x, y) THEN
(4) se retrag $g = \min(-e(y), \hat{r}(x, y), \overline{r} + e(x))$ unităţi de flux de la nodul y la nodul x;
(5) ELSE $d(y) := \min\{d(x) \mid (x, y) \in \hat{A}\} + 1$;
(6) END;

Numim *fază de scalare* o fază a algoritmului pe parcursul căreia $\overline{r}$ rămâne constant. O fază de scalare pentru o valoare dată a lui $\overline{r}$ se numeşte *fază de* $\overline{r}-$ *scalare.*

Exemplul 7.7. Ilustrăm algoritmul de scalare a deficitului pe reţeaua din figura 7.8(a). După iniţializările din liniile (4) - (10) se obţine reţeaua reziduală din figura 7.8(b) şi $\overline{r} = 8$, iar nodurile cu deficit mare sunt 2 şi 3. Dacă algoritmul selectează nodul 2 atunci se retrag $g = \min\{-e(2), \hat{r}(1,2), \overline{r} + e(1)\} = \min\{6, 3, 22\} = 3$ unităţi de flux pe arcul admisibil $(1,2)$ şi se obţine reţeaua din figura 7.8(c). Apoi se selectează singurul nod cu deficit mare, nodul 3, se retrag 4 unităţi de flux pe arcul admisibil (1,3) şi se obţine reţeaua reprezentată în figura 7.8(d), în care nu mai există noduri cu deficite mari. Deci, se înjumătăţeşte valoarea lui $\overline{r}$ şi se începe o nouă fază de scalare pentru $\overline{r} = 4$. Singurul nod cu deficit mare este nodul 2 pe care algoritmul îl va selecta şi îl va reeticheta: $d(2) = 2$, creând astfel arcul admisibil (3,2). La următoarea iteraţie se selectează din nou nodul 2, se retrag 2 unităţi de flux pe arcul (3,2) şi se obţine reţeaua din figura 7.8(e). După această retragere de flux nodul 2 va avea deficit mic, iar nodul 3 va avea deficit mare, deci va fi selectat şi se vor retrage 2 unităţi de flux pe arcul admisibil (1,3). În figura 7.8(f) este reprezentată reţeaua reziduală astfel obţinută, în care nu mai există niciun nod cu deficit mare. Deci, se înjumătăţeşte valoarea lui $\overline{r}$ şi se începe o nouă fază de scalare pentru $\overline{r} = 2$. Algoritmul va selecta nodul 2 care este singurul nod cu deficit mare şi-l va reeticheta: $d(2) = 5$. Apoi se va selecta din nou nodul 2 şi se va retrage o unitate de flux pe arcul admisibil (4,2) obţinându-se reţeaua reziduală reprezentată în figura 7.8(g) în care nu mai există noduri cu deficit mare. Se efectuează $\overline{r} = 1$ şi, deoarece nu mai există noduri active algoritmul se termină. În figura 7.8(h) este reprezentat fluxul minim determinat de algoritmul de scalare a deficitului.

Teorema 7.18. *Dacă există un flux admisibil în reţeaua* G*, atunci algoritmul de scalare a deficitului determină un flux minim în* G*.*
Demonstraţie. La începutul algoritmului $\overline{r} := 2^{\lceil \log \overline{c} \rceil}$, deci $\overline{c} \leq \overline{r} \leq 2\overline{c}$. Pe parcursul fazei de $\overline{r}-$ scalare, $e_{\max}$ poate să crească şi să descrească, dar trebuie să fie îndeplinite condiţiile $\overline{r}/2 < e_{\max} \leq \overline{r}$. Când $e_{\max} \leq \overline{r}/2$ atunci se înjumătăţeşte valoarea lui $\overline{r}$ şi se începe o nouă fază de scalare. După $1 + \lceil \log \overline{c} \rceil$ faze de scalare, $e_{\max}$ devine nul şi fluxul astfel obţinut este un flux minim. ■

Teorema 7.19. *Pe parcursul fiecărei faze de* $\overline{r}-$ *scalare sunt îndeplinite următoarele două condiţii:*
(a) *la fiecare retragere incompletă de flux se retrag cel puţin* $\overline{r}/2$ *unităţi de flux*
(b) $e_{\max} \leq \overline{r}$.
Demonstraţie. (a) Considerăm o retragere incompletă de flux pe un arc oarecare (x, y). Deoarece arcul (x, y) este un arc admisibil, avem că $d(y) = d(x)+1 > d(x)$. Dar, nodul y a fost selectat ca fiind nodul cu cea mai mică etichetă distanţă dintre

nodurile cu deficit mare. Deci, $e(y) \leq -\overline{r}/2$ şi $e(x) > -\overline{r}/2$. Deoarece pe arcul (x, y) se efectuează o retragere incompletă de flux, rezultă că se micşorează fluxul cu $\min\{-e(y), \overline{r} + e(x)\} \geq \overline{r}/2$ unităţi.

(b) O retragere de flux pe arcul (x, y) măreşte doar valoarea absolută a deficitului nodului x. Noul deficit al nodului x este $e'(x) = e(x) - \min\{-e(y), r(x, y), \overline{r} + e(x)\} \geq e(x) - (\overline{r} + e(x)) = -\overline{r}$. Deci, $e'(x) \geq -\overline{r}$ şi $e_{\max} \leq \overline{r}$. ■

Fig. 7.8.

Teorema 7.20. *Pe parcursul fiecărei faze de scalare, algoritmul de scalare a deficitului efectuează $O(n^2)$ retrageri incomplete de flux.*

Demonstraţie. Considerăm funcţia potenţial $P = -\sum_{x \in N} e(x)d(x)/\overline{r}$. Valoarea iniţială a lui P la începutul fazei de $\overline{r}-$ scalare este mărginită superior de $2n^2$ deoarece $e(x) \geq -\overline{r}$ şi $d(x) \leq 2n$, $x \in N$, conform Teoremei 7.19.*b* După ce algoritmul a selectat nodul y, putem avea unul din următoarele două cazuri:

Cazul 1. Nu există arc admisibil incident către interior cu nodul y. În acest caz eticheta distanţă a nodului y creşte cu $q \geq 1$ unităţi, ceea ce duce la creşterea lui P cu cel mult q unităţi pentru că $e(y) \geq -\overline{r}$. Deoarece, pentru fiecare nod y, $d(y)$ poate creşte cu cel mult $2n$ unităţi pe parcursul execuţiei algoritmului, rezultă că valoarea lui P creşte cu cel mult $2n^2$ unităţi din cauza reetichetării nodurilor.

Cazul 2. Există un arc admisibil incident către interior cu nodul y, fie acesta (x, y), pe care algoritmul efectuează o retragere completă sau incompletă. În ambele cazuri P descreşte. După o retragere incompletă de flux pe arcul (x, y), fluxul de la nodul x la nodul y scade, conform Teoremei 7.19.*a*, cu cel puţin $\overline{r}/2$ unităţi, deci P descreşte cu cel puţin $1/2$ pentru că $d(y) = d(x) + 1$. Deoarece valoarea iniţială a lui P la începutul fazei de scalare era cel mult $2n^2$ şi putea să crească cu încă cel mult $2n^2$ unităţi, rezultă că acest caz poate să apară de cel mult $8n^2$ ori. Adică, pe parcursul fiecărei faze de scalare, algoritmul de scalare a deficitului efectuează $O(n^2)$ retrageri incomplete de flux. ■

Teorema 7.21 *Complexitatea algoritmului de scalare a deficitului este $O(nm + n^2 \log \overline{c})$.*

Demonstraţie. Numărul total de retrageri incomplete de flux este $O(n^2 \log \overline{c})$, conform Teoremei 7.20, deoarece algoritmul efectuează $O(\log \overline{c})$ faze de scalare. Celelalte operaţii (retrageri complete, reetichetări de noduri şi determinări de arce admisibile) necesită $O(nm)$ timp, ca şi în cazul algoritmului preflux generic pentru fluxul minim. Deci, complexitatea algoritmului de scalare a deficitului este $O(nm + n^2 \log \overline{c})$. ■

7.4 Algoritmul minimax

Putem rezolva problema fluxului minim de la nodul sursă s la nodul stoc t în reţeaua G aplicând un algoritm de flux maxim de la t la s în reţeaua reziduală. Această abordare se bazează pe următoarea idee: obiectivul în cazul problemei fluxului minim este de a trimite cât mai puţin flux de la nodul sursă la nodul stoc, adică inversul obiectivului problemei fluxului maxim. Algoritmul minimax calculează un flux minim de la s la t în modul următor: fiind dat un flux admisibil, determină un flux maxim de la nodul t la nodul s în reţeaua reziduală $\tilde{G}(f)$, definită ca pentru o problemă de flux maxim. Acest flux va fi un flux minim de la s la t în reţeaua G.

Algoritmul minimax este următorul:
(1) PROGRAM MINIMAX;
(2) BEGIN
(3) fie f un flux admisibil în G;
(4) se determină reţeaua reziduală $\tilde{G}(f)$;
(5) se determină un flux maxim f^* de la t la s în $\tilde{G}(f)$;
(6) f^* este un flux minim de la s la t în G;
(7) END.

Exemplul 7.8. Ilustrăm algoritmul minimax pe reţeaua din figura 7.9(a) în care $s = 1$ şi $t = 4$. În figura 7.9(b) este reprezentată reţeaua reziduală $\tilde{G}(f)$ corespunzătoare fluxului admisibil ilustrat în figura 7.9(a). În figura 7.9(c) sunt reprezentate capacităţile reziduale corespunzătoare unui flux maxim de la nodul 4 la nodul 1, care pot fi obţinute aplicând orice algoritm pentru fluxul maxim. Pornind de le aceste capacităţi reziduale şi folosind relaţiile

$$f(x,y) = \ell(x,y) + \max(0,\, c(x,y) - r(x,y) - \ell(x,y))$$

şi

$$f(y,x) = \ell(y,x) + \max(0,\, c(y,x) - r(y,x) - \ell(y,x))$$

obţinem fluxul din figura 7.9(d) care este un flux minim de la nodul 1 la nodul 4.

Fig.7.9

Teorema 7.22. *Algoritmul minimax determină un flux minim de la nodul s la nodul t în reţeaua $G = (N, A, \ell, c, s, t)$.*
Demonstraţie. Fie f^* fluxul obţinut la sfârşitul execuţiei algoritmului. Deci, f^* este un flux maxim de la nodul t la nodul s. Conform Teoremei 5.11, rezultă că

în reţeaua reziduală $\tilde{G}(f)$ nu există drum de la nodul t la nodul s. Deci, există o tăietură $t-s$ $[X, \overline{X}]$ astfel încât

$$r(x,y) = 0, \ \ (x,y) \in (X, \overline{X}).$$

Sau echivalent,

$$c(x,y) - f(x,y) + f(y,x) - \ell(y,x) = 0, \ \ (x,y) \in (X, \overline{X}).$$

Dar $f(x,y) \leq c(x,y), \ (x,y) \in A$ şi $f(y,x) \geq \ell(y,x), \ (y,x) \in A$.
Deci,

$$f(x,y) = c(x,y), \ \ (x,y) \in (X, \overline{X})$$
$$f(y,x) = \ell(y,x), \ \ (y,x) \in (\overline{X}, X)$$

Sau echivalent,

$$c(x,y) - f(x,y) + f(y,x) - \ell(y,x) = 0, \ \ (y,x) \in (\overline{X}, X).$$

adică

$$\hat{r}(y,x) = 0, \ (y,x) \in (\overline{X}, X) \tag{7.7}$$

Deoarece $[X, \overline{X}]$ este o tăietură $t-s$, rezultă că $[\overline{X}, X]$ este o tăietură $s-t$. Din relaţia (7.7), rezultă că în reţeaua reziduală $\hat{G}(f^*)$ definită pentru problema fluxului minim nu există drum de la nodul s la nodul t. Deci, conform Teoremei 7.3, f^* este un flux minim de la nodul s la nodul t.

Teorema 7.23. *Complexitatea algoritmului minimax este egală cu complexitatea algoritmului folosit pentru determinarea fluxului maxim de la nodul stoc la nodul sursă.*

Demonstraţie. Evident. ■

7.5 Aplicaţii şi comentarii bibliografice

7.5.1 Problema planificării lucrărilor

La un atelier trebuie efectuate într-o singură zi p lucrări. Se cunosc timpii $\tau(i)$ şi $\tau'(i)$ de începere şi de terminare a fiecărei lucrări $i, \ i = 1, \ldots, p$. De asemenea, se cunosc timpii $\tau_2(i,j)$ de deplasare a unui muncitor de la lucrarea i la lucrarea j. Muncitorii trebuie să realizeze aceste lucrări conform orarului, astfel încât o lucrare să fie efectuată de un singur muncitor şi care nu poate lucra în acelaşi timp la mai multe lucrări.

Problema planificării lucrărilor se poate modela ca o problemă de flux minim într-o reţea $G = (N, A, l, c)$, unde: $N = N_1 \cup N_2 \cup N_3 \cup N_4, \ N_1 = \{s\}, \ N_2 =$

$\{i \mid i = 1, \ldots, p\}$, $N_3 = \{i' \mid i' = 1, \ldots, p\}$, $N_4 = \{t\}$, $A = A_1 \cup A_2 \cup A_3 \cup A_4$, $A_1 = \{(s, i) \mid i \in N_2\}$, $A_2 = \{(i, i') \mid i, i' = 1, \ldots, p\}$, $A_3 = \{(i', j) \mid \tau'(i') + \tau_2(i', j) \leq \tau(j)\}$, $A_4 = \{(i', t), \mid i' \in N_3\}$, $l(s, i) = 0$, $c(s, i) = 1$, $(s, i) \in A_1$, $l(i, i') = 1$, $c(i, i') = 1, (i, i') \in A_2, l(i', j) = 0$, $c(i', j) = 1$, $(i', j) \in A_3$, $l(i', t) = 0$, $c(i', t) = 1$, $(i', t) \in A_4$. Deoarece i şi i' reprezintă aceeaşi lucrare avem $\tau(i') = \tau(i)$ şi $\tau'(i') = \tau'(i)$.

Exemplul 7.9 Pentru $p = 3$ se dau timpii $\tau(i), \tau'(i), \tau_2(i, j)$ în tabelul de mai jos.

i	$\tau(i)$	$\tau'(i)$	$\tau_2(i,j)$ în minute		
			1	2	3
1	13.00	13.30	0	20	25
2	18.00	20.00	15	0	35
3	19.00	21.00	30	5	0

Reţeaua $G = (N, A, l, c)$ este prezentată în figura 7.10.

Fig. 7.10.

Un flux minim este reprezentat pe reţeaua din figura 7.11.

Fig. 7.11.

Rezultă că numărul minim de muncitori pentru efectuarea celor trei lucrări conform orarului este $q = v = 2$. Primul muncitor execută prima lucrare, iar apoi după terminarea acesteia se deplasează la a doua lucrare pe care o efectuează. Al doilea muncitor execută ultima lucrare. Soluţia nu este unică. Cititorul poate să determine o a doua soluţie.

7.5.2 Comentarii bibliografice

În acest capitol sunt prezentate trei abordări ale problemei fluxului minim. Prima clasă de algoritmi pentru determinarea unui flux minim este formată din algoritmi cu drumuri de micşorare. Ciurea şi Ciupală (2001) au descris algoritmul generic cu drumuri de micşorare, care a fost obţinut plecând de la algoritmul generic cu drumuri de mărire pentru problema fluxului maxim.

Algoritmii bazaţi pe prefluxuri formează a doua clasă de algoritmi pentru determinarea unui flux minim. Ciurea şi Ciupală (2001) au descris algoritmul preflux generic pentru determinarea unui flux minim şi două variante îmbunătăţite ale acestuia obţinute prin impunerea diferitelor reguli de selectare a nodurilor active: algoritmul preflux FIFO şi algoritmul preflux cu eticheta cea mai mare.

A treia abordare a problemei fluxului minim de la nodul sursă s la nodul stoc t constă în a determina un flux maxim de la nodul t la nodul s în reţeaua reziduală. Algoritmul care se bazează pe această observaţie a fost descris independent de Bang - Jensen şi Gutin (2001) şi Ciupală şi Ciurea (2001).

Capitolul 8

Fluxuri de cost minim

8.1 Noţiuni introductive

Problema fluxului de cost minim este una dintre cele mai importante probleme de fluxuri în reţele. În capitolele 4, 5 şi 6 am prezentat două cazuri particulare ale acesteia: problema drumului minim şi problema fluxului maxim.

Fie $G = (N, A)$ un digraf definit prin mulţimea N cu n noduri şi mulţimea A cu m arce. Se definesc funcţiile capacitate $c : A \to \Re_+$, cost $b : A \to \Re$ şi cerere/ofertă $v : N \to \Re$. Capacitatea $c(x, y)$ a arcului (x, y) reprezintă cantitatea maximă de flux care poate traversa arcul, costul $b(x, y)$ reprezintă costul transportului unei unităţi de flux de la nodul x la nodul y pe arcul (x, y), iar $v(x)$ reprezintă cererea nodului x, dacă $v(x) < 0$ sau oferta nodului x, în caz contrar.

Problema fluxului de cost minim în reţeaua $G = (N, A, c, b)$ constă în a determina un flux f care verifică constrângerile.

$$\text{minimizează } z(f) = \sum_{(x,y)\in A} b(x, y) \cdot f(x, y) \tag{8.1.a}$$

$$\sum_y f(x, y) - \sum_y f(y, x) = v(x), \ \ x \in N \tag{8.1.b}$$

$$0 \le f(x, y) \le c(x, y), \ \ (x, y) \in A \tag{8.1.c}$$

8.2 Ipoteze asupra problemei fluxului de cost minim

În continuare vom nota cu $\overline{c} = \max(\max\{c(x,y) \mid (x,y) \in A\},\ \max\{v(x) \mid x \in N\})$ şi cu $\overline{b} = \max\{b(x,y)) \mid (x,y) \in A\}$.

Ipoteza 1. Reţeaua este orientată.

Această ipoteză nu restrânge generalitatea deoarece problemele formulate în reţele neorientate sau mixte pot fi rezolvate în reţele orientate echivalente, care se construiesc în modul următor: o muchie $[x,y]$ de cost $b[x,y]$ şi capacitate $c[x,y]$ permite trecerea fluxului de la nodul x la nodul y şi invers, de la nodul y la nodul x. Transportul unei unităţi de flux în fiecare direcţie are costul $b[x,y]$, iar cantitatea de flux de pe muchia $[x,y]$ nu poate depăşi $c[x,y]$. Deci, muchia $[x,y]$ are constrângerea $f[x,y] + f[y,x] \leq c[x,y]$ şi măreşte funcţia obiectiv $z(f)$ cu $b[x,y] \cdot f[x,y] + b[x,y] \cdot f[y,x]$. Considerăm că fluxul este permis numai într-unul dintre cele două sensuri, deci $f[x,y] \cdot f[y,x] = 0$. Pentru a transforma problema fluxului de cost minim din reţele neorientate sau mixte în reţele orientate, înlocuim fiecare muchie $[x,y]$ cu două arce: (x,y) şi (y,x), ambele având costul $b[x,y]$ şi capacitatea $c[x,y]$. Pentru a demonstra echivalenţa dintre reţeaua astfel obţinută şi reţeaua iniţială trebuie să arătăm că oricărui flux din reţeaua iniţială îi corespunde un flux cu acelaşi cost în reţeaua transformată şi reciproc.

Fie f un flux din reţeaua iniţială. Definim fluxul f' din reţeaua transformată astfel:

- dacă fluxul este de la x la y atunci $f'(x,y) = f[x,y]$ şi $f'(y,x) = 0$
- dacă fluxul este de la y la x atunci $f'(x,y) = 0$ şi $f'(y,x) = f[x,y]$.

Evident, fluxurile f şi f' au acelaşi cost.

Reciproc, fie f' un flux din reţeaua transformată. Definim fluxul f din reţeaua iniţială astfel:

- dacă $f'(x,y) > f'(y,x)$ atunci $f[x,y] = f'(x,y) - f'(y,x)$ şi $f[y,x] = 0$;
- dacă $f'(x,y) = f'(y,x)$ atunci $f[x,y] = 0$ şi $f[y,x] = 0$;
- dacă $f'(x,y) < f'(y,x)$ atunci $f[x,y] = 0$ şi $f[y,x] = f'(y,x) - f'(x,y)$.

Evident, costul fluxului f este egal cu costul fluxului f'.

Ipoteza 2. Toate datele problemei (capacităţi, costuri, cereri/oferte) sunt întregi.

Această ipoteză nu este restrictivă deoarece calculatorul lucrează cu numere raţionale, iar acestea pot fi transformate în numere întregi prin înmulţirea cu un număr întreg convenabil ales.

Ipoteza 3. Reţeaua nu conţine circuite de cost negativ formate doar din arce cu capacitate infinită.

Dacă reţeaua ar conţine astfel de circuite atunci costul fluxului poate fi arbitrar de mic (ar putea fi micşorat oricât prin mărirea fluxului de-a lungul unui astfel de circuit).

Ipoteza 4. Costurile arcelor sunt nenegative.

Această ipoteză nu restrânge generalitatea problemei deoarece putem transforma reţeaua într-o reţea echivalentă care nu conţine arce cu costuri negative. Dar, această transformare poate fi făcută doar pentru arcele cu capacitate finită. Cum, conform ipotezei 3, reţeaua nu conţine circuite de cost negativ formate doar din arce cu capacitate infinită, putem înlocui arcele cu capacitate infinită cu arce cu capacitatea egală cu $\sum_{(x,y)\in A} c(x,y) + \sum_{x\in N} v(x)$. Transformarea arcului (x,y) de cost negativ în arcul (y,x) de cost pozitiv, este ilustrată în figura 8.1. Transformarea constă în înlocuirea variabilei $f(x,y)$ cu $c(x,y) - f(y,x)$ şi are următoarea interpretare. Mai întâi trimitem $c(x,y)$ unităţi de flux pe arcul (x,y) (ceea ce duce la micşorarea lui $v(x)$ cu $c(x,y)$ şi mărirea lui $v(y)$ cu $c(x,y)$) şi apoi înlocuim arcul (x,y) cu arcul (y,x) care va avea costul $-b(x,y)$.

Fig.8.1.

Ipoteza 5. Cererile/ofertele nodurilor satisfac condiţia $\sum_{x\in N} v(x) = 0$ şi problema fluxului de cost minim are o soluţie admisibilă.

Putem determina dacă problema fluxului de cost minim are o soluţie admisibilă rezolvând o problemă de flux maxim de la $\tilde{s}$ la $\tilde{t}$ în reţeaua $\tilde{G} = \left(\tilde{N}, \tilde{A}, \tilde{c}\right)$ unde $\tilde{N} = N \cup \{\tilde{s}, \tilde{t}\}$, $\tilde{A} = A \cup A_{\tilde{s}} \cup A_{\tilde{t}}$, $A_{\tilde{s}} = \{(\tilde{s}, x) \mid x \in N \text{ şi } v(x) > 0\}$, $A_{\tilde{t}} = \{(x, \tilde{t}) \mid x \in N \text{ şi } v(x) < 0\}$, $\tilde{c}(\tilde{s}, x) = v(x)$, $(\tilde{s}, x) \in A_{\tilde{s}}$, $\tilde{c}(x,y) = c(x,y)$, $(x,y) \in A$, $\tilde{c}(x, \tilde{t}) = -v(x)$, $(x, \tilde{t}) \in A_{\tilde{t}}$. Dacă fluxul maxim saturează toate arcele din $A_{\tilde{s}}$ şi $A_{\tilde{t}}$ atunci problema fluxului de cost minim are o soluţie admisibilă; în caz contrar, nu are soluţie admisibilă.

Ipoteza 6. În reţea există câte un drum de capacitate infinită (fiecare arc al drumului are capacitate infinită) între oricare două noduri.

Dacă reţeaua nu îndeplineşte această condiţie, adăugăm arce artificiale de forma $(1,y)$ şi $(y,1), y \in N$ cu costul mare şi capacitatea infinită. Dacă problema fluxului de cost minim are o soluţie admisibilă în reţeaua fără arce artificiale, atunci fluxul de cost minim va fi nul pe arcele artificiale.

Ipoteza 7. Reţeaua nu conţine două noduri între care să existe arce în ambele sensuri. Această condiţie nu restrânge generalitatea problemei deoarece dacă $(x,y) \in A$ şi $(y,x) \in A$ atunci introducem un nou nod z şi înlocuim arcul (y,x) cu arcele (y,z) şi (z,x) cu $c(y,z) = c(y,x)$, $b(y,z) = b(y,x)$, $c(z,x) = c(y,x)$, $b(z,x) = 0$.

8.3 Condiţii de optimalitate

Înainte de a prezenta condiţiile de optimalitate, trebuie să definim noţiunea de reţea reziduală. Reţeaua reziduală $\tilde{G}(f)$ corespunzătoare fluxului f se construieşte astfel: înlocuim fiecare arc $(x, y) \in A$ cu două arce (x, y) şi (y, x). Arcul (x, y) are costul $b(x, y)$ şi la capacitatea reziduală $r(x, y) = c(x, y) - f(x, y)$, iar arcul (y, x) are costul $-b(x, y)$ şi capacitatea reziduală $r(y, x) = f(x, y)$. Reţeaua reziduală $\tilde{G}(f)$ este formată doar din arcele cu capacitatea reziduală strict pozitivă.

8.3.1 Condiţiile de optimalitate cu circuite de cost negativ

Teorema 8.1 (Condiţiile de optimalitate cu circuite de cost negativ). *Un flux f^* este un flux de cost minim dacă şi numai dacă reţeaua reziduală $\tilde{G}(f^*)$ nu conţine circuite de cost negativ.*

Demonstraţie. Fie f^* un flux de cost minim. Presupunem prin absurd că reţeaua reziduală $\tilde{G}(f^*)$ conţine un circuit de cost negativ. Mărind fluxul pe acest circuit se obţine un flux cu costul mai mic decât costul lui f^*, ceea ce contrazice ipoteza. Rezultă că presupunerea a fost falsă deci, dacă f^* este un flux de cost minim atunci reţeaua reziduală $\tilde{G}(f^*)$ nu conţine circuite de cost negativ.

Reciproc, considerăm că f^* este un flux admisibil şi reţeaua reziduală $\tilde{G}(f^*)$ nu conţine circuite de cost negativ. Fie f^0 un flux de cost minim, $f^0 \neq f^*$. Conform Teoremei 5.1 putem descompune $f^0 - f^*$ în cel mult m circuite de mărire în raport cu fluxul f^*, iar costul total al acestor circuite este $bf^0 - bf^*$. Cum reţeaua reziduală $\tilde{G}(f^*)$ nu conţine circuite de cost negativ rezultă că $bf^0 - bf^* \geq 0$. Dar f^0 este un flux de cost minim, deci $bf^0 \leq bf^*$. Rezultă că $bf^0 = bf^*$, deci f^* este de asemenea un flux de cost minim.■

8.3.2 Condiţiile de optimalitate cu costuri reduse

Asociem fiecărui nod $x \in N$ un număr real $p(x)$, numit *potenţialul nodului* x. Având potenţialele nodurilor p, definim costul redus al arcului (x, y) astfel

$$b^p(x, y) = b(x, y) - p(x) + p(y).$$

Observaţia 8.1. Noţiunea de cost redus are sens atât în reţeaua originară cât şi în reţeaua reziduală.

Proprietatea 8.2.(a) Fie D un drum oarecare de la nodul w la nodul z. Are loc următoarea relaţie:

$$\sum_{(x,y)\in D} b^p(x, y) = \sum_{(x,y)\in D} b(x, y) - p(w) + p(z).$$

(b) Fie $\overset{\circ}{D}$ un circuit oarecare. Are loc următoarea relaţie:

$$\sum_{(x,y)\in\overset{\circ}{D}} b^p(x,y) = \sum_{(x,y)\in\overset{\circ}{D}} b(x,y).$$

Demonstraţie.

$$\textbf{(a)} \sum_{(x,y)\in D} b^p(x,y) = \sum_{(x,y)\in D} b(x,y) - \sum_{(x,y)\in D} p(x) + \sum_{(x,y)\in D} p(y) =$$

$$= \sum_{(x,y)\in D} b(x,y) - p(w) + p(z)$$

$$\textbf{(b)} \sum_{(x,y)\in\overset{\circ}{D}} b^p(x,y) = \sum_{(x,y)\in\overset{\circ}{D}} b(x,y) - \sum_{(x,y)\in\overset{\circ}{D}} p(x) + \sum_{(x,y)\in\overset{\circ}{D}} p(y) =$$

$$= \sum_{(x,y)\in\overset{\circ}{D}} b(x,y)$$

Observaţia 8.2. Prima parte a Proprietăţii 8.2 implică faptul că potenţialele nodurilor nu modifică drumul minim de la un nod oarecare w la un nod oarecare z, deoarece potenţialele măresc distanţa de la w la z cu $p(z) - p(w)$, care este o constantă. Din partea a doua a Proprietăţii 8.2 rezultă că dacă $\overset{\circ}{D}$ este un circuit de cost negativ în raport cu costurile $b(x,y)$, atunci $\overset{\circ}{D}$ este de asemenea un circuit de cost negativ în raport cu costurile reduse $b^p(x,y)$.

Teorema 8.3. (Condiţiile de optimalitate cu costuri reduse) *Un flux admisibil f^* este un flux de cost minim dacă şi numai dacă există potenţialele p astfel încât:*

$$b^p(x,y) \geq 0 \text{ pentru toate arcele } (x,y) \text{ din reţeaua reziduală } \tilde{G}(f^*). \tag{8.2}$$

Demonstraţie. Vom demonstra echivalenţa dintre condiţiile de optimalitate cu costuri reduse şi condiţiile de optimalitate cu circuite de cost negativ.

Fie f^* un flux care satisface condiţiile de optimalitate cu costuri reduse. Rezultă că $\sum_{(x,y)\in\overset{\circ}{D}} b^p(x,y) \geq 0$, pentru orice circuit $\overset{\circ}{D}$ din reţeaua reziduală $\tilde{G}(f^*)$. Conform Proprietăţii 8.2 (b),

$$\sum_{(x,y)\in\overset{\circ}{D}} b(x,y) = \sum_{(x,y)\in\overset{\circ}{D}} b^p(x,y) \geq 0,$$

deci în reţeaua reziduală $\tilde{G}(f^*)$ nu există circuite de cost negativ.

Invers, fie f^* un flux care satisface condiţiile de optimalitate cu circuite de cost negativ. Fie $d(\cdot)$ vectorul distanţelor de la nodul 1 la celelalte noduri în reţeaua reziduală $\tilde{G}(f^*)$. Cum $\tilde{G}(f^*)$ nu conţine circuite de cost negativ, distanţele îndeplinesc următoarea condiţie:

$$d(y) \leq d(x) + b(x,y) \text{ pentru toate arcele } (x,y) \text{ din } \tilde{G}(f^*)$$

sau, echivalent, $b(x,y) - (-d(x)) + (-d(y)) \geq 0$, sau $b^p(x,y) \geq 0$, cu $p = -d$. Deci, f^* satisface condiţiile de optimalitate cu costuri reduse. ■

Definiţia 8.1. Pentru un flux admisibil f, potenţialele p care îndeplinesc condiţia $b^p(x,y) \geq 0$, pentru toate arcele (x,y) din $\tilde{G}(f)$, se numesc *potenţiale optime.*

8.3.3 Condiţiile de optimalitate cu ecarturi complementare

Teoremele 8.1 şi 8.3 constau în condiţii de optimalitate formulate în reţeaua reziduală. Condiţiile de optimalitate pot fi reformulate în reţeaua originară astfel:

Teorema 8.4. (Condiţiile de optimalitate cu ecarturi complementare) *Un flux admisibil f^* este un flux de cost minim dacă şi numai dacă există potenţialele p astfel încât pentru orice arc $(x,y) \in A$ avem:*

$$\text{Dacă } b^p(x,y) > 0 \text{ atunci } f^*(x,y) = 0. \tag{8.3.a}$$

$$\text{Dacă } 0 < f^*(x,y) < c(x,y) \text{ atunci } b^p(x,y) = 0. \tag{8.3.b}$$

$$\text{Dacă } b^p(x,y) < 0 \text{ atunci } f^*(x,y) = c(x,y). \tag{8.3.c}$$

Demonstraţie. Vom arăta echivalenţa dintre condiţiile de optimalitate cu ecarturi complementare şi condiţiile de optimalitate cu costuri reduse.

Fie f^* un flux şi p un vector de potenţiale care satisfac condiţiile de optimalitate cu costuri reduse. Pentru orice arc $(x,y) \in A$ există trei posibilităţi:

1. Dacă $b^p(x,y) > 0$ atunci arcul (y,x) nu poate face parte din reţeaua reziduală deoarece $b^p(y,x) = -b^p(x,y) < 0$, ceea ce contrazice condiţiile de optimalitate cu costuri reduse. Rezultă că $f^*(x,y) = 0$.

2. Dacă $0 < f^*(x,y) < c(x,y)$ atunci reţeaua reziduală conţine ambele arce: (x,y) şi (y,x). Conform condiţiilor de optimalitate cu costuri reduse $b^p(x,y) \geq 0$ şi $b^p(y,x) \geq 0$. Cum $b^p(y,x) = -b^p(x,y)$, rezultă că $b^p(x,y) = b^p(y,x) = 0$.

3. Dacă $b^p(x,y) < 0$ atunci arcul (x,y) nu poate face parte din reţeaua reziduală, deoarece ar contrazice condiţiile de optimalitate cu costuri reduse. Rezultă că $f^*(x,y) = c(x,y)$.

Invers, fie f^* un flux şi p un vector de potenţiale care îndeplinesc condiţiile de optimalitate cu ecarturi complementare. Fie (x,y) un arc oarecare din A. Există trei posibilităţi şi vom arăta că în nici una dintre ele nu există arce de cost redus negativ în reţeaua reziduală.

1. Dacă $b^p(x,y) > 0$ atunci din (8.3.a) rezultă că $f^*(x,y) = 0$, deci reţeaua reziduală nu va conţine arcul (y,x), ci doar arcul (x,y) cu costul redus $b^p(x,y) > 0$.

2. Dacă $0 < f^*(x,y) < c(x,y)$ atunci reţeaua reziduală va conţine ambele arce (x,y) şi (y,x). Din (8.3.b) rezultă că $b^p(x,y) = 0$. Cum $b^p(y,x) = -b^p(x,y)$, avem că $b^p(x,y) = b^p(y,x) = 0$.

3. Dacă $b^p(x,y) < 0$ atunci din (8.3.c) rezultă că $f^*(x,y) = c(x,y)$, deci reţeaua reziduală nu va conţine arcul (x,y), ci doar arcul (y,x), care are costul redus $b^p(y,x) = -b^p(x,y) > 0$. ■

8.4 Dualitatea pentru problema fluxului de cost minim

Oricărei probleme de programare liniară, pe care o vom numi problemă primară, îi putem asocia o altă problemă de programare liniară care depinde de problema primară şi pe care o vom numi problemă duală. Pentru a obţine problema duală a unei probleme primare, asociem câte o variabilă duală cu fiecare constrângere cu excepţia celor de forma:

$$0 \leq f(x,y), \;\; (x,y) \in A.$$

Pentru problema fluxului de cost minim (8.1), asociem variabila $p(x)$ cu constrângerea de conservare a fluxului corespunzătoare nodului x, iar variabila $q(x,y)$ cu constrângerea de mărginire superioară a fluxului pe arcul (x,y). Duala problemei fluxului de cost minim este următoarea:

$$\text{maximizează } \; w(p,q) = \sum_{x \in N} v(x)p(x) - \sum_{(x,y) \in A} c(x,y) \cdot q(x,y) \tag{8.4.a}$$

$$p(x) - p(y) - q(x,y) \leq b(x,y), \; (x,y) \in A \tag{8.4.b}$$

$$q(x,y) \geq 0, \; (x,y) \in A \tag{8.4.c}$$

Teorema 8.5. (Teorema slabă de dualitate) *Fie $z(f)$ valoarea funcţiei obiectiv a problemei fluxului de cost minim pentru un flux admisibil f şi fie $w(p,q)$ valoarea funcţiei obiectiv a problemei duale pentru o soluţie admisibilă (p,q). Atunci $w(p,q) \leq z(f)$.*

Demonstraţie. Dacă înmulţim inecuaţiile (8.4.b) cu $f(x,y)$ şi însumăm după $(x,y) \in A$ obţinem:

$$\sum_{(x,y)\in A} (p(x) - p(y))\, f(x,y) - \sum_{(x,y)\in A} q(x,y)\cdot f(x,y) \leq \sum_{(x,y)\in A} b(x,y)\cdot f(x,y) \quad (8.5)$$

Dar,

$$\sum_{(x,y)\in A} (p(x) - p(y))\, f(x,y) = \sum_{(x,y)\in A} p(x)\cdot f(x,y) - \sum_{(y,x)\in A} p(x)\cdot f(y,x) =$$

$$= \sum_{x\in N} \sum_{y\,|\,(x,y)\in A} p(x)\cdot f(x,y) - \sum_{x\in N} \sum_{y\,|\,(y,x)\in A} p(x)\cdot f(y,x) =$$

$$= \sum_{x\in N} p(x) \left(\sum_{y\,|\,(x,y)\in A} f(x,y) - \sum_{y\,|\,(y,x)\in A} f(y,x) \right) = \sum_{x\in N} p(x)\cdot v(x).$$

Ţinând cont de această relaţie, inegalitatea (8.5) poate fi scrisă şi sub forma:

$$\sum_{x\in N} p(x)\cdot v(x) - \sum_{(x,y)\in A} q(x,y)\cdot f(x,y) \leq z(f). \quad (8.6)$$

Dar $q(x,y) \geq 0$ şi $f(x,y) \leq c(x,y),\ (x,y) \in A$, deci din (8.6) obţinem:

$$\sum_{x\in N} p(x)\cdot v(x) - \sum_{(x,y)\in A} q(x,y)\cdot c(x,y) \leq z(f)$$

sau, echivalent

$$w(p,q) \leq z(f) \qquad \blacksquare$$

Putem rescrie inegalitatea (8.4.b) în forma echivalentă

$$q(x,y) \geq -b^p(x,y) \quad (8.7)$$

Deoarece coeficientul lui $q(x,y)$ din funcţia obiectiv (8.4.a) este $-c(x,y)$ şi căutăm să maximizăm valoarea funcţiei obiectiv, rezultă că $q(x,y)$ va avea o valoare cât se poate de mică. Ţinând cont de (8.4.c) şi (8.7), obţinem că

$$q(x,y) = \max\{0, -b^p(x,y)\} \quad (8.8)$$

Înlocuind $q(x,y)$ în (8.4.a) obţinem:

$$\text{maximizează } w(p) = \sum_{x\in N} v(x)p(x) - \sum_{(x,y)\in A} \max\{0, -b^p(x,y)\}c(x,y). \quad (8.9)$$

Deci, problema dual ă constă în a determina un vector p care să maximizeze (8.9) (deoarece putem determina valorile $q(x,y)$ folosind relaţia (8.8)).

Teorema 8.6. (Teorema tare de dualitate) *Dacă pentru problema fluxului de cost minim există un flux admisibil, atunci există un flux optim f^* şi pentru problema dual ă există o soluţie p astfel încât $z(f^*) = w(p)$.*

Demonstraţie. Fie f^* un flux de cost minim. Conform Teoremei 8.4, există potenţialele p astfel încât sunt îndeplinite condiţiile de optimalitate cu ecarturi complementare:

1. dacă $b^p(x,y) > 0$ atunci $f^*(x,y) = 0$
2. dacă $0 < f^*(x,y) < c(x,y)$ atunci $b^p(x,y) = 0$
3. dacă $b^p(x,y) < 0$ atunci $f^*(x,y) = c(x,y)$

Vom arăta că în toate cele trei cazuri este adevărată egalitatea:

$$-b^p(x,y)f(x,y) = \max\{0, -b^p(x,y)\}c(x,y). \quad (8.10)$$

În primele două cazuri, ambii membri ai egalităţii sunt nuli; iar dacă $b^p(x,y) < 0$ atunci $\max\{0, b^p(x,y)\} = -b^p(x,y)$, deci şi în acest caz relaţia este adevărată.

Înlocuind (8.10) în funcţia obiectiv (8.9) a problemei duale obţinem:

$$w(p) = \sum_{x\in N} v(x)\cdot p(x) + \sum_{(x,y)\in A} b^p(x,y)\cdot f^*(x,y) = \sum_{x\in N} v(x)\cdot p(x) +$$

$$+ \sum_{(x,y)\in A} b(x,y)\cdot f^*(x,y) - \sum_{(x,y)\in A} p(x)\cdot f^*(x,y) + \sum_{(x,y)\in A} p(y)f^*(x,y) =$$

$$= \sum_{(x,y)\in A} b(x,y)\cdot f^*(x,y) + \sum_{x\in N} v(x)p(x) - \sum_{x\in N}\sum_{x\,|\,(x,y)\in A} p(x)\cdot f^*(x,y) +$$

$$+ \sum_{x\in N}\sum_{x\,|\,(y,x)\in A} p(x)f^*(y,x) = \sum_{(x,y)\in A} b(x,y)\cdot f^*(x,y) = z(f^*). \qquad \blacksquare$$

Teorema 8.7. *Dacă f este un flux admisibil şi p sunt nişte potenţiale astfel încât $z(f) = w(p)$ atunci f şi p îndeplinesc condiţiile de optimalitate cu ecarturi complementare.*

Demonstraţie. Deoarece $z(f) = w(p)$, folosind relaţia (8.9) rezultă că:

$$\sum_{(x,y)\in A} b(x,y)f(x,y) = \sum_{x\in N} v(x)\cdot p(x) - \sum_{(x,y)\in A} \max\{0, -b^p(x,y)\}c(x,y).$$

Deoarece

$$\sum_{x\in N} v(x)\cdot p(x) - \sum_{(x,y)\in A} b(x,y)\cdot f(x,y) = -\sum_{(x,y)\in A} b^p(x,y)\cdot f(x,y)$$

(vezi demonstraţia Teoremei 8.6), rezultă că

$$\sum_{(x,y)\in A} \max\{0,-b^p(x,y)\}c(x,y) = \sum_{(x,y)\in A} -b^p(x,y)\cdot f(x,y). \tag{8.11}$$

Dar $\max\{0,-b^p(x,y)\} \geq -b^p(x,y),\ \ (x,y)\in A$ şi $c(x,y)\geq f(x,y),\ \ (x,y)\in A$, deci egalitatea (8.11) poate fi îndeplinită doar dacă

$$\max\{0,-b^p(x,y)\}\cdot c(x,y) = -b^p(x,y)\cdot f(x,y),\ \ (x,y)\in A. \tag{8.12}$$

Considerăm următoarele trei cazuri:

1. Dacă $b^p(x,y)>0$ atunci $\max\{0,-b^p(x,y)\}=0$ şi din relaţia (8.12) rezultă că $f(x,y)=0$. Deci este îndeplinită condiţia (8.3.a)
2. Dacă $0<f(x,y)<c(x,y)$ atunci $b^p(x,y)=0$, deci este verificată condiţia (8.3.b).
3. Dacă $b^p(x,y)<0$ atunci $\max\{0,-b^p(x,y)\}=-b^p(x,y)$, deci $f(x,y)=c(x,y)$ şi este îndeplinită condiţia (8.3.c).

Deci, f şi p îndeplinesc condiţiile de optimalitate cu ecarturi complementare.

■

Proprietatea 8.8. *Dacă f^* este un flux de cost minim şi p este o soluţie optimă a problemei duale atunci f^* şi p îndeplinesc condiţiile de optimalitate cu ecarturi complementare.*
Demonstraţie. Din Teorema 8.6 rezultă că $z(f^*)=w(p)$ şi conform Teoremei 8.7, f^* şi p îndeplinesc condiţiile de optimalitate cu ecarturi complementare.■

8.5 Legătura dintre fluxurile de cost minim şi potenţialele optime

Fiind dat un flux de cost minim f^*, putem determina potenţiale optime rezolvând o problemă de drum minim. Deoarece f^* este un flux de cost minim, în reţeaua reziduală $\tilde{G}(f^*)$ nu există circuite de cost negativ, conform Teoremei 8.1. Fie d vectorul distanţelor minime de la nodul 1 la celelalte noduri în $\tilde{G}(f^*)$ considerând costurile arcelor ca fiind lungimile lor. Rezultă că

$$d(y)\leq d(x)+b(x,y) \text{ pentru toate arcele } (x,y) \text{ din } \tilde{G}(f^*).$$

Fie $p = -d$. Obţinem

$b^p(x, y) = b(x, y) - p(x) + p(y) \geq 0$, pentru toate arcele (x, y) din $\tilde{G}(f^*)$.

Deci, conform Teoremei 8.3, p este un vector de potenţiale optime.

Fiind dat un vector p de potenţiale optime, putem determina un flux de cost minim f^* rezolvând o problemă de flux maxim. Împărţim arcele din A în trei categorii, în funcţie de costurile reduse:

1. Dacă $b^p(x, y) > 0$ atunci pentru a îndeplini condiţia de optimalitate (8.3.a) se efectuează atribuirea $f^*(x, y) = 0$ şi eliminăm arcul (x, y).
2. Dacă $b^p(x, y) < 0$ atunci pentru a îndeplini condiţia de optimalitate (8.3.c) se atribuie $f^*(x, y) = c(x, y)$; se micşorează $v(x)$ cu $c(x, y)$, se măreşte $v(y)$ cu $c(x, y)$ şi se elimină arcul (x, y).
3. Dacă $b^p(x, y) = 0$ atunci $f(x, y)$ poate avea orice valoare cuprinsă între 0 şi $c(x, y)$.

Fie $G' = (N, A')$ reţeaua astfel obţinută şi v' ofertele/cererile nodurilor modificate. Pentru a obţine un flux de cost minim în G trebuie să determinăm un flux admisibil în G'; problemă care se reduce la o problemă de flux maxim.

8.6 Algoritmi pseudopolinomiali pentru fluxul de cost minim

8.6.1 Algoritmul Klein de eliminare a circuitelor de cost negativ

Acest algoritm se bazează pe condiţiile de optimalitate cu circuite de cost negativ. Algoritmul Klein de eliminare a circuitelor de cost negativ determină mai întâi un flux admisibil rezolvând o problemă de flux maxim. Apoi se determină circuitele de cost negativ din reţeaua reziduală şi se măreşte fluxul de-a lungul lor, ceea ce duce la eliminarea lor şi la micşorarea costului fluxului. Algoritmul se termină atunci când în reţeaua reziduală nu mai există circuite de cost negativ, ceea ce conform Teoremei 8.1, implică faptul că fluxul astfel determinat este un flux de cost minim. Algoritmul Klein de eliminare a circuitelor de cost negativ este următorul:

```
(1) PROGRAM KLEIN-ELIMINARE-CIRCUITE;
(2) BEGIN
(3)     se determină un flux admisibil f;
```

(4) WHILE în $\tilde{G}(f)$ există circuite de cost negativ DO
(5) BEGIN
(6) se determină un circuit de cost negativ $\overset{\circ}{D}$;
(7) se determină $r(\overset{\circ}{D}) = \min\{r(x,y) \,|\, (x,y) \in \overset{\circ}{D}\}$;
(8) se efectuează mărirea de flux de-a lungul circuitului $\overset{\circ}{D}$;
(9) se actualizează $\tilde{G}(f)$;
(10) END;
(12) END.

Exemplul 8.1. Ilustrăm algoritmul Klein de eliminare a circuitelor de cost negativ pe reţeaua din figura 8.2 (a), în care $s = 1$ şi $t = 4$. În figura 8.2 (a) avem un flux admisibil al cărui cost este $\sum_{(x,y)\in A} b(x,y) \cdot f(x,y) = 2 \cdot 2 + 4 \cdot 2 + 1 \cdot 0 + 4 \cdot 2 + 2 \cdot 2 = 24$. Figura 8.2(b) reprezintă reţeaua reziduală care conţine un circuit de cost negativ $\overset{\circ}{D} = (2, 3, 4, 2)$. Costul circuitului $\overset{\circ}{D}$ este $1 + 2 + (-4) = -1$, iar capacitatea sa reziduală este $r(\overset{\circ}{D}) = \min\{1, 1, 2\} = 1$.

Fig.8.2.

Algoritmul va mări cu o unitate fluxul de-a lungul circuitului $\overset{\circ}{D}$, iar reţeaua reziduală actualizată după mărirea de flux este ilustrată în figura 8.2(c). Deoarece această reţea reziduală nu mai conţine circuite de cost negativ, algoritmul se termină, iar capacităţile reziduale astfel obţinute sunt capacităţi reziduale optime. Fluxurile pe arce corespunzătoare acestor capacităţi reziduale optime sunt ilustrate în figura 8.2 (d) şi se calculează după următoarele reguli:

- dacă $c(x,y) \geq r(x,y)$ atunci $f(x,y) = c(x,y) - r(x,y)$ şi $f(y,x) = 0$
- dacă $c(x,y) < r(x,y)$ atunci $f(x,y) = 0$ şi $f(y,x) = r(x,y) - c(x,y)$.

Costul fluxului de cost minim astfel determinat este $\sum_{(x,y)\in A} b(x,y) \cdot f(x,y) = 2 \cdot 2 + 4 \cdot 2 + 1 \cdot 1 + 4 \cdot 1 + 2 \cdot 3 = 23$.

Teorema 8.9. (Teorema valorilor întregi) *Dacă arcele au capacităţi întregi şi cererile/ofertele nodurilor sunt întregi atunci algoritmul Klein de eliminare a circuitelor de cost negativ se termină după un număr finit de iteraţii şi la sfârşitul execuţiei lui se obţine un flux de cost minim cu valori întregi.*
Demonstraţie. Demonstraţia se face prin inducţie după numărul de iteraţii. Determinarea unui flux admisibil se reduce la a rezolva o problemă de flux maxim. Deoarece datele problemei sunt întregi, conform Teoremei 5.8 rezultă că într-un număr finit de iteraţii se va determina un flux maxim cu valori întregi. Deoarece fluxul iniţial are valori întregi, rezultă că toate arcele au capacităţi reziduale întregi. La fiecare iteraţie se măreşte cu $r(\overset{\circ}{D})$ fluxul de-a lungul circuitului de cost negativ $\overset{\circ}{D}$. Cum $r(\overset{\circ}{D})$ este minimul capacităţilor reziduale ale arcelor care formează circuitul $\overset{\circ}{D}$, rezultă că $r(\overset{\circ}{D})$ este o valoare întreagă mai mare sau egală cu 1. Deci, după mărirea de flux capacităţile reziduale vor fi tot întregi, iar costul fluxului va scădea cu cel puţin o unitate. În consecinţă, la sfârşitul algoritmului capacităţile reziduale optime vor fi întregi, deci fluxul de cost minim va avea valori întregi.

Costul oricărui flux din reţeaua G este cuprins între $-m\overline{b}\overline{c}$ şi $m\overline{b}\overline{c}$ deoarece $-\overline{b} \leq b(x,y) \leq \overline{b}$ şi $f(x,y) \leq \overline{c},\ \forall (x,y) \in A$. După fiecare iteraţie a algoritmului Klein de eliminare a circuitelor de cost negativ, costul fluxului scade cu cel puţin o unitate, pentru că $\left(\sum_{(x,y)\in \overset{\circ}{D}} b(x,y)\right) r(\overset{\circ}{D}) \leq -1$. Deoarece am presupus că toate datele problemei sunt întregi, rezultă că după $O(m\overline{b}\overline{c})$ iteraţii algoritmul se termină. ■

Teorema 8.10. *Complexitatea algoritmului Klein de eliminare a circuitelor de cost negativ este $O(nm^2\overline{b}\overline{c})$.*
Demonstraţie. Algoritmul efectuează $O(m\overline{b}\overline{c})$ iteraţii (vezi demonstraţia Teoremei 8.9), iar complexitatea unei iteraţii este dată de complexitatea algoritmului folosit pentru determinarea unui circuit de cost negativ. Deoarece determinarea unui circuit de cost negativ se poate face în $O(nm)$ modificând algoritmul Bellman-Ford pentru drumuri minime (vezi Observaţia 4.3) rezultă că algoritmul Klein de eliminare a circuitelor de cost negativ are complexitatea $O(nm^2\overline{b}\overline{c})$. ■

Algoritmul prezentat mai sus este un algoritm generic deoarece nu precizează nici o regulă după care se alege circuitul de cost negativ de-a lungul căruia se măreşte fluxul. Se poate alege circuitul de cost negativ cu costul mediu minim sau circuitul format doar din arce cu costurile reduse negative sau circuitul care aduce cea mai mare micşorare a costului fluxului sau circuitul de cost negativ cu capacitatea reziduală suficient de mare etc. Impunând anumite reguli de alegere a circuitului de cost negativ se obţin algoritmi polinomiali, deci mai performanţi decât algoritmul Klein de eliminare a circuitelor de cost negativ care este pseudopolinomial. Unul dintre aceşti algoritmi va fi prezentat în capitolul următor.

8.6.2 Algoritmul Busaker - Gowen al drumurilor minime succesive

Înainte de a prezenta acest algoritm trebuie să introducem conceptul de pseudoflux. Un *pseudoflux* este o funcţie $f : A \to \Re_+$ care satisface doar condiţia de mărginire:

$$0 \leq f(x,y) \leq c(x,y), \ (x,y) \in A.$$

Deoarece un pseudoflux f nu satisface şi condiţia de conservare a fluxului, are sens să definim excesul /deficitul unui nod x astfel:

$$e(x) = v(x) + \sum_{y \mid (y,x) \in A} f(y,x) - \sum_{y \mid (x,y) \in A} f(x,y), \ x \in N$$

Dacă pentru un nod x, $e(x) > 0$ atunci $e(x)$ se numeşte *excesul* nodului x, dacă $e(x) < 0$ atunci $e(x)$ se numeşte *deficitul* nodului x, iar dacă $e(x) = 0$ atunci spunem că nodul x este *echilibrat.* Notăm cu S mulţimea nodurilor cu exces, deci $S = \{x \in N \mid e(x) > 0\}$ şi cu T mulţimea nodurilor cu deficit, deci $T = \{x \in N \mid e(x) < 0\}$. Deoarece $\sum_{x \in N} e(x) = \sum_{x \in N} v(x) = 0$, rezultă că $\sum_{x \in S} e(x) = -\sum_{x \in T} e(x)$. Deci, dacă în reţea există un nod cu exces atunci trebuie să existe şi un nod cu deficit.

Reţeaua reziduală corespunzătoare unui pseudoflux se defineşte analog cu reţeaua reziduală corespunzătoare unui flux. Un flux este un caz particular de pseudoflux (pentru $S = T = \emptyset$).

Teorema 8.11. *Fie f un pseudoflux care satisface condiţiile de optimalitate cu costuri reduse în raport cu potenţialele p. Fie d vectorul distanţelor de la s la celelalte noduri în $\tilde{G}(f)$ considerând $b^p(x,y)$ ca fiind lungimea arcului (x,y). Au loc următoarele două proprietăţi:*
(a) Pseudofluxul f satisface condiţiile de optimalitate cu costuri reduse şi în raport cu potenţialele $p' = p - d$.
(b) Costurile reduse $b^{p'}(x,y)$ sunt nule pentru toate arcele (x,y) care aparţin unui drum minim de la s la oricare alt nod.
Demonstraţie. (a) Deoarece f satisface condiţiile de optimalitate în raport cu potenţialele p rezultă că $b^p(x,y) \geq 0$ pentru toate arcele (x,y) din reţeaua reziduală $\tilde{G}(f)$. Cum d este vectorul distanţelor de la s la celelalte noduri în $\tilde{G}(f)$ considerând $b^p(x,y)$ ca fiind lungimea arcului (x,y) rezultă că

$$d(y) \leq d(x) + b^p(x,y) \text{ pentru orice arc } (x,y) \text{ din } \tilde{G}(f).$$

Echivalent,

$$d(y) \leq d(x) + b(x,y) - p(x) + p(y)$$

sau

$$b(x,y) - (p(x) - d(x)) + (p(y) - d(y)) \geq 0$$

sau

$$b^{p'}(x,y) \geq 0.$$

Deci, f îndeplineşte condiţiile de optimalitate cu costuri reduse în raport cu potenţialele p'.

(b) Fie (x,y) un arc oarecare care aparţine unui drum minim de la s la un alt nod. Rezultă că

$$d(y) = d(x) + b^p(x,y) = d(x) + b(x,y) - p(x) + p(y)$$

sau

$$b(x,y) - (p(x) - d(x)) + (p(y) - d(y)) = 0$$

sau

$$b^{p'}(x,y) = 0. \qquad \blacksquare$$

Teorema 8.12. *Fie f un pseudoflux care satisface condiţiile de optimalitate cu costuri reduse şi f' pseudofluxul obţinut din f prin mărirea fluxului de-a lungul unui drum minim de la nodul s la un alt nod k. Atunci f' satisface condiţiile de optimalitate cu costuri reduse.*

Demonstraţie. Fie potenţialele p şi p' definite ca în Teorema 8.11. Conform Teoremei 8.11, $b^{p'}(x,y) = 0$ pentru orice arc (x,y) care aparţine unui drum minim de la s la k. Mărirea fluxului de-a lungul drumului minim de la s la k poate introduce arce inverse (y,x), în reţeaua reziduală. Dar $b^{p'}(y,x) = -b^{p'}(x,y)$. Deoarece $b^{p'}(x,y) = 0$ rezultă că $b^{p'}(y,x) = 0$, deci şi arcele inverse, care apar în reţeaua reziduală în urma măririi fluxului, satisfac condiţiile de optimalitate cu costuri reduse. ■

Algoritmul Busaker - Gowen al drumurilor minime succesive (BGDMS) este următorul:

(1) PROGRAM BGDMS;
(2) BEGIN
(3) $f := 0$;
(4) $p := 0$;
(5) FOR toate nodurile $x \in N$ DO $e(x) := v(x)$;
(6) $S := \{x \in N \mid e(x) > 0\}$;
(7) $T := \{x \in N \mid e(x) < 0\}$;
(8) WHILE $S \neq \emptyset$ DO
(9) BEGIN
(10) se selectează un nod $s \in S$ şi un nod $t \in T$;

(11) se determină distanţele $d(\cdot)$ de la s la celelalte noduri în $\tilde{G}(f)$ considerând costurile reduse ca fiind lungimile arcelor;
(12) fie D drumul minim de la nodul s la nodul t;
(13) $p := p - d$;
(14) $r(D) = \min\{e(s), -e(t), \min\{r(x,y) \mid (x,y) \in D\}\}$;
(15) se măreşte fluxul cu $r(D)$ de-a lungul drumului D;
(16) se actualizează f, $\tilde{G}(f), S, T$ şi costurile reduse;
(17) END;
(18) END.

Exemplul 8.2. Ilustrăm algoritmul BGDMS pe reţeaua din figura 8.3(a). Reţeaua reziduală obţinută după iniţializările din liniile (3)-(7) este reprezentată în figura 8.3(b). Deoarece $S = \{1\}$ şi $T = \{4\}$, va trebui să determinăm drumul minim de la nodul 1 la nodul 4. Vectorul distanţelor este $d = (0, 2, 3, 5)$ şi drumul minim de la 1 la 4 este $D = (1, 2, 3, 4)$. Figura 8.3 (c) ilustrează reţeaua reziduală după actualizarea potenţialelor şi a costurilor reduse în urma măririi cu $r(D) = \min\{4, -(-4), \min\{2, 1, 3\}\} = 1$ a fluxului pe drumul D. În continuare $S = \{1\}$ şi $T = \{4\}$, deci vom determina vectorul distanţelor de la nodul 1 la celelalte noduri: $d = (0, 0, 1, 1)$. Drumul minim de la 1 la 4 este $D = (1, 2, 4)$. Se măreşte fluxul cu $r(D) = 1$ de-a lungul drumului D. În figura 8.3 (d) este reprezentată reţeaua reziduală după actualizarea potenţialelor şi a costurilor reduse. În continuare $S = \{1\}$ şi $T = \{4\}$, iar vectorul distanţelor de la nodul 1 la celelalte noduri este $d = (0, 0, 0, 0)$. Se măreşte cu 2 unităţi fluxul pe drumul $D = (1, 3, 4)$. Reţeaua reziduală astfel obţinută este reprezentată în figura 8.3 (e). Deoarece $S = T = \emptyset$, algoritmul se termină. Revenind de la capacităţile reziduale la flux obţinem reţeaua din figura 8.3 (f). Costul fluxului este 23.

Fig.8.3.

Teorema 8.13. *Algoritmul BGDMS are complexitatea* $O\left(n\bar{c} \cdot D\left(n, m\right)\right)$, *unde* $D(n, m)$ *este complexitatea algoritmului de determinare a drumului minim de la un nod dat la celelalte noduri într-o reţea cu lungimile arcelor pozitive.*

Demonstraţie. Iniţial, fluxul nul îndeplineşte condiţiile de optimalitate cu costuri reduse în raport cu potenţialele nule deoarece $b^p(x, y) = b(x, y) \geq 0$ conform Ipotezei 4. Cât timp există un nod cu exces (deci există şi un nod cu deficit), algoritmul poate selecta un nod s de la care să determine drumurile minime către celelalte noduri în reţeaua reziduală considerând costurile reduse ca fiind lungimile arcelor. Deci, lungimile arcelor sunt pozitive. Mărirea fluxului de-a lungul drumului minim D de la nodul s la nodul t duce la micşorarea excesului nodului s şi a deficitului nodului t cu cel puţin o unitate. Deoarece iniţial $|e(x)| \leq \bar{c},\ x \in N$, rezultă că după $O(n\bar{c})$ iteraţii toate nodurile vor fi echilibrate, adică

pseudofluxul f, care îndeplineşte condiţiile de optimalitate cu costuri reduse, este un flux de cost minim. Deci, dacă $D(n,m)$ este complexitatea algoritmului folosit pentru determinarea drumului minim, atunci complexitatea algoritmului BGDMS este $O\left(n\bar{c}D\left(n,m\right)\right)$. ■

Dacă pentru rezolvarea problemei de drum minim se foloseşte algoritmul Dijkstra cu heap-uri Fibonacci, complexitatea algoritmului BGDMS este $O\left(n\bar{c}\left(m+n\log n\right)\right)$.

8.6.3 Algoritmul primal - dual Ford - Fulkerson

Algoritmul primal-dual Ford-Fulkerson se aseamănă cu algoritmul BGDMS prin faptul că pe parcursul iteraţiilor intermediare menţine un pseudoflux care îndeplineşte condiţiile de optimalitate cu costuri reduse şi care, treptat, este transformat în flux. Spre deosebire de algoritmul prezentat în paragraful precedent care mărea fluxul de-a lungul unui drum minim, algoritmul primal - dual Ford-Fulkerson rezolvă o problemă de flux maxim (deci măreşte fluxul pe toate drumurile minime).

Mai întâi, problema fluxului de cost minim este transformată într-o problemă cu un singur nod s' cu exces şi un singur nod t' cu deficit în reţeaua extinsă $G'=(N',A',c',b')$, unde $N'=N\cup\{s',t'\}$, $A'=A\cup A_{s'}\cup A_{t'}$, $A_{s'}=\{(s',x)\mid x\in N \text{ şi } v(x)>0\}$, $A_{t'}=\{(x,t')\mid x\in N \text{ şi } v(x)<0\}$, $v'(s')=\sum_{x\in N\mid v(x)>0}v(x)$, $v'(x)=0$, $x\in N$, $v'(t')=-v'(s')$, $b'(s',x)=0$, $(s',x)\in A_{s'}$, $b'(x,y)=b(x,y)$, $(x,y)\in A$, $b'(x,t')=0$, $(x,t')\in A_{t'}$, $c'(s',x)=v(x)$, $(s',x)\in A_{s'}$, $c'(x,y)=c(x,y)$, $(x,y)\in A$, $c'(x,t')=-v(x)$, $(x,t')\in A_{t'}$.

Algoritmul primal-dual Ford-Fulkerson rezolvă o problemă de flux maxim într-un subgraf al reţelei reziduale $\tilde{G}'(f)$ notat cu $\tilde{G}'_0(f)$ şi numit *reţea admisibilă.* Definim reţeaua admisibilă $\tilde{G}'_0(f)$ corespunzătoare pseudofluxului f care îndeplineşte condiţiile de optimalitate cu costuri reduse în raport cu potenţialele p ca fiind formată din arcele din $\tilde{G}'(f)$ care au costul redus nul. Evident, un drum oarecare de la s' la t' în $\tilde{G}'_0(f)$ este un drum minim de la s' la t' în $\tilde{G}'(f)$.

Algoritmul primal-dual Ford-Fulkerson este următorul:

```
(1) PROGRAM PRIMAL-DUAL-FF;
(2) BEGIN
(3)     f := 0;
(4)     p := 0;
(5)     e'(s') := v'(s');
(6)     e'(t') := v'(t');
(7)     WHILE e'(s') > 0 DO
(8)     BEGIN
(9)         se determină vectorul distanţelor minime d(·) de la s' la
            celelalte noduri în G̃'(f) în raport cu costurile reduse b'^p(x,y);
```

(10) $p := p - d$;
(11) se determină reţeaua admisibilă $\tilde{G}'_0(f)$;
(12) se determină un flux maxim de la s' la t' în $\tilde{G}'_0(f)$;
(13) se actualizează $e'(s')$, $e'(t')$ şi $\tilde{G}'(f)$;
(14) END;
(15) END.

Exemplul 8.3. Ilustrăm algoritmul primal-dual Ford-Fulkerson pe reţeaua din figura 8.4 (a).

Fig.8.4.

În figura 8.4 (b) este reprezentată reţeaua extinsă, în care determinăm vectorul distanţelor minime de la s' la celelalte noduri $d = (d(s'), d(1), d(2), d(3), d(4), d(t'))$ = (0, 0, 0, 1, 2, 1). În figura 8.4 (c) este reprezentată reţeaua extinsă după actualizarea potenţialelor şi a costurilor reduse, iar figura 8.4 (d) ilustrează reţeaua admisibilă. Fluxul maxim de la s' la t' în reţeaua admisibilă se obţine mărind fluxul cu 2 unităţi pe drumul $(s', 1, 3, t')$. Reţeaua admisibilă astfel obţinută este reprezentată în figura 8.4 (e), iar reţeaua extinsă în figura 8.4 (f). Vectorul distanţelor minime de la s' la celelalte noduri este d = (0, 1, 0, 1, 0, 1). În figura 8.4 (g) este reprezentată reţeaua extinsă după actualizarea potenţialelor şi a costurilor reduse, iar figura 8.4 (h) ilustrează reţeaua admisibilă. În această reţea fluxul maxim se obţine prin mărirea fluxului cu o unitate pe drumul $(s', 2, 4, t')$. Reţeaua admisibilă astfel obţinută este reprezentată în figura 8.4 (i), iar în figura 8.4 (j) este reţeaua extinsă. Cum $e'(s') = 0$, algoritmul se termină şi în figura 8.4 (k) este reprezentat fluxul de cost minim obţinut plecând de la capacităţile reziduale optime din figura 8.4 (j).

Teorema 8.14. *Algoritmul primal-dual Ford-Fulkerson are complexitatea* $O\left(\min\{n\overline{c},\, n\overline{b}\}\,(D(n,m) + F(n,m,\overline{c}))\right)$, *unde* $D(n,m)$ *şi* $F(n,m,\overline{c})$ *sunt complexităţile algoritmilor de determinare a drumului minim şi, respectiv, a fluxului maxim.*

Demonstraţie. La fiecare iteraţie a algoritmului primal-dual Ford-Fulkerson, se micşorează excesul nodului s'. Şi potenţialul nodului t' se micşorează la fiecare iteraţie pentru că, după determinarea fluxului maxim de la s' la t' în $\tilde{G}'_0(f)$, în reţeaua reziduală $\tilde{G}'(f)$ nu există nici un drum de la s' la t' format doar din arce cu costurile reduse nule. Deci, la următoarea iteraţie, când se determină vectorul distanţelor minime, vom avea $d(t') \geq 1$. Deoarece, iniţial $e'(s') \leq n\overline{c}$ şi potenţialul unui nod nu poate deveni mai mic decât $-n\overline{b}$, rezultă că algoritmul efectuează cel mult $\min\{n\overline{c},\, n\overline{b}\}$ iteraţii. Complexitatea unei iteraţii este dată de determinarea drumului minim şi a fluxului maxim. Deci, dacă $D(n,m)$ este complexitatea al-

goritmului de determinare a drumului minim, iar $F(n, m, \overline{c})$ este complexitatea algoritmului de determinare a fluxului maxim, atunci complexitatea algoritmului primal-dual Ford-Fulkerson este $O\left(\min\{n\overline{c}, n\overline{b}\}\,(D(n,m) + F(n,m,\overline{c}))\right)$. ■

8.6.4 Algoritmul nonconform Minty - Fulkerson

Numele algoritmului se datorează faptului că, pe parcursul iteraţiilor intermediare, fiecare arc fie îndeplineşte condiţiile de optimalitate (caz în care spunem că este un *arc conform*), fie nu le îndeplineşte (caz în care spunem că este un *arc nonconform*).

Algoritmul nonconform Minty-Fulkerson lucrează în reţeaua reziduală. Definim *numărul conform* $k(x, y)$ al unui arc (x, y) din reţeaua reziduală $\tilde{G}(f)$ astfel:

$$k(x,y) = \begin{cases} 0, & \text{dacă} \quad b^p(x,y) \geq 0 \\ r(x,y), & \text{dacă} \quad b^p(x,y) < 0 \end{cases}$$

Numărul conform al unui arc (x, y) din reţeaua reziduală reprezintă cantitatea cu care trebuie modificată capacitatea reziduală a arcului (sau, echivalent, fluxul) astfel încât arcul să devină conform fără a-i modifica costul redus. Suma $K = \sum_{(x,y)} k(x, y)$ a numerelor conforme ale arcelor arată cât de "departe" de optim este fluxul curent.

Algoritmul nonconform Minty-Fulkerson porneşte de la un flux admisibil f şi de la potenţiale nule. La fiecare iteraţie, algoritmul micşorează numărul conform al unui arc sau al mai multor arce, menţinând arcele conforme şi transformând treptat arcele nonconforme în arce conforme. Deci, K descreşte la fiecare iteraţie astfel încât la sfârşitul algoritmului $K = 0$ (toate arcele sunt conforme) şi fluxul astfel obţinut este un flux de cost minim.

Algoritmul nonconform Minty-Fulkerson este următorul:

(1) PROGRAM NONCONFORM - MF;
(2) BEGIN
(3) $p := 0$;
(4) se determină un flux admisibil f;
(5) se determină reţeaua reziduală $\tilde{G}(f)$ şi numerele conforme ale arcelor;
(6) WHILE în reţeaua reziduală $\tilde{G}(f)$ există un arc nonconform DO
(7) BEGIN
(8) se selectează un arc nonconform (o, q) din $\tilde{G}(f)$;
(9) se consideră $\max\{0, b^p(x, y)\}$ ca fiind lungimea arcului (x, y), pentru toate arcele (x, y) din $\tilde{G}(f)$;
(10) fie d vectorul distanţelor minime de la nodul q la celelalte noduri în reţeaua $\tilde{G}(f)$ din care am eliminat arcul (q, o);

(11) fie D drumul minim de la nodul q la nodul o;
(12) $p' := p - d$;
(13) IF $b^{p'}(o,q) < 0$
(14) THEN BEGIN
(15) $\overset{\circ}{D} := D \cup \{(o,q)\}$;
(16) $r(\overset{\circ}{D}) := \min\{r(x,y) \mid (x,y) \in \overset{\circ}{D}\}$;
(17) se măreşte cu $r(\overset{\circ}{D})$ unităţi fluxul de-a lungul circuitului $\overset{\circ}{D}$;
(18) se actualizează f şi $\tilde{G}(f)$;
(19) END;
(20) $p := p'$;
(21) se actualizează costurile reduse;
(22) END;
(23) END.

Teorema 8.15. *Actualizarea potenţialelor nu măreşte numărul conform al nici unui arc din reţeaua reziduală.*

Demonstratie. Fie p şi p' potenţialele nodurilor înainte şi după actualizare. Numărul conform al arcului (x,y) ar creşte în urma actualizării potenţialelor doar în cazul în care $b^p(x,y) \geq 0$ şi $b^{p'}(x,y) < 0$. Vom arăta că acest caz nu poate să existe. Fie (x,y) un arc oarecare cu $b^p(x,y) \geq 0$. Vom arăta că $b^{p\prime}(x,y) \geq 0$. Cum (o,q) - arcul selectat de algoritm este un arc nonconform, $b^p(o,q) < 0$, deci $(x,y) \neq (o,q)$. Deoarece d este vectorul distanţelor minime considerând $\max\{0, b^p(x,y)\}$ ca fiind lungimea arcului (x,y) rezultă că

$$d(y) \leq d(x) + \max\{0, b^p(x,y)\} = d(x) + b^p(x,y)$$

sau, echivalent,

$$b^p(x,y) + d(x) - d(y) = b^{p'}(x,y) \geq 0.$$

Adică numărul conform al arcului (x,y) rămâne nemodificat, deci actualizarea potenţialelor nu măreşte numerele conforme ale arcelor. ■

Teorema 8.16. *Mărirea fluxului de-a lungul circuitului $\overset{\circ}{D}$ nu duce la creşterea numărului conform al nici unui arc şi micşorează numărul conform al arcului (o,q).*

Demonstraţie. Mărirea fluxului de-a lungul circuitului $\overset{\circ}{D}$ poate determina doar modificarea numerelor conforme ale arcelor din $\overset{\circ}{D} = D \cup \{(o,q)\}$ şi ale arcelor inverse lor. Deoarece D este un drum minim, rezultă că pentru orice arc $(x,y) \in D$ este îndeplinită următoarea condiţie:

$$d(y) = d(x) + \max\{0, b^p(x,y)\} \geq d(x) + b^p(x,y)$$

sau, echivalent,

$$b^p(x,y) + d(x) - d(y) = b^{p\prime}(x,y) \leq 0.$$

Deoarece costul redus al arcului (x, y) în raport cu potenţialele p' este negativ sau nul, mărirea fluxului pe acest arc nu duce la creşterea numărului său conform, ci, dimpotrivă poate duce la micşorarea lui deoarece se micşorează capacitatea reziduală a arcului (x, y).

Mărirea fluxului pe arcul (x, y) poate introduce (dacă nu există deja) arcul (y, x) în reţeaua reziduală. Dar, $b^{p'}(y, x) = -b^{p'}(x, y) \geq 0$, deci arcul (y, x) este un arc conform.

Considerăm acum arcul (o, q). Algoritmul efectuează mărirea de flux de-a lungul circuitului $\overset{\circ}{D}$ numai dacă şi după actualizarea potenţialelor (o, q) a rămas nonconform, adică $b^{p'}(o, q) < 0$. Dar, mărirea fluxului pe arcul (o, q) duce la scăderea capacităţii sale reziduale, deci şi a numărului său conform.

Mărirea fluxului pe arcul (o, q) poate duce la introducerea arcului (q, o) în reţeaua reziduală (dacă nu există deja). Acest arc este conform deoarece $b^{p'}(q, o) = -b^{p'}(o, q) > 0$. ■

Teorema 8.17. *Algoritmul nonconform Minty-Fulkerson determină un flux de cost minim.*
Demonstraţie. Algoritmul determină un flux de cost minim deoarece pe parcursul execuţiei sale numerele conforme ale arcelor descresc (conform Teoremelor 8.15 şi 8.16) şi algoritmul se termină când $K = 0$. ■

Teorema 8.18. *Algoritmul nonconform Minty-Fulkerson are complexitatea* $O\left(m\bar{c}D(n, m)\right)$, *unde* $D(n, m)$ *este complexitatea algoritmului pentru determinarea drumului minim într-o reţea cu lungimile arcelor pozitive.*
Demonstraţie. Iniţial, numărul conform al fiecărui arc este cel mult $\bar{c}$, deci suma numerelor conforme ale arcelor, K, este cel mult $m\bar{c}$. La fiecare iteraţie algoritmul selectează un arc nonconform şi fie îl transformă într-un arc conform prin actualizarea potenţialelor, fie îi reduce numărul conform prin mărirea fluxului de-a lungul unui circuit din care arcul face parte. Conform Teoremelor 8.15 şi 8.16, suma K a numerelor conforme scade cu cel puţin o unitate la fiecare iteraţie. Deci, după $O(m\bar{c})$ iteraţii, K devine nul şi algoritmul se termină. Pe parcursul unei iteraţii, operaţia cu cea mai mare complexitate este determinarea drumului minim. Deci, dacă $D(n, m)$ este complexitatea algoritmului pentru determinarea drumului minim, atunci complexitatea algoritmului nonconform Minty-Fulkerson este $O\left(m\bar{c}D(n, m)\right)$. ■

Exemplul 8.4. Ilustrăm algoritmul nonconform Minty-Fulkerson pe reţeaua din figura 8.5 (a). În figura 8.5 (b) este reprezentată reţeaua reziduală core-

Fig.8.5.

spunzătoare fluxului admisibil prezentat în figura 8.5 (a). Se selectează arcul nonconform (2,1) şi se determină vectorul distanţelor minime de la nodul 1 la celelalte noduri: $d = (d(1), d(2), d(3), d(4)) = (0, 6, 4, 6)$. Reţeaua obţinută după actualizarea potenţialelor este reprezentată în figura 8.5 (c). Se observă că, în urma actualizării potenţialelor, arcul (2,1) a devenit arc conform. Singurul arc nonconform este (4,2), deci se va determina vectorul distanţelor minime de la nodul 2 la celelalte noduri: $d = (3, 0, 3, 3)$. În figura 8.5 (d) este reprezentată reţeaua reziduală obţinută în urma actualizării potenţialelor. Deoarece arcul (4,2) a rămas arc nonconform, se va mări fluxul de-a lungul circuitului $\overset{\circ}{D}= (2, 3, 4, 2)$ cu $r(\overset{\circ}{D}) = \min\{2, 2, 2\} = 2$. Reţeaua reziduală obţinută după mărirea fluxului este ilustrată în figura 8.5 (e). Deoarece nu mai există arce nonconforme, algoritmul se termină. În figura 8.5 (f) este reprezentat fluxul de cost minim obţinut pornind de la capacităţile reziduale optime. Costul fluxului este $2\cdot2+4\cdot2+1\cdot2+4\cdot0+2\cdot4 = 22$.

8.6.5 Algoritmul Bertsekas - Tseng al relaxării

Algoritmul Bertsekas-Tseng al relaxării se bazează pe tehnica relaxării lagrangeanului care se foloseşte la rezolvarea problemelor de programare întreagă şi constă în a determina constrângerile care vor fi relaxate, a le înmulţi cu câte un scalar şi a scădea rezultatul astfel obţinut din funcţia obiectiv. Algoritmul Bertsekas-Tseng al relaxării relaxează constrângerile de conservare a fluxului (8.1.b), le înmulţeşte cu potenţialele nodurilor şi scade rezultatul din funcţia obiectiv. Astfel se obţine următoarea problemă relaxată (denumită problema $LR(p)$):

$$w(p) = \text{minimizează}_f \left[\sum_{(x,y)\in A} b(x,y)\cdot f(x,y) + \sum_{x\in N} p(x) \right.$$

$$\left. \left(- \sum_{y\,|\,(x,y)\in A} f(x,y) + \sum_{y\,|\,(y,x)\in A} f(y,x) + v(x) \right) \right] \tag{8.13.a}$$

$$0 \le f(x,y) \le c(x,y),\ (x,y) \in A \tag{8.13.b}$$

Soluţia optimă a problemei $LR(p)$ este un pseudoflux pentru că e posibil să nu îndeplinească condiţiile (8.1.b) de conservare a fluxului. Putem rescrie funcţia obiectiv a problemei $LR(p)$ în următoarea formă:

$$w(p) = \text{minimizează}_f \left(\sum_{(x,y)\in A} b(x,y)\cdot f(x,y) + \sum_{x\in N} p(x)e(x) \right). \tag{8.14}$$

Observăm că în al doilea termen al funcţiei obiectiv (8.13.a) $f(x,y)$ apare de două ori: o dată cu coeficientul $-p(x)$ şi altă dată cu coeficientul $p(y)$ şi putem rescrie (8.13.a) în modul următor:

$$w(p) = \text{minimizează}_f \left(\sum_{(x,y)\in A} (b(x,y) - p(x) + p(y))\, f(x,y) + \sum_{x\in N} p(x)\cdot v(x) \right)$$

sau, echivalent,

$$w(p) = \text{minimizează}_f \left(\sum_{(x,y)\in A} b(x,y)\cdot f(x,y) + \sum_{x\in N} p(x)\cdot v(x) \right). \qquad (8.15)$$

Proprietatea 8.19. *Dacă un pseudoflux f al problemei fluxului de cost minim satisface condiţiile de optimalitate cu costuri reduse în raport cu potenţialele p atunci f este o soluţie optimă pentru problema $LR(p)$.*
Demonstratie. Considerăm funcţia obiectiv în forma (8.15) şi (1) dacă $b^p(x,y) > 0$ considerăm $f(x,y) = 0$; (2) dacă $b^p(x,y) < 0$ considerăm $f(x,y) = c(x,y)$; (3) dacă $b^p(x,y) = 0$ atunci $f(x,y)$ poate avea orice valoare cuprinsă între 0 şi $c(x,y)$. Soluţia astfel obţinută este un pseudoflux pentru problema fluxului de cost minim şi satisface condiţiile de optimalitate cu costuri reduse. ■

Fie z^* valoarea optimă a funcţiei obiectiv a problemei fluxului de cost minim.
Teorema 8.20. (a) *Pentru orice potenţiale $p, w(p) \le z^*$.*
(b) *Există potenţialele p^* astfel încât $w(p^*) = z^*$.*
Demonstraţie. (a) Fie f^* un flux de cost minim (costul lui f^* este z^*). Evident, pentru orice potenţiale p, f^* este o soluţie admisibilă a problemei $LR(p)$ şi valoarea funcţie obiectiv a problemei $LR(p)$ este tot z^*. Deci, valoarea minimă a funcţiei obiectiv a problemei $LR(p)$ este mai mică sau egală cu z^*.
(b) Fie p^* un vector de potenţiale în raport cu care f^* îndeplineşte condiţiile de optim cu ecarturi complementare. Din Proprietatea 8.19 rezultă că f^* este o soluţie optimă pentru $LR(p^*)$ şi $w(p^*) = bf^* = z^*$. ■

Algoritmul Bertsekas-Tseng al relaxării menţine pe parcursul execuţiei sale un vector de potenţiale p şi un pseudoflux f care este o soluţie optimă pentru $LR(p)$, adică f îndeplineşte condiţiile de optimalitate cu costuri reduse în raport cu p. Algoritmul poate efectua una din următoarele două operaţii:

1. Păstrând p constant, înlocuieşte f cu f' astfel încât şi f' este o soluţie optimă a problemei $LR(p)$ şi excesul unui nod scade.

2. Înlocuieşte p cu p' şi f cu f' astfel încât f' este o soluţie optimă a problemei $LR(p')$ şi $w(p') > w(p)$.

Dacă la un moment dat algoritmul ar putea efectua ambele operaţii atunci va efectua operaţia 2. Deci, obiectivul principal al algoritmului relaxării este să mărească $w(p)$, iar obiectivul secundar este să transforme treptat pseudofluxul f în flux menţinând $w(p)$ constant.

În continuare discutăm algoritmul Bertsekas-Tseng al relaxării mai detaliat. Algoritmul efectuează mai multe iteraţii principale şi, într-o iteraţie principală, mai multe iteraţii secundare. La o iteraţie principală, se selectează un nod s cu exces şi se creează un arbore cu rădăcina s şi format doar din noduri cu exces nenegativ şi arce cu cost redus nul. La fiecare iteraţie secundară se adaugă câte un nod în arbore. O iteraţie principală se termină fie cu o mărire de flux, fie cu creşterea lui $w(p)$.

Fie S mulţimea nodurilor care aparţin arborelui la un moment dat şi $\overline{S} = N \setminus S$. Algoritmul utilizează două variabile $e(S)$ şi $r(p, S)$ definite în modul următor:

$$e(S) = \sum_{x \in S} e(x).$$

$$r(p, S) = \sum_{(x,y) \in (S,\overline{S}), b^p(x,y)=0} r(x, y).$$

Fiind dată mulţimea S, algoritmul verifică mai întâi care este relaţia dintre $e(S)$ şi $r(p, S)$. Dacă $e(S) > r(p, S)$ atunci se va mări $w(p)$ în modul următor: mai întâi se măreşte fluxul pe arcele din $(S, \overline{S})$ care au costul redus nul astfel încât să devină saturate. Mărirea fluxului pe aceste arce nu duce la modificarea lui $w(p)$ deoarece costul redus al arcelor este nul, dar duce la micşorarea cu $r(p, S)$ a sumei exceselor nodurilor. Cum $e(S) > r(p, S)$, suma actualizată a exceselor $(e(S) - r(p, S))$ este tot pozitivă. În acest moment, toate arcele din $(S, \overline{S})$ au costurile reduse strict pozitive. Algoritmul determină costul redus minim al arcelor din $(S, \overline{S})$, notat cu b_{pm} şi măreşte potenţialele nodurilor din S cu $b_{pm} > 0$ unităţi. Din expresia (8.14) a funcţiei obiectiv a problemei $LR(p)$ rezultă că modificarea potenţialelor nu afectează primul termen, dar duce la creşterea celui de-al doilea cu $(e(S) - r(p, S))\, b_{pm}$, care este o valoare strict pozitivă. Mărirea potenţialelor nodurilor din S duce la micşorarea costurilor reduse ale arcelor din $(S, \overline{S})$ cu b_{pm} unităţi, şi la mărirea costurilor reduse ale arcelor din $(\overline{S}, S)$ cu b_{pm} unităţi şi nu afectează costurile reduse ale celorlalte arce. Mărirea costurilor reduse nu afectează îndeplinirea condiţiilor de optimalitate cu costuri reduse. Deoarece înainte de actualizarea potenţialelor $b^p(x, y) \geq b_{pm},\ (x, y) \in (S, \overline{S})$, după actualizare $b^{p\prime}(x, y) \geq 0,\ (x, y) \in (S, \overline{S})$; deci nici micşorarea costurilor reduse nu afectează îndeplinirea condiţiilor de optimalitate. Deci, condiţiile de optimalitate rămân îndeplinite.

Dacă $e(S) \leq r(p,S)$ atunci cel puţin un arc $(x,y) \in (S,\overline{S})$ are costul redus nul deoarece $r(p,S) \geq e(S) > 0$. Dacă $e(y) \geq 0$ atunci nodul y este adăugat în S, se încheie o iteraţie secundară şi se repetă procesul. Dacă $e(y) < 0$ atunci se măreşte fluxul pe drumul de la s la y. Deoarece mărirea se face pe arce cu costul redus nul, rezultă că nu se modifică valoarea funcţiei obiectiv a problemei $LR(p)$. Mărirea fluxului duce la micşorarea lui $e(S)$ şi încheie o iteraţie principală.

Algoritmul Bertsekas-Tseng al relaxării este următorul:

```
(1) PROGRAM BT-RELAXARE;
(2) BEGIN
(3)     f := 0;
(4)     p := 0;
(5)     WHILE în reţea există un nod s cu e(s) > 0 DO
(6)     BEGIN
(7)         S := {s};
(8)         IF e(S) > r(p,S)
(9)             THEN MODIFICĂ - POTENŢIAL;
(10)        REPEAT
(11)            se selectează un arc (x,y) ∈ (S,S̄) din reţeaua
                reziduală cu b^p(x,y) = 0;
(12)            IF e(y) ≥ 0
(13)                THEN BEGIN
(14)                    pred (y) := x;
(15)                    S := S ∪ {y};
(16)                END
(17)        UNTIL e(y) < 0 sau e(S) > r(p,S);
(18)        IF e(S) > r(p,S)
(19)            THEN MODIFICĂ - POTENŢIAL;
(20)            ELSE MODIFICĂ - FLUX (s,y)
(21)    END;
(22) END.
```

```
(1) PROCEDURA MODIFICĂ-POTENŢIAL;
(2) BEGIN
(3)     FOR toate arcele (x,y) ∈ (S,S̄) cu b^p(x,y) = 0 DO
(4)         se măreşte cu r(x,y) unităţi fluxul pe arcul (x,y);
(5)     b_pm := min{b^p(x,y) | (x,y) ∈ (S,S̄) şi r(x,y) > 0};
(6)     FOR toate nodurile x ∈ S DO p(x) := p(x) + b_pm;
(7)     se actualizează costurile reduse;
(8) END;
```

(1) PROCEDURA MODIFICĂ-FLUX (s, y);
(2) BEGIN
(3) se determină un drum D de la s la y cu ajutorul vectorului *pred*;
(4) $r(D) := \min\left(e(s), -e(y), \min\{r(x, y) \mid (x, y) \in D\}\right)$;
(5) se măreşte cu $r(D)$ unităţi fluxul de-a lungul drumului D;
(6) se actualizează excesele nodurilor şi capacităţile reziduale;
(7) END;

Exemplul 8.5 Ilustrăm algoritmul Bertsekas-Tseng al relaxării pe reţeaua din figura 8.6(a). Iniţial $S = \{1\}$, $e(S) = 1$ şi $r(p, S) = 0$. Cum $e(S) > r(p, S)$ se aplică PROCEDURA MODIFICĂ-POTENŢIAL. Reţeaua reziduală astfel obţinută este reprezentată în figura 8.6 (b). Se selectează arcul (1,2), *pred*(2)=1, $S = \{1, 2\}$, $e(S) = 1 > 0 = r(p, S)$ şi se aplică din nou PROCEDURA MODIFICĂ-POTENŢIAL, iar reţeaua obţinută este ilustrată în figura 8.6 (c). Din nou $S = \{1\}$, $e(S) = 1$, iar $r(p, S) = 3$. Se selectează arcul (1,2), *pred*$(2) = 1, S = \{1, 2\}, r(p, S) = 1 = e(S)$. Se selectează arcul (2,3), *pred*$(3) = 2, S = \{1, 2, 3\}, r(p, S) = 0 < e(S) = 1$. Rezultă că se aplică PROCEDURA MODIFICĂ-POTENŢIAL cu $b_{pm} = 2$. Reţeaua reziduală obtinută este ilustrată în figura 8.6 (d). Apoi, $S = \{1\}, e(S) = 1$, iar $r(p, S) = 2$. Se selectează arcul (1,2), *pred*$(2) = 1, S = \{1, 2\}$, $r(p, S) = 1$. Rezultă că se selectează arcul (2,3), *pred*$(3) = 2, S = \{1, 2, 3\}, r(p, S) = 3$. Se selectează arcul (3,4), *pred*$(4) = 3, S = \{1, 2, 3, 4\}, r(p, S) = 0$. Cum $e(4) < 0$ şi $r(p, S) = 0 = e(S)$ se aplică PROCEDURA MODIFICĂ-FLUX. Drumul D este (1, 2, 3, 4), $r(D) = \min\{r(1, 2), r(2, 3), r(3, 4)\} = 1$. Se măreşte cu 1 fluxul pe drumul D, iar reţeaua reziduală obţinută după mărire este reprezentată în figura 8.6 (e). Fiindcă nu mai există noduri cu exces, algoritmul se termină cu un flux de cost minim care este reprezentat în figura 8.6 (f) şi al cărui cost este 5.

Teorema 8.21. *Algoritmul Bertsekas-Tseng al relaxării determină corect un flux de cost minim.*
Demonstraţie. Algoritmul se termină atunci când în reţea nu mai există noduri cu exces, deci când pseudofluxul a fost transformat în flux. Mai mult, acest flux este un flux de cost minim deoarece pseudofluxul din care a fost obţinut îndeplinea la fiecare iteraţie condiţiile de optimalitate cu costuri reduse. ■

Teorema 8.22. *Complexitatea algoritmului Bertsekas-Tseng al relaxării este* $O(nm^3 \overline{b}\,\overline{c}^{\,2})$.
Demonstraţie. Suma exceselor nodurilor la sfârşitul fiecărei execuţii a PROCEDURII MODIFICĂ - POTENŢIAL este cel mult $O((m + n)\,\overline{c}) = O(m\overline{c})$. Deci, între două execuţii ale PROCEDURII MODIFICĂ - POTENŢIAL se execută de

cel mult $O(m\overline{c})$ ori PROCEDURA MODIFICĂ - FLUX, deoarece după fiecare execuţie a PROCEDURII MODIFICĂ - FLUX se micşorează excesul unui nod cu cel puţin o unitate. Dar, numărul de execuţii ale PROCEDURII MODIFICĂ - POTENŢIAL este cel mult $O(m\,\overline{b}\,\overline{c})$, deoarece după fiecare execuţie a procedurii valoarea $w(p)$ creşte cu cel puţin o unitate, iar $w(p)$ iniţial era nulă şi nu poate depăşi $m\,\overline{b}\,\overline{c}$. Deci, numărul de iteraţii principale efectuate de algoritm este $O(m^2\overline{b}\overline{c}^{\,2})$. Deoarece pe parcursul unei iteraţii principale se efectuează cel mult n iteraţii secundare, rezultă că numărul iteraţiilor secundare efectuate de algoritm este $O(nm^2\overline{b}\,\overline{c}^{\,2})$. Cum complexitatea unei iteraţii secundare este $O(m)$, rezultă că complexitatea algoritmului Bertsekas-Tseng al relaxării este $O(nm^3\,\overline{b}\,\overline{c}^{\,2})$. ■

Fig.8.6.

8.7 Aplicaţii şi comentarii bibliografice

8.7.1 Probleme de distribuţie

Un producător de maşini are mai multe fabrici care produc diferite modele de maşini, pe care apoi le transportă la mai mulţi distribuitori. Fiecare distribuitor solicită câte un număr specificat de maşini din fiecare model. Producătorul trebuie să determine câte un plan de producţie pentru fiecare fabrică şi un plan de transport astfel încât să fie satisfăcute cererile distribuitorilor şi să se minimizeze costul total de producţie şi transport.

Exemplul 8.6. În figura 8.7 ilustrăm o problemă de distribuţie în care sunt două fabrici, doi distribuitori şi trei modele de maşini.

Fig.8.7

Acest model are patru tipuri de noduri:
(1) **noduri fabrică**, reprezentând diferitele fabrici
(2) **noduri fabrică/model**, corespunzătoare fiecărui model construit la fiecare fabrică
(3) **noduri distribuitor/model**, corespunzătoare fiecărui model solicitat de fiecare distribuitor
(4) **noduri distribuitor**, reprezentând fiecare distribuitor.

Reţeaua conţine trei tipuri de arce:
(1) **arce de producţie.** Un arc de producţie are ca extremitate iniţială un nod fabrică, iar ca extremitate finală un nod fabrică/model. Costul unui astfel de arc este costul de producţie a unei maşini din modelul respectiv la fabrica specificată.

Se pot introduce margini inferioare şi capacităţile acestor arce pentru a impune un număr minim şi un număr maxim de maşini din fiecare model produse la fiecare fabrică.
(2) **arce de transport.** Un arc de trasnport are ca extremitate iniţială un nod fabrică/model, iar ca extremitate finală un nod distribuitor/model. Costul unui asemenea arc reprezintă costul transportului unei maşini de la fabrica la care a fost produsă la distribuitorul care o solicită. Un arc de transport poate avea margine inferioară şi/sau capacitate.
(3) **arce de distribuţie.** Un arc de distribuţie are ca extremitate iniţială un nod distribuitor/model, iar ca extremitate finală un nod distribuitor. Costul unui astfel de arc este nul, iar marginea sa inferioară este egală cu numărul de maşini din modelul respectiv solicitate de distribuitorul respectiv.

Evident, un flux admisibil în reţeaua astfel construită corespunde unui plan de producţie şi transport al producătorului de maşini şi reciproc. Deci, un flux de cost minim va furniza un plan de producţie şi transport optim.

8.7.2 Problema şcolilor mixte

Considerăm un oraş în care sunt NS şcoli. Presupunem că oraşul este împărţit în NZ zone, iar m_i şi $\overline{m}_i$ reprezintă numărul de studenţi care aparţin populaţiei minoritare, respectiv majoritare din zona i. Considerăm d_{ij} ca fiind distanţa pe care un elev din zona i trebuie să o parcurgă ca să ajungă la şcoala j. În şcoala j pot învăţa cel mult c_j elevi. Fie $\underline{p}$ o margine inferioară şi $\overline{p}$ o margine superioară pentru procentul de elevi din populaţia minoritară care pot învăţa la fiecare şcoală. Problema şcolilor mixte constă în a înscrie elevii la şcoli astfel încât procentul de elevi minoritari din fiecare şcoală să fie cuprins între $\underline{p}$ şi $\overline{p}$ şi să se minimizeze distanţa totală parcursă de elevi.

Formulăm problema şcolilor mixte ca o problemă de flux de cost minim. În figura 8.8 este reprezentată o reţea care corespunde unui oraş cu două şcoli şi trei zone. Fiecărei zone i îi corespund două noduri z_i' şi z_i'', iar fiecărei şcoli j îi corespund două noduri s_j' şi s_j''. Fluxul de pe arcul $\left(z_i', s_j'\right)$ va reprezenta numărul de elevi minoritari care locuiesc în zona i şi învaţă la şcoala j, iar fluxul de pe arcul $\left(z_i'', s_j''\right)$ va reprezenta numărul de studenţi majoritari care locuiesc în zona i şi învaţă la şcoala j. Considerăm $\ell\left(z_i', s_j'\right) = 0,\ c\left(z_i', s_j'\right) = \infty,\ b\left(z_i', s_j'\right) = d_{ij},\ \ell\left(z_i'', s_j''\right) = 0,\ c\left(z_i'', s_j''\right) = \infty, ; b\left(z_i'', s_j''\right) = d_{ij}$, pentru $i = 1, \cdots,$ NZ , $j = 1, \cdots,$ NS. Adăugăm arce $\left(s_j', s_j\right)$ şi $\left(s_j'', s_j\right)$, pentru $j = 1, \cdots,$ NS . Fluxul de pe arcul $\left(s_j', s_j\right)$ va reprezenta numărul de studenţi minoritari care învaţă la şcoala j, iar fluxul de pe arcul $\left(s_j'', s_j\right)$ va reprezenta numărul de studenţi

majoritari care învaţă la şcoala j. Considerăm $\ell\left(s'_j, s_j\right) = \underline{p}c_j$, $c\left(s'_j, s_j\right) = \overline{p}c_j$, $b\left(s'_j, s_j\right) = 0$, $\ell\left(s''_j, s_j\right) = 0$, $c\left(s''_j, s_j\right) = \infty$, $b\left(s''_j, s_j\right) = 0$, pentru $j = 1, \cdots, \text{NS}$. Adăugăm un nod stoc t şi arce (s_j, t), cu $\ell(s_j, t) = 0$, $c(s_j, t) = c_j$ şi $b(s_j, t) = 0$, pentru $j = 1, \cdots,$ NS. Evident, problema fluxului de cost minim în reţeaua astfel construită modelează corect problema şcolilor mixte. Deci, un flux de cost minim va corespunde unei soluţii optime a problemei şcolilor mixte.

Fig.8.8

8.7.3 Comentarii bibliografice

În acest capitol am prezentat cinci algoritmi pseudopolinomiali pentru rezolvarea problemei fluxului de cost minim. În capitolul 9 sunt prezentaţi algoritmi polinomiali pentru determinarea fluxului de cost minim. Ford şi Fulkerson (1957) au rezolvat o problemă de transport într-o reţea cu capacităţi finite folosind algoritmul primal-dual. Mai târziu, Ford şi Fulkerson (1962) au generalizat acest algoritm pentru a-l folosi la rezolvarea problemei fluxului de cost minim. Independent, Jewell (1958), Iri (1960) şi Busaker şi Gowen (1961) au descris algoritmul drumurilor minime succesive, bazându-se pe faptul că se poate determina un flux de cost minim rezolvând mai multe probleme de drum minim în reţele cu lungimi arbitrare ale arcelor. Tomizava (1972) şi Edmons şi Karp (1972) au observat independent că, folosind potenţialele nodurilor, algoritmul drumurilor minime succesive poate fi implementat astfel încât să se determine drumurile minime în reţele cu lungimi pozitive ale arcelor.

Independent, Minty (1960) şi Fulkerson (1961) au prezentat algoritmul nonconform. Mai târziu, Aashtiani şi Magnanti (1976) au descris o implementare mai eficientă a acestui algoritm.

Klein (1967) a prezentat algoritmul de eliminare a circuitelor de cost negativ din reţeaua reziduală. Cinci implementări ale acestui algoritm au complexitate polinomială: prima, datorată lui Goldberg şi Tarjan (1988), măreşte fluxul de-a lungul circuitului negativ cu costul mediu minim; a doua, datorată tot lui Goldberg şi Tarjan (1988) măreşte fluxul de-a lungul circuitului format numai din arce cu costurile reduse nule; a treia datorată lui Barahona şi Tardos (1989), măreşte fluxul de-a lungul circuitului de cost negativ care furnizează cea mai mare micşorare a costului fluxului; a patra implementare a fost descrisă de Wallacher şi Zimmerman (1991); iar a cincea, se datorează lui Sokkalingam, Ahuja şi Orlin, măreşte fluxul pe circuitul de cost cu capacitatea reziduală suficient de mare şi va fi prezentată în capitolul următor.

Bertsekas şi Tseng (1989) au descris algoritmul relaxării care are complexitatea $O\left(nm^3 \overline{b}\, \overline{c}^{\,2}\right)$.

Capitolul 9

Algoritmi polinomiali pentru fluxul de cost minim

9.1 Algoritmul Orlin al scalării capacităţii

Algoritmul Orlin al scalării capacităţii este o variantă a algoritmului Busaker-Gowen al drumurilor minime succesive care a fost obţinută folosind tehnica scalării. Algoritmul Orlin al scalării capacităţii impune condiţia ca după fiecare mărire valoarea fluxului să crească "suficient de mult". Această modificare reduce numărul de iteraţii de la $O(n\,\overline{c})$ (efectuate de algoritmul Busaker-Gowen al drumurilor minime succesive) la $O(m \log \overline{c})$.

Algoritmul Orlin al scalării capacităţii porneşte de la un pseudoflux nul şi potenţiale nule. Deci, la începutul execuţiei algoritmului, sunt îndeplinite condiţiile de optimalitate cu costuri reduse. Pe parcursul execuţiei sale, algoritmul foloseşte un pseudoflux f care îndeplineşte condiţiile de optimalitate cu costuri reduse şi pe care îl transformă treptat în flux prin măriri de flux de-a lungul drumurilor minime de la noduri cu exces la noduri cu deficit.

Algoritmul Orlin al scalării capacităţii efectuează mai multe faze de scalare pentru diferite valori ale parametrului $\overline{r}$. Numim fază de $\overline{r}$-scalare o fază de scalare pentru o valoare dată a lui $\overline{r}$. La începutul algoritmului $\overline{r} = 2^{\lfloor \log\, \overline{c} \rfloor}$. În fiecare fază de $\overline{r}$ -scalare, fiecare mărire de flux se face cu exact $\overline{r}$ unităţi. Când acest lucru nu mai este posibil din cauză că nu există noduri cu exces mai mare sau egal cu $\overline{r}$ sau noduri cu deficit mai mic sau egal cu $-\overline{r}$, valoarea lui $\overline{r}$ este înjumătăţită şi apoi se reia procesul. La sfârşitul algoritmului $\overline{r} = 1$, iar la sfârşitul fazei de 1- scalare pseudofluxul devine flux. Acest flux este un flux de cost minim deoarece îndeplineşte condiţiile de optimalitate cu costuri reduse.

Definim reţeaua $\overline{r}-$ reziduală $\tilde{G}(f, \overline{r})$ ca fiind un subgraf al reţelei reziduale $\tilde{G}(f)$ format doar din arcele a căror capacitate reziduală este mai mare sau egală

cu $\overline{r}$. În fiecare fază de $\overline{r}-$ scalare toate arcele din reţeaua $\overline{r}-$ reziduală $\tilde{G}(f,\overline{r})$ îndeplinesc condiţiile de optimalitate cu costuri reduse (este posibil ca arcele care nu aparţin reţelei $\tilde{G}(f,\overline{r})$ să nu îndeplinească aceste condiţii de optimalitate). Algoritmul Orlin al scalării capacităţii este următorul:

```
(1) PROGRAM SCALARE CAPACITATE;
(2) BEGIN
(3)     f := 0;
(4)     p := 0;
(5)     r̄ := 2^⌊log c̄⌋;
(6)     se construieşte reţeaua G̃(f, r̄);
(7)     WHILE r̄ ≥ 1 DO
(8)     BEGIN
(9)        FOR toate arcele (x, y) din G̃(f) DO
(10)           IF r(x, y) ≥ 0 şi b^p(x, y) < 0
(11)              THEN BEGIN
(12)                        se măreşte cu r(x, y) fluxul pe arcul (x, y);
(13)                        se actualizează f şi e;
(14)                   END;
(15)        S(r̄) := {x ∈ N | e(x) ≥ r̄};
(16)        T(r̄) := {x ∈ N | e(x) ≤ −r̄};
(17)        WHILE S(r̄) ≠ ∅ şi T(r̄) ≠ ∅ DO
(18)        BEGIN
(19)           se selectează un nod s ∈ S(r̄) şi un nod t ∈ T(r̄);
(20)           se determină distanţele minime d(·) de la s la celelalte noduri
                  în G̃(f, r̄) în raport cu costurile reduse;
(21)           se determină drumul minim D de la s la t în G̃(f, r̄)
(22)           p := p − d;
(23)           se măreşte cu r̄ unităţi fluxul pe drumul D;
(24)           se actualizează f, S(r̄), T(r̄) şi G̃(f, r̄);
(25)        END;
(26)        r̄ := r̄/2;
(26)     END;
(27) END.
```

Exemplul 9.1. Ilustrăm algoritmul Orlin al scalării capacităţii pe reţeaua din figura 9.1(a). Reţeaua reziduală obţinută după iniţializările din liniile (3) şi (4) este reprezentată în figura 9.1(b). $\overline{r} = 2$, iar reţeaua 2 - reziduală $\tilde{G}(f,2)$ este ilustrată în figura 9.1(c). Deoarece $S(2) = \{1\}$ şi $T(2) = \{4\}$, vom determina distanţele minime de la nodul 1 la celelalte noduri: $d = (0,2,4,6)$. Drumul minim de la 1 la 4 este $D = (1,2,4)$. Reţeaua 2-reziduală obţinută după actualizarea

potenţialelor şi mărirea fluxului este reprezentată în figura 9.1(d). În continuare $S(2) = \{1\}$ şi $T(2) = \{4\}$, deci vom determina din nou distanţele minime de la 1 la celelalte noduri: $d = (0, 0, 0, 0)$. Drumul minim de la 1 la 4 este $D = (1, 3, 4)$, iar reţeaua 2-reziduală obţinută după actualizarea potenţialelor şi după mărirea fluxului pe drumul D este ilustrată în figura 9.1(e). Deoarece $S(2) = T(2) = \emptyset$ se trece la următoarea fază de scalare. Reţeaua 1-reziduală, după mărirea fluxului cu o unitate pe arcul (2,3), care avea costul redus negativ, este reprezentată în figura 9.1(f). Deoarece $S(1) = \{3\}$ şi $T(1) = \{2\}$, vom determina distanţele minime de la 3 la celelalte noduri $d = (1, 0, 0, 0)$. Drumul minim de la 3 la 2 este $D = (3, 4, 2)$. Reţeaua 1-reziduală obţinută după mărirea fluxului de-a lungul drumului D este reprezentată în figura 9.1(g). Cum $S(1) = T(1) = \emptyset$, algoritmul se termină. Fluxul de cost minim este ilustrat în figura 9.1(h), iar costul său este 23.

Fig.9.1.

Teorema 9.1. *Algoritmul Orlin al scalării capacităţii determină corect un flux de cost minim.*
Demonstraţie. Faza de $2\overline{r}$-scalare, se termină atunci când $S(2\overline{r}) = \emptyset$ sau $T(2\overline{r}) = \emptyset$, adică $e(x) < 2\overline{r},\ x \in N$ sau $e(x) > -2\overline{r}, x \in N$. Rezultă că, la sfârşitul fazei de $2\overline{r}$-scalare, suma exceselor este cel mult $2n\overline{r}$.

La începutul fazei de $\overline{r}$-scalare se introduc în reţeaua $\overline{r}$-reziduală noi arce, care pot să nu verifice condiţiile de optimalitate. Algoritmul saturează acele arce care nu îndeplinesc condiţiile de optimalitate, eliminându-le astfel din reţeaua $\overline{r}$-reziduală şi introducând în reţeaua $\overline{r}$-reziduală arcele inverse, care însă satisfac condiţiile de optimalitate. Deoarece $r(x, y) < 2\overline{r}$, pentru fiecare arc (x, y) introdus în reţeaua $\overline{r}$-reziduală la începutul fazei de $\overline{r}$- scalare, rezultă că saturarea arcului (x, y) duce la modificarea excesului sau deficitului nodurilor x şi y cu cel mult $2\overline{r}$. Deci, după saturarea tuturor arcelor care nu îndeplinesc condiţiile de optimalitate, suma exceselor este mai mică decât $2n\overline{r} + 2m\overline{r} = 2(n + m)\overline{r}$.

Pe parcursul fazei de $\overline{r}$-scalare, fiecare mărire de flux se efectuează de-a lungul unui drum minim de la un nod $s \in S(\overline{r})$ la un nod $t \in T(\overline{r})$. Conform Proprietăţii 8.2(a) şi după mărirea de flux arcele vor îndeplini condiţiile de optimalitate cu costuri reduse.

Faza de $\overline{r}$-scalare se termină când $S(\overline{r}) = \emptyset$ sau $T(\overline{r}) = \emptyset$. Se înjumătăţeşte $\overline{r}$ şi se începe o nouă fază de scalare. După $O(\log \overline{c})$ faze de scalare, $\overline{r} = 1$, iar la sfârşitul fazei de 1-scalare $e(x) = 0,\ x \in N$ deoarece datele problemei sunt întregi conform ipotezei 2). Pe parcursul fazei de 1-scalare, $\tilde{G}(f, \overline{r}) = \tilde{G}(f)$ şi fiecare arc din reţeaua reziduală îndeplineşte condiţiile de optimalitate cu costuri reduse. Rezultă că, la sfârşitul fazei de 1-scalare, algoritmul determină un flux de cost minim. ■.

Teorema 9.2. *Complexitatea algoritmului Orlin al scalării capacităţii este* $O(m \log \overline{c}\ D(n, m))$, *unde* $D(n, m)$ *este complexitatea algoritmului de determinare a drumului minim într-o reţea cu arce cu lungimi pozitive.*
Demonstraţie. Algoritmul efectuează $O(log\overline{c})$ faze de scalare. Cum la începutul fazei de $\overline{r}$-scalare suma exceselor este mărginită de $2(n + m)\overline{r}$, iar la sfârşit este nulă şi fiecare mărire de flux se face cu $\overline{r}$ unităţi, rezultă că în fiecare fază de scalare se efectuează cel mult $2(n + m)$ măriri de flux. Deoarece la fiecare mărire de flux se determină un drum minim, rezultă că algoritmul Orlin are complexitatea $O(m \log \overline{c} D(n, m))$. ■

9.2 Algoritmul Goldberg-Tarjan al scalării costului

Înainte de a descrie algoritmul Goldberg-Tarjan al scalării costului trebuie să prezentăm conceptul de optimalitate aproximativă. Spunem că un pseudoflux f este ϵ-optimal ($\epsilon > 0$) dacă există potenţialele nodurilor p astfel încât să fie îndeplinite următoarele condiţii de ϵ-optimalitate.

$$\text{dacă } b^p(x,y) > \epsilon \text{ atunci } f(x,y) = 0 \tag{9.1.a}$$

$$\text{dacă } -\epsilon \leq b^p(x,y) \leq \epsilon \text{ atunci } 0 \leq f(x,y) \leq c(x,y) \tag{9.1.b}$$

$$\text{dacă } b^p(x,y) < -\epsilon \text{ atunci } f(x,y) = c(x,y) \tag{9.1.c}$$

Aceste condiţii sunt relaxări ale condiţiilor de optimalitate cu ecarturi complementare (8.3) şi se reduc la condiţiile de optimalitate cu ecarturi complementare în cazul în care $\epsilon = 0$.

Condiţiile de ϵ-optimalitate se pot exprima şi în reţeaua reziduală. Spunem că un pseudoflux f este ϵ-optimal ($\epsilon > 0$) dacă există potenţialele nodurilor p astfel încât să fie îndeplinite următoarele condiţii de ϵ-optimalitate.

$$b^p(x,y) \geq -\epsilon \text{ pentru orice arc } (x,y) \text{ din reţeaua reziduală } \tilde{G}(f) \tag{9.2}$$

Evident, condiţiile (9.1) şi (9.2) sunt echivalente.

Teorema 9.3. *Într-o problemă de flux de cost minim cu costurile întregi, orice flux admisibil este ϵ-optimal pentru $\epsilon \geq \overline{b}$. Mai mult, dacă $\epsilon < 1/n$, atunci orice flux admisibil ϵ-optimal este un flux de cost minim.*

Demonstraţie. Fie f un flux admisibil şi fie $p = 0$. Atunci $b^p(x,y) = b(x,y) \geq -\overline{b}$ pentru orice arc (x,y) din reţeaua reziduală $\tilde{G}(f)$. Deci f este ϵ-optimal pentru $\epsilon \geq \overline{b}$.

Fie $\epsilon < (1/n)$ şi f un flux ϵ-optimal în raport cu potenţialele p. Fie $\overset{\circ}{D}$ un circuit oarecare din reţeaua reziduală $\tilde{G}(f)$. Rezultă că $\sum_{(x,y)\in \overset{\circ}{D}} b^p(x,y) \geq -\epsilon\, n > -1$. Deoarece costurile sunt întregi rezultă că $\sum_{(x,y)\in \overset{\circ}{D}} b^p(x,y) \geq 0$. Dar $\sum_{(x,y)\in \overset{\circ}{D}} b(x,y) = \sum_{(x,y)\in \overset{\circ}{D}} b^p(x,y)$ (conform Proprietăţii 8.2 (b)), deci circuitul $\overset{\circ}{D}$ are costul nenegativ. Cum circuitul $\overset{\circ}{D}$ a fost ales aleator, rezultă că reţeaua reziduală $\tilde{G}(f)$ nu conţine circute de cost negativ, deci f este un flux de cost minim (conform Teoremei 8.1). ■

Algoritmul Goldberg-Tarjan al scalării costului determină fluxuri ϵ-optimale pentru valori ale parametrului ϵ din ce în ce mai mici. Iniţial, $\epsilon = \overline{b}$ şi, conform Teoremei 9.3, orice flux admisibil este ϵ-optimal. Algoritmul efectuează faze de

scalare a costului aplicând procedura ÎMBUNĂTĂŢEŞTE-APROXIMAŢIA care transformă un flux ϵ -optimal într-un pseudoflux $\frac{\epsilon}{2}$ -optimal, iar pe acesta într-un flux $\frac{\epsilon}{2}$ -optimal. După $1 + \lceil \log(n\bar{b}) \rceil$ faze de scalare a costului se obţine $\epsilon < \frac{1}{n}$ şi algoritmul se termină cu un flux de cost minim. Spunem că orice arc (x, y) din reţeaua reziduală este admisibil dacă $-\frac{\epsilon}{2} \leq b^p(x, y) < 0$. Procedura ÎMBUNĂTĂŢEŞTE-APROXIMAŢIA transformă un pseudoflux $\frac{\epsilon}{2}$ -optimal într-un flux $\frac{\epsilon}{2}$ -optimal astfel: selectează un nod activ x şi măreşte fluxul pe arcele admisibile care ies din x'. Când în reţeaua reziduală nu mai există astfel de arce admisibile, se actualizează potenţialul nodului x, $p(x)$, pentru a crea arce admisibile care ies din x.

Algoritmul Goldberg-Tarjan al scalării costului este următorul:

```
(1) PROGRAM GOLDBERG-TARJAN;
(2) BEGIN
(3)     se determină un flux admisibil f în G;
(4)     p := 0;
(5)     ε := b̄;
(6)     WHILE ε ≥ 1/n DO
(7)     BEGIN
(8)        ÎMBUNĂTĂŢEŞTE - APROXIMAŢIA (ε, f, p)
(9)        ε := ε/2;
(10)    END;
(11) END.
```

```
(1) PROCEDURA ÎMBUNĂTĂŢEŞTE - APROXIMAŢIA (ε, f, p);
(2) BEGIN
(3)     FOR toate arcele (x, y) ∈ A DO
(4)           IF b^p(x, y) > 0
(5)              THEN f(x, y) = 0
(6)              ELSE IF b^p(x, y) < 0
(7)                        THEN f(x, y) := c(x, y);
(8)     se determină excesele / deficitele nodurilor;
(9)     WHILE G̃(f) conţine un nod activ DO
(10)    BEGIN
(11)          se selectează un nod activ x;
(12)          ÎNAINTARE / REETICHETARE (x);
(13)     END;
(14) END.
```

```
(1) PROCEDURA ÎNAINTARE / REETICHETARE (x);
```

(2) BEGIN
(3) IF există un arc admisibil (x, y) în $\tilde{G}(f)$;
(4) THEN se măreşte cu $g = \min\{e(x), r(x, y)\}$ fluxul de la nodul x la nodul y;
(5) ELSE $p(x) := p(x) + \epsilon/2$;
(6) END.

Exemplul 9.2. Ilustrăm prima fază de scalare a costului efectuată de algoritmul Goldberg-Tarjan pe reţeaua din figura 9.2 (a), în care este reprezentat şi un flux admisibil. Iniţial $\epsilon = \overline{b} = 4$, iar reţeaua reziduală obţinută după efectuarea liniilor (3)-(8) din PROCEDURA ÎMBUNĂTĂŢEŞTE - APROXIMAŢIA este reprezentată în figura 9.2 (b). Nodul 1 este activ şi deoarece nu există arce admisibile care ies din nodul 1, se actualizează potenţialul său $p(1) = 2$. În continuare nodul 1 este activ şi din el nu ies arce admisibile, deci i se modifică potenţialul $p(1) = 4$, creându-se astfel arcul admisibil (1,2) ($b^p(1, 2) = -2 \in [-2, 0)$). În figura 9.2 (c) este reprezentată reţeaua reziduală după mărirea cu $g = \min\{e(1), r(1, 2)\} = \min\{4, 3\} = 3$ a fluxului de la nodul 1 la nodul 2. În acest moment, există două noduri active 1 şi 2. Dacă se selectează din nou nodul 1, se va actualiza potenţialul său, $p(1) = 6$, creându-se astfel arcul admisibil (1,3). Reţeaua reziduală obţinută după mărirea cu o unitate a fluxului de la 1 la 3 este reprezentată în figura 9.2 (d). Presupunem că se selectează nodul activ 2. Deoarece nu există arce admisibile care ies din 2, se va modifica potenţialul nodului 2, $p(2) = 2$. Astfel arcul (2,3) a devenit admisibil şi se măreşte cu o unitate fluxul pe acest arc. Reţeaua reziduală obţinută este ilustrată în figura 9.2 (e). Presupunem că din nou se selectează nodul activ 2. După modificarea potenţialului său, $p(2) = 4$, apoi $p(2) = 6$, se creează arcele admisibile (2,1) şi (2,4). Considerăm că se alege arcul (2,1). Reţeaua reziduală obţinută după mărirea fluxului cu două unităţi pe arcul (2,1) este reprezentată în figura 9.2 (f). Nodurile active sunt 1 şi 3. Presupunem că se selectează nodul 1. Se va mări fluxul cu două unităţi pe arcul (1,3). În figura 9.2 (g) este reprezentată reţeaua reziduală astfel obţinută. Singurul nod activ este 3, i se modifică potenţialul $p(3) = 2$, apoi $p(3) = 4$ pentru a crea arce admisibile. Reţeaua reziduală obţinută după mărirea cu trei unităţi a fluxului pe arcul (3,4) este reprezentată în figura 9.2 (h). Singurul nod activ din reţea este nodul 3. Se măreşte cu o unitate fluxul de la nodul 3 la nodul 1, iar reţeaua reziduală astfel obţinută este reprezentată în figura 9.2 (i). Singurul nod activ din reţea este nodul 1. Se modifică potenţialul $p(1) = 8$, apoi $p(1) = 10$ şi se măreşte fluxul cu o unitate pe arcul (1,2). Reţeaua astfel obţinută este reprezentată în figura 9.2 (j). Nodul 2 este singurul nod activ, se măreşte fluxul cu o unitate pe arcul admisibil (2,4) şi se obţine reţeaua din figura 9.2 (k). Fluxul corespunzător este 2 -optim, deci s-a încheiat faza de scalare a costului pentru $\epsilon = 4$.

Fig.9.2.

Teorema 9.4. *Algoritmul Goldberg-Tarjan al scalării costului determină corect un flux de cost minim.*

Demonstraţie. Algoritmul efectuează faze de scalare a costului pentru valori ale lui ϵ din ce în ce mai mici. Pe parcursul fiecărei faze de scalare se apelează PROCEDURA ÎMBUNĂTĂŢEŞTE - APROXIMAŢIA sau, vom arăta că, transformă un flux ϵ -optimal într-un flux $\frac{\epsilon}{2}$ -optimal. Astfel, după $(1 + \lceil \log n\bar{b} \rceil)$ faze de scalare a costului, $\epsilon < 1/n$ şi, conform Teoremei 9.3, fluxul obţinut este un flux de cost minim.

Revenim acum la PROCEDURA ÎMBUNĂTĂŢEŞTE - APROXIMAŢIA. După execuţia instrucţiunilor din liniile (3) - (7) se obţine un pseudoflux 0 -optimal, deci şi $\frac{\epsilon}{2}$ -optimal. Vom arăta că pe parcursul execuţiei procedurii pseudofluxul rămâne $\frac{\epsilon}{2}$ -optimal, iar la sfârşitul ei se obţine un flux $\frac{\epsilon}{2}$ -optimal. Mărirea fluxului cu g unităţi pe arcul admisibil (x, y) păstrează $\frac{\epsilon}{2}$ -optimalitatea pseudofluxului deoarece arcul invers pe care-l poate adăuga în reţeaua reziduală îndeplineşte condiţia $b^p(y, x) \geq -\frac{\epsilon}{2}$ (pentru că $b^p(y, x) = -b^p(x, y) > 0$). Mărirea potenţialului nodului x se efectuează doar când nu există arce admisibile care ies din x (adică $b^p(x, y) \geq 0$). După actualizarea potenţialului $p(x)$ vom avea $b^p(x, y) \geq -\frac{\epsilon}{2}$, deci este îndeplinită condiţia de $\frac{\epsilon}{2}$ -optimalitate. Actualizarea potenţialului nodului x duce şi la mărirea cu $\frac{\epsilon}{2}$ a costurilor reduse ale arcelor care intră în nodul x, care, evident, vor îndeplini în continuare condiţiile de optimalitate. Deci, pe parcursul execuţiei PROCEDURII ÎMBUNĂTĂŢEŞTE - APROXIMAŢIA pseudofluxul este $\frac{\epsilon}{2}$ -optim, iar la sfârşit se obţine un flux $\frac{\epsilon}{2}$ -optim. ■

Teorema 9.5. *Algoritmul Goldberg-Tarjan al scalării costului are complexitatea* $O\left(n^2 m \log(n\bar{b})\right)$.

Demonstraţie. Pentru demonstraţie se recomandă cititorului monografiile precizate la bibliografie. ■

9.3 Algoritmul Ahuja - Goldberg - Orlin - Tarjan (AGOT) al scalării duble

Pentru a putea aplica algoritmul AGOT al scalării duble, mai întâi transformăm problema de flux de cost minim într-o problemă de transport cu arcele de capacitate infinită. Presupunem că fiecare arc din problema de flux de cost minim are capacitate finită. Înlocuim fiecare arc (x, y) cu două arce (x, z) şi

(y, z), introducând un nou nod z a cărui valoare va fi $-c(x, y)$.

Dacă introducem variabila $q(x, y) \geq 0$ putem rescrie condiţia de mărginire a fluxului pe arcul (x, y), $f(x, y) \leq c(x, y)$, în formă de egalitate:

$$f(x, y) + q(x, y) = c(x, y),$$

sau echivalent,

$$-f(x, y) - q(x, y) = -c(x, y) \tag{9.3}$$

Această ultimă relaţie va fi considerată ca fiind constrângerea de conservare a fluxului pentru nodul z. Deoarece $f(x, y)$ apare în trei relaţii de conservare a fluxului, iar $q(x, y)$ doar în una, vom scădea relaţia (9.3) din relaţia de conservare a fluxului pentru nodul y. Astfel atât $f(x, y)$ cât şi $q(x, y)$ vor apărea în exact două constrângeri de mărginire a fluxului: într-una cu semnul plus, iar în cealaltă cu minus. Transformarea astfel obţinută este ilustrată în figura 9.3.

Fig.9.3.

Dacă $f(x, y)$ este fluxul de pe arcul (x, y) în reţeaua originară, atunci fluxul corespunzător din reţeaua transformată va fi $f'(x, y) = f(x, y)$ şi $f'(y, z) = c(x, y) - f(x, y)$. Evident, fluxurile f şi f' au acelaşi cost. Invers, dacă avem fluxurile $f'(x, z)$ şi $f'(y, z)$ din reţeaua transformată, obţinem fluxul $f(x, y) = f'(x, y)$ în reţeaua iniţială. Evident, fluxul f are acelaşi cost ca fluxul f'.

Exemplul 9.3. Ilustrăm transformarea în urma căreia se elimină capacităţile arcelor pe reţeaua din figura 9.4 (a). Reţeaua transformată este bipartită şi este reprezentată în figura 9.4 (b).

Fig.9.4.

Reţeaua obţinută în urma transformării este bipartită $G = (N_1 \cup N_2, A)$, unde N_1 este mulţimea nodurilor ofertă, N_2 este mulţimea nodurilor cerere, $|N_1| = n$ şi $|N_2| = m$.

Algoritmul AGOT al scalării duble efectuează faze de scalare a costului pentru a obţine fluxuri ϵ -optimale pentru valori din ce în ce mai mici ale parametrului ϵ. Pe parcursul unei faze de scalare a costului, se porneşte de la un pseudoflux şi se efectuează mai multe faze de scalare a capacităţii, numite faze de $\overline{r}$ -scalare. Pe parcursul unei faze de $\overline{r}$ -scalare, algoritmul determină drumurile admisibile de la nodurile x cu $e(x) \geq \overline{r}$ la noduri cu deficit şi măreşte cu $\overline{r}$ unităţi fluxul de-a lungul acestor drumuri. Când toate excesele nodurilor sunt mai mici decât $\overline{r}$, se înjumătăţeşte valoarea lui $\overline{r}$ şi se începe o nouă fază de $\overline{r}$ -scalare. La sfârşitul fazei de 1 -scalare se obţine un flux.

Algoritmul AGOT al scalării duble este la fel ca algoritmul Goldberg-Tarjan al scalării costului, în care se elimină linia (3) şi se înlocuieşte PROCEDURA ÎMBUNĂTĂŢEŞTE - APROXIMAŢIA cu următoarea procedură:

(1) PROCEDURA ÎMBUNĂTĂŢEŞTE - APROXIMAŢIA (ϵ, f, p);
(2) BEGIN
(3) $f := 0$;
(4) se determină excesele / deficitele nodurilor;
(5) FOR $y \in N_2$ DO
(6) $p(y) := p(y) + \epsilon$;
(7) $\overline{r} := 2^{\lfloor \log \overline{c} \rfloor}$;
(8) WHILE în reţeaua $\tilde{G}(f)$ există un nod activ DO
(9) BEGIN
(10) $S(\overline{r}) := \{x \in N_1 \cup N_2 \mid e(x) \geq \overline{r}\}$;
(11) WHILE $S(\overline{r}) \neq \emptyset$ DO
(12) BEGIN
(13) se selectează un nod $x \in S(\overline{r})$;
(14) se determină un drum admisibil D de la nodul x la un nod z cu $e(z) < 0$;
(15) se măreşte fluxul cu $\overline{r}$ unităţi de-a lungul drumului D;
(16) se actualizează f şi $S(\overline{r})$;
(17) END;
(18) $\overline{r} := \overline{r}/2$;
(19) END;
(20) END;

Algoritmul AGOT al scalării duble este o îmbunătăţire a algoritmului Orlin al scalării capacităţii deoarece determină mai eficient drumurile admisibile de-a lungul cărora se măreşte fluxul. Determinarea unui drum admisibil D care porneşte din nodul x se face plecând de la un drum parţial admisibil care iniţial este un nod şi la care se adaugă arce până se ajunge la un nod cu deficit. Memorarea drumului parţial admisibil D se face cu ajutorul vectorului predecesor $\tilde{p}$: dacă $(u, v) \in D$ atunci $\tilde{p}(v) = u$. În funcţie de situaţie se va efectua una din următoarele operaţii:

înaintare (u) Dacă în reţeaua reziduală există un arc admisibil (u, v) atunci $D := D \cup \{(u, v)\}$ şi $\tilde{p}(v) := u$. Dacă $e(v) < 0$ atunci STOP .

înapoiere (u) Dacă în reţeaua reziduală nu există arce admisibile (u, v) atunci $p(u) = p(u) + \epsilon/2$. Dacă $u \neq x$ atunci $D = D \setminus \{(\tilde{p}(u), u)\}$.

Scopul operatiei de înapoiere este de a crea arce admisibile (u, v) prin mărirea potenţialului nodului u.

Exemplul 9.4. Ilustrăm prima fază de scalare a costului (pentru $\epsilon = 3$) efec-

tuată de algoritmul AGOT al scalării duble pe reţeaua din figura 9.5 (a). Mai întâi transformăm problema de flux minim din reţeaua din figura 9.5 (a) într-o problemă de transport cu arcele de capacitate infinită, care este reprezentată în figura 9.5 (b). În figura 9.5 (c) este reţeaua reziduală obţinută după iniţializările din liniile (3) - (7) ale PROCEDURII ÎMBUNĂTĂŢEŞTE - APROXIMAŢIA, $\epsilon = 3$, $\overline{r} = 2$ şi $S(2) = \{1, 2\}$. Presupunem că se selectează nodul $1 \in S(2)$. Se iniţializează $D = \emptyset$. Deoarece nu există arce admisibile care ies din 1, se aplică operaţia înapoiere (1) şi se actualizează potenţialul nodului 1: $p(1) = 3/2$, $p(1) = 3$, apoi $p(1) = 9/2$, creându-se astfel arcul admisibil (1,4). Rezultă că se va efectua operaţia înaintare (1) $D = \{(1, 4)\}$, $\tilde{p}(4) = 1$. Cum $e(4) = -2 < 0$, se măreşte cu două unităţi fluxul pe arcul (1,4) şi $S(2) = \{2\}$. Apoi se selectează nodul 2, care nu are arce admisibile. Deci, se va actualiza potenţialul nodului 2: $p(2) = 3/2$, $p(2) = 3$, apoi $p(2) = 9/2$, creându-se astfel arcele admisibile (2,4) şi (2,6). Presupunem că în operaţia înaintare (2) se alege arcul (2,6). Rezultă că $D = \{(2, 6)\}$, $\tilde{p}(6) = 2$ şi se măreşte cu două unităţi fluxul pe arcul (2,6). După efectuarea măririi de flux $S(2) = \emptyset$ şi se încheie faza de 2 -scalare. Reţeaua reziduală este ilustrată în figura 9.5 (d). Se efectuează $\overline{r} = 2/2 = 1$ şi se începe faza de 1 -scalare. Deoarece $S(1) = \{3\}$, se selectează nodul 3 şi se iniţializează $D = \emptyset$. Deoarece din nodul 3 nu ies arce admisibile, se aplică operaţia înapoiere (3) şi se măreşte potenţialul nodului 3: $p(3) = 3/2$, $p(3) = 3$, iar apoi $p(3) = 9/2$ creându-se astfel arcele admisibile (3,5) şi (3,6). Presupunem că este ales arcul (3,5). Rezultă că $D = \{(3, 5)\}$. Reţeaua obţinută în urma măririi fluxului este reprezentată în figura 9.5(e). Acum $S(1) = \emptyset$, se încheie faza de 1 -scalare şi se încheie şi faza de scalare a costului pentru $\epsilon = 3$. Se înjumătăţeşte valoarea lui ϵ şi se începe o nouă fază de scalare a costului.

Fig.9.5.

Teorema 9.6. *Algoritmul AGOT al scalării duble determină corect un flux de cost minim.*
Demonstraţie. La începutul PROCEDURII ÎMBUNĂTĂŢEŞTE - APROXIMAŢIA se efectuează $f := 0$, ceea ce înseamnă că reţeaua reziduală coincide cu reţeaua originară. Deoarece la sfârşitul fazei de scalare a costului corespunzătoare lui 2ϵ fluxul obţinut este ϵ -optimal rezultă că $b^p(x,y) \geq -\epsilon$ pentru toate arcele $(x,y) \in A$. Deci, prin actualizarea potenţialelor din liniile (5) şi (6) ale PROCEDURII ÎMBUNĂTĂŢEŞTE - APROXIMAŢIA se obţine un pseudoflux care este 0 -optimal, deci şi $\frac{\epsilon}{2}$ -optimal. Pe parcursul execuţiei PROCEDURII ÎMBUNĂTĂŢEŞTE - APROXIMAŢIA se măreşte fluxul de-a lungul unor drumuri formate doar din arce admisibile şi se actualizează potenţialele nodurilor din care nu ies arce admisibile. Deci, pe parcursul execuţiei procedurii pseudofluxul rămâne $\frac{\epsilon}{2}$ -optimal, iar la sfârşitul ei devine un flux $\frac{\epsilon}{2}$ -optimal. După $(1 + \lceil \log n\,\overline{b} \rceil)$ faze de scalare a costului, $\epsilon < 1/n$ şi conform Teoremei 9.3, fluxul obţinut care este ϵ -optimal este un flux de cost minim. ■

Teorema 9.7. *Algoritmul AGOT al scalării duble are complexitatea* $O\left(nm \log \overline{c} \log(n\overline{b})\right)$.
Pentru demonstraţie se recomandă cititorului monografiile precizate la bibliografie.

9.4 Algoritmul Sokkalingam - Ahuja - Orlin

Acest algoritm a fost obţinut aplicând tehnica scalării asupra algoritmului Klein de eliminare a circuitelor negative din reţeaua reziduală. Algoritmul Sokkalingam-Ahuja-Orlin măreşte fluxul de-a lungul circuitelor de cost negativ care au capacitatea reziduală "suficient de mare". Mai întâi vom arăta cum putem determina astfel de circuite folosind un algoritm de drum minim într-o reţea cu lungimile arcelor pozitive.

Spunem că un arc (x,y) din reţeaua reziduală este admisibil dacă $b^p(x,y) < 0$; în caz contrar este inadmisibil. Un arc admisibil (x,y) este $\overline{r}$ -admisibil dacă $r(x,y) \geq \overline{r}$.

PROCEDURA DETERMINĂ - CIRCUIT testează dacă există un circuit de cost negativ care să conţină arcul $\overline{r}$ -admisibil (u,w) care este dat ca parametru de intrare. Dacă există un astfel de circuit, atunci variabila ℓ va avea valoarea TRUE, iar $\overset{\circ}{D}$ va fi un astfel de circuit; dacă nu ℓ va fi FALSE şi se vor actualiza potenţialele nodurilor astfel încât arcul (u,w) să devină inadmisibil. PROCE-

DURA DETERMINĂ - CIRCUIT este următoarea:

(1) PROCEDURA DETERMINĂ - CIRCUIT $\left((u,w),\ell,\overset{\circ}{D}\right)$;
(2) BEGIN
(3) $\ell := false$;
(4) se determină mulţimea S a nodurilor către care există drumuri de la w la reţeaua $\overline{r}$ -reziduală $\tilde{G}(f,\overline{r})\backslash\{(w,u)\}$;
(5) IF $u \notin S$
(6) THEN se definesc potenţialele
(7) $$p'(y) = \begin{cases} p(y) + |b^p(u,w)|, & \text{pentru } y \in S \\ p(y), & \text{pentru } y \notin S \end{cases}$$
(8) ELSE BEGIN
(9) se determină distanţele minime $d(x)$ şi drumurile minime $D(x)$ de la nodul w la toate nodurile $x \in S$ în reţeaua $\tilde{G}(f,\overline{r})\backslash\{(w,u)\}$ considerând $b'(x,y) = \max\{0, b^p(x,y)\}$ ca fiind lungimea arcului (x,y).
(10) fie $\overline{d} = \max\{d(y) \mid y \in S\}$
(11) se definesc potenţialele
$$p'(y) = \begin{cases} p(y) + \overline{d} - d(y), & y \in S \\ p(y), & y \notin S \end{cases}$$
(12) IF $b^{p'}(u,w) < 0$
(13) THEN BEGIN
(14) ℓ: = true;
(15) $\overset{\circ}{D}$:= $D(u) \cup \{(u,w)\}$;
(16) END;
(17) END;
(18) $p := p'$;
(19) END;

Teorema 9.8. PROCEDURA DETERMINĂ - CIRCUIT *nu creează arce admisibile.*

Demonstraţie. Procedura modifică doar potenţialele nodurilor din S. Deci, se modifică doar costurile reduse ale arcelor care au o extremitate sau ambele extremităţi în S.

Cazul 1. Arcul (x,y) din $\tilde{G}(f,\overline{r})$ are o extremitate în S. Din modul în care a fost definit S rezultă că nu există nici un drum de la un nod din S la un nod din $\overline{S}$. Deci, $x \in \overline{S}$ şi $y \in S$. Fie că se execută instrucţiunea din linia (7), fie cea din linia (11), $p'(y) \geq p(y)$, deci $b^p(x,y) \geq b^{p'}(x,y)$, ceea ce înseamnă că în acest caz nu se creează arce admisibile.

Cazul 2. Arcul (x, y) din $\tilde{G}(f, \overline{r})$ are ambele extremităţi în S. Dacă se efectuează instrucţiunea din linia (7) atunci $b^{p'}(x, y) = b^p(x, y)$, deci nu se creează arce admisibile. Dacă s-a efectuat instrucţiunea din linia (11) rezultă că

$$b^{p'}(x, y) = b(x, y) - \Big(p(x) + \overline{d} - d(x)\Big) + \Big(p(y) + \overline{d} - d(y)\Big)$$

$$b^{p'}(x, y) = b^p(x, y) + d(x) - d(y) \tag{9.4}$$

Deoarece d este vectorul distanţelor minime considerând $\max\{0, b^p(x, y)\}$ ca fiind lungimile arcelor rezultă că

$$d(y) \leq d(x) + \max\{0, b^p(x, y)\} \tag{9.5}$$

Înlocuind (9.5) în (9.4) obţinem

$$b^{p'}(x, y) \geq b^p(x, y) - \max\{0, b^p(x, y)\}.$$

Dacă $\max\{0, b^p(x, y)\} = 0$ atunci $b^{p'}(x, y) \geq b^p(x, y)$; iar dacă $\max\{0, b^p(x, y)\} = b^p(x, y)$ atunci $b^{p'}(x, y) \geq 0$.

Deci, în nici o situaţie nu se creează arce admisibile. ■

Teorema 9.9.a)*Dacă* PROCEDURA DETERMINĂ - CIRCUIT *identifică un circuit* $\overset{\circ}{D} = D(u) \cup \{(u, w)\}$ *atunci* $\overset{\circ}{D}$ *este un circuit de cost negativ.*

b) PROCEDURA DETERMINĂ - CIRCUIT fie identifică un circuit $\overset{\circ}{D}$ din $\tilde{G}(f, \overline{r})$ care conţine arcul (u, w) fie transformă arcul (u, w) într-un arc inadmisibil.

Demonstraţie. a) Fie (x, y) un arc oarecare din $D(u)$. Deoarece $D(u)$ este un drum minim de la w la u în $\tilde{G}(f, \overline{r})$, rezultă că $d(y) = d(x) + \max\{0, b^p(x, y)\} \geq d(x) + b^p(x, y)$. Adică $b^{p'}(x, y) \leq 0$.

Procedura determină un circuit doar dacă $b^{p'}(u, w) < 0$. Deci

$$\sum_{(x,y)\in \overset{\circ}{D}} b^{p'}(x, y) = \sum_{(x,y)\in D(u)} b^{p'}(x, y) + b^{p'}(u, w) < 0.$$

Conform Proprietăţii 8.2 (b) rezultă că $\sum_{(x,y)\in \overset{\circ}{D}} b(x, y) = \sum_{(x,y)\in \overset{\circ}{D}} b^{p'}(x, y) < 0$, deci $\overset{\circ}{D}$ este un circuit de cost negativ.

b) Considerăm două cazuri: $u \notin S$ şi $u \in S$.

Dacă $u \notin S$ atunci după execuţia instrucţiunii din linia (7) $p'(w) = p(w) + |b^p(u, w)|$ şi $p'(u) = p(u)$. Deci, $b^{p'}(u, w) = b(u, w) - p'(u) + p'(w) = b(u, w) - p(u) + p(w) + |b^p(u, w)| = b^p(u, w) + |b^p(u, w)| = 0$, deci arcul (u, w) devine inadmisibil.

Dacă $u \in S$ atunci se modifică potenţialele în linia (11). Fie $b^{p'}(u,w) \geq 0$, caz în care arcul (u,w) este inadmisibil, fie $b^{p'}(u,w) < 0$ caz în care se determină circuitul $\overset{\circ}{D} = D(u) \cup \{(u,w)\}$ care este un circuit de cost negativ, conform punctului a). ■

Teorema 9.10. *Complexitatea* PROCEDURII DETERMINĂ - CIRCUIT *este* $O(m + n \log n)$.

Demonstraţie. Operaţia efectuată de procedură care consumă cel mai mult timp este determinarea drumurilor minime de la nodul w la celelalte noduri într-o reţea cu lungimile arcelor pozitive. Dacă se foloseşte implementarea cu heapuri Fibonacci a algoritmului Dijkstra pentru determinarea drumurilor minime, complexitatea procedurii este $O(m + n \log n)$. ■

Algoritmul Sokkalingam-Ahuja-Orlin efectuează faze de $\overline{r}$ -scalare, pentru valori ale parametrului $\overline{r}$ din ce în ce mai mici. Algoritmul Sokkalingam-Ahuja-Orlin este următorul:

```
(1) PROGRAM SAO;
(2) BEGIN
(3)     se determină un flux f cu valori întregi;
(4)     p := 0;
(5)     r̄ := mc̄;
(6)     WHILE r̄ ≥ 1/2 DO
(7)     BEGIN
(8)                 ELIMINĂ - CIRCUITE;
(9)                 r' = max{r(x,y) | (x,y) este arc admisibil din G̃(f)};
(10)                r̄ := r'/2;
(11)    END;
(12) END.
```

```
(1) PROCEDURA ELIMINĂ - CIRCUITE;
(2) BEGIN
(3)     WHILE în G̃(f) există un arc r̄ -admisibil DO
(4)     BEGIN
(5)       se selectează un arc r̄ -admisibil (u,w);
(6)       DETERMINĂ - CIRCUIT ((u,w), ℓ, D°);
(7)       IF ℓ
(8)         THEN BEGIN
(9)                 g := min{r(x,y) | (x,y) ∈ D°};
(10)                se măreşte cu g unităţi fluxul de-a lungul circuitului D°;
(11)                se actualizează f;
```

(12) END;
(13) END;
(14) END.

Exemplul 9.5. Ilustrăm algoritmul Sokkalingam-Ahuja-Orlin pe reţeaua din figura 9.6 (a), în care este reprezentat un flux admisibil al cărui cost este $\sum_{(x,y)\in A} b(x,y)\cdot f(x,y) = 2\cdot 2+4\cdot 2+1\cdot 0+4\cdot 2+2\cdot 2 = 24$. Se fac iniţializările $p = 0$, $\overline{r} = 15$. Reţeaua reziduală $\tilde{G}(f)$ reprezentată în figura 9.6 (b) nu conţine nici un arc $\overline{r}$ -admisibil, deci se determină $r' = \max\{r(2,1), r(3,1), r(4,2), r(4,3)\} = 2$, $\overline{r} = r'/2 = 1$ şi se încheie prima fază de $\overline{r}$ -scalare. În faza de 1 -scalare, în reţeaua reziduală există arce 1 -admisibile. Presupunem că este ales arcul (4,2). Se va apela PROCEDURA DETERMINĂ - CIRCUIT $((4,2), \ell, \overset{\circ}{D})$. Se determină mulţimea $S = \{1,2,3,4\}$ a nodurilor către care există drumuri în $\tilde{G}(f,1) \setminus \{(2,4)\} = \tilde{G}(f) \setminus \{(2,4)\}$ pornind din nodul 2.

Fig.9.6.

Vectorul distanţelor minime de la nodul 2 la celelalte noduri în $\tilde{G}(f) \setminus \{(2,4)\}$ este $d = (0,0,1,3)$. Se determină $d' = 3$ şi $p' = (3,3,2,0)$. Deoarece $b^{p'}(4,2) = b(4,2) - p'(4) + p'(2) = -1 < 0$, $\ell = true$, iar $\overset{\circ}{D} = D(4) \cup \{(4,2)\} = \{(2,3),(3,4),(4,2)\}$. Se măreşte cu $g = \min\{r(2,3), r(3,4), r(4,2)\} = 1$ fluxul de-a lungul circuitului $\overset{\circ}{D}$, iar reţeaua reziduală astfel obţinută este ilustrată în figura 9.6 (c) şi mai conţine arce 1 -admisibile. Presupunem că este ales arcul (3,1). Se va apela PROCEDURA DETERMINĂ - CIRCUIT $((3,1), \ell, \overset{\circ}{D})$, se determină $S = \{1\}, p'(1) = 3 + 3 = 6$ şi astfel arcele (3,1) şi (2,1) devin inadmisibile. Reţeaua reziduală obţinută este reprezentată în figura 9.6 (d). Singurul arc 1 -admisibil este (4,2), deci se apelează PROCEDURA DETERMINĂ - CIRCUIT $((4,2), \ell, \overset{\circ}{D})$. Se vor determina $S = \{1,2,3\}$ şi $p' = (7,4,3,0)$. Reţeaua reziduală astfel obţinută este reprezentată în figura 9.6 (e) şi nu conţine arce 1 -admisibile. Se determină $r' = 0, \overline{r} = 0$ şi se termină algoritmul cu un flux de cost minim care este reprezentat în figura 9.6 (f). Costul acestui flux este $\sum_{(x,y)\in A} b(x,y)\cdot f(x,y) = 2\cdot 2 + 4\cdot 2 + 1\cdot 1 + 4\cdot 1 + 2\cdot 3 = 23$.

Teorema 9.11. *Într-o fază de $\overline{r}$ -scalare,* PROCEDURA ELIMINĂ - CIRCUITE *îndeplineşte următoarele proprietăţi:*
(a) La începutul procedurii, $r(k,\ell) \le 2\overline{r}$ pentru toate arcele admisbile (x,y).
(b) La fiecare iteraţie, numărul arcelor $\overline{r}$ -admisibile descreşte şi capacităţile reziduale ale arcelor admisibile nu cresc.
(c) La sfârşitul procedurii, $r(x,y) \le \overline{r}$, pentru toate arcele admisibile (x,y).
Demonstraţie. (a) Înainte de a începe o nouă fază de scalare, algoritmul efectuează

$\overline{r} := r'/2$, unde r' este maximul capacităţilor reziduale ale arcelor admisibile. Deci, $r(x,y) \leq 2\overline{r}$ pentru orice arc admisibil (x,y).

(b) În demonstraţia Teoremei 9.9 (a), am arătat ca $b^p(x,y) \leq 0, \forall (x,y) \in \overset{\circ}{D}$. Mărirea fluxului de-a lungul circuitului $\overset{\circ}{D}$ duce la micşorarea capacităţilor reziduale ale arcelor din $\overset{\circ}{D}$ (care au costurile reduse negative sau nule) şi la mărirea capacităţilor reziduale ale arcelor inverse lor (care au costurile reduse pozitive sau nule). Deci, capacităţile reziduale ale arcelor admisibile nu cresc. La execuţia PROCEDURII DETERMINĂ - CIRCUIT $((u,w), \ell, \overset{\circ}{D})$, fie arcul (u,w) devine inadmisibil după actualizarea potenţialelor, fie se măreşte fluxul de-a lungul circuitului $\overset{\circ}{D}$ cu cel puţin $\overline{r}$ unităţi, caz în care $r(u,w) \leq \overline{r}$, deci (u,w) nu mai este $\overline{r}$ -admisibil.

(c) PROCEDURA ELIMINĂ - CIRCUITE se termină atunci când nu mai există arce $\overline{r}$ -admisibile, adică $r(x,y) \leq \overline{r}$, pentru orice arc admisibil (x,y). ■

Teorema 9.12. *Algoritmul Sokkalingam-Ahuja-Orlin determină corect un flux de cost minim.*
Demonstraţie. După ultima fază de $\overline{r}$ -scalare efectuată de algoritm, $\overline{r} < 1$. Deoarece capacităţile reziduale sunt întregi, rezultă conform Teoremei 9.11 (c) că în reţeaua reziduală $\tilde{G}(f)$ nu există arce admisibile, adică $b^p(x,y) \geq 0$, pentru orice arc (x,y) din $\tilde{G}(f)$.Deci, conform Teoremei 8.3, f este un flux de cost minim. ■

Teorema 9.13. *Complexitatea algoritmului Sokkalingam-Ahuja-Orlin este* $O\left(m(m + n\log n)\log(n\overline{c})\right)$.
Demonstraţie. Din Teorema 9.11 (b) rezultă că într-o fază de $\overline{r}$ -scalare se efectuează cel mult m iteraţii, deci complexitatea unei faze de $\overline{r}$ -scalare este $O\left(m(m + n\log n)\right)$.

La începutul PROCEDURII ELIMINĂ - CIRCUITE, $r(x,y) \leq 2\overline{r}$ pentru toate arcele admisibile (x,y) (conform Teoremei 9.11 (a)), iar la sfârşitul procedurii $r(x,y) \leq \overline{r}$, pentru toate arcele admisibile (x,y) (conform Teoremei 9.11 (c)). Deci, noua valoare a lui $\overline{r}$, care este $r'/2$, este mai mică decât jumătate din vechea valoare. Iniţial, $\overline{r} = m\overline{c}$, la sfârşitul algoritmului $\overline{r} < 1/2$, iar la fiecare fază de scalare $\overline{r}$ este micşorat la cel mult jumătate. Rezultă că numărul fazelor de $\overline{r}$ -scalare este $O\left(\log(m\overline{c})\right) = O(\log(n\overline{c}))$. Deci, complexitatea algoritmului Sokkalingam-Ahuja-Orlin este $O\left(m(m + n\log n)\log(n\overline{c})\right)$. ■

În continuare se prezintă o implementare tare-polinomială a algoritmului Sokkalingam-Ahuja-Orlin.

Pentru a simplifica prezentarea implementării tare-polinomială a algoritmului

Sokkalingam-Ahuja-Orlin, presupunem că sunt îndeplinite următoarele ipoteze:
Ipoteza (a) Toate arcele din reţeaua G au capacitate infinită.
Ipoteza (b) Pentru orice flux f, în reţeaua reziduală $\tilde{G}(f)$ nu există circuite netriviale de cost zero. (Un circuit trivial este format dintr-un arc (x, y) şi inversul său (y, x)).

Nici una dintre aceste ipoteze nu restrânge generalitatea problemei. Pentru a îndeplini ipoteza (a) se efectuează transformarea pe care am prezentat-o în paragraful 9.3 (vezi figura 9.3), iar pentru a îndeplini ipoteza (b) se poate efectua o mărire cu o constantă a costurilor arcelor.

Spunem că arcul (x, y) din reţeaua reziduală este *acceptabil* dacă $b^p(x, y) \leq 0$. Un circuit $\overset{\circ}{D}$ din reţeaua reziduală format doar din arce acceptabile se numeşte *circuit acceptabil*.

Implementarea tare-polinomială a algoritmului Sokkalingam-Ahuja-Orlin este următoarea:

(1) PROGRAM SAO - ÎMBUNĂTĂŢIT;
(2) BEGIN
(3) se determină un flux f cu valori întregi;
(4) $p := 0$;
(5) $\overline{r} := m\overline{c}$;
(6) WHILE $D \geq 1/2$ DO
(7) BEGIN
(8) ELIMINĂ - CIRCUITE;
(9) ELIMINĂ - CIRCUITE - ACCEPTABILE;
(10) $r' = \max\{r(x, y) \mid (x, y)$ este arc admisibil din $\tilde{G}(f)\}$;
(11) $\overline{r} := r'/2$;
(12) END;
(13) END.

(1) PROCEDURA ELIMINĂ - CIRCUITE - ACCEPTABILE;
(2) BEGIN
(3) WHILE în $\tilde{G}(f)$ există circuite acceptabile DO
(4) BEGIN
(5) se selectează un circuit acceptabil $\overset{\circ}{D}$;
(6) se determină $g := \min\{r(x, y) \mid (x, y) \in \overset{\circ}{D}\}$;
(7) se măreşte cu g unităţi fluxul de-a lungul circuitului $\overset{\circ}{D}$;
(8) END;
(9) END;

(1) PROCEDURA ELIMINĂ - CIRCUITE;

(2) BEGIN
(3) WHILE în $\tilde{G}(f)$ există un arc $\overline{r}$ -admisibil DO
(4) BEGIN
(5) se selectează un arc $\overline{r}$ -admisibil (u, w);
(6) DETERMINĂ - CIRCUIT $\left((u, w), \ell, \overset{\circ}{D}\right)$;
(7) IF ℓ
(8) THEN BEGIN
(9) $g := \min\{r(x, y) \,|\, (x, y) \in \overset{\circ}{D}\}$;
(10) se măreşte cu g unităţi fluxul de-a lungul circuitului $\overset{\circ}{D}$;
(11) se actualizează f;
(12) END;
(13) END;
(14) END.

Teorema 9.14. *Complexitatea algoritmului Sokkalingam-Ahuja-Orlin îmbunătăţit este* $O\left(m(m + n \log n) \min\{\log(n\overline{c}), m \log n\}\right)$.
Demonstraţie. Pentru demonstraţie se recomandă cititorului să consulte bibliografia. ■

9.5 Algoritmul Goldberg - Tardos - Tarjan al scalării repetate a capacităţii

Algoritmul Goldberg - Tardos - Tarjan al scalării repetate a capacităţii este o versiune îmbunătăţită a agoritmului Orlin al scalării capacităţii şi determină mai întâi potenţialele nodurilor optime, iar apoi, pornind de la acestea, un flux de cost minim. Pentru simplitate, vom descrie algoritmul Goldberg - Tardos - Tarjan pentru flux de cost minim în reţele cu capacităţile arcelor infinite.

Proprietatea 9.15. *Algoritmul Orlin al scalării capacităţii pentru fluxul de cost minim în reţele cu capacităţile arcelor infinite îndeplineşte următoarele proprietăţi:*

(a) *excesele nodurilor descresc*
(b) *suma exceselor nodurilor la începutul fazei de* $\overline{r}$ *- scalare este cel mult* $2n\overline{r}$
(c) *algoritmul efectuează cel mult* $2n$ *măriri de flux în fiecare fază de* $\overline{r}$ *- scalare.*

Demonstraţie. (a) Deoarece algoritmul se aplică reţelelor cu capacităţi infinite, capacităţile reziduale ale arcelor sunt multipli de $\overline{r}$. Deci, $\tilde{G}(f, \overline{r}) = \tilde{G}(f)$, ceea ce înseamnă că nu se execută liniile (8) - (13) din algoritmul Orlin al scalării capacităţii în care se saturau arcele care nu îndeplineau condiţiile de optimalitate

şi care ar fi putut duce la creşterea exceselor. Deoarece algoritmul măreşte fluxul pe drumuri de la noduri cu exces la noduri cu deficit, rezultă că excesele nodurilor scad.

(b) La sfârşitul fazei de $2\overline{r}$ - scalare, $S(2\overline{r}) = \emptyset$ sau $T(2\overline{r}) = \emptyset$, ceea ce înseamnă că suma exceselor este mai mică de $2n\overline{r}$. Deci, la începutul fazei de $\overline{r}$ - scalare, suma exceselor este mai mică decât $2n\overline{r}$.

(c) Algoritmul efectuează cel mult $2n$ măriri de flux în fiecare fază de $\overline{r}$ - scalare, deoarece la fiecare mărire de flux se trimit $\overline{r}$ unităţi de flux de-a lungul unui drum de la un nod cu exces la un nod cu deficit, iar suma exceselor nodurilor la începutul fazei de $\overline{r}$ - scalare este cel mult $2n\overline{r}$.■

Teorema 9.16. (a) *Dacă la începutul fazei de $\overline{r}$ - scalare există un nod z cu $v(z) > 6n^2\overline{r}$ atunci există un arc incident către exterior (z, w) cu $f(z, w) > 4n\overline{r}$.*

(b) *Dacă la începutul fazei de $\overline{r}$ - scalare , $f(z, w) > 4n\overline{r}$ atunci există un flux de cost minim astfel încât $f(z, w) > 0$.*

(c) *Dacă $f(z, w) > 0$ şi f este un flux de cost minim atunci pentru orice potenţiale optime, costul redus al arcului (z, w) este nul.*

Demonstraţie. (a) Conform Proprietăţii 9.15(b), suma exceselor nodurilor la începutul fazei de $\overline{r}$ - scalare este cel mult $2n\overline{r}$. Deci, $e(z) \leq 2n\overline{r}$. Dar $f(z, N) - f(N, z) = v(z) - e(z)$. Rezultă că $f(z, N) > 6n^2\overline{r} - 2n\overline{r}$. Cum numărul de arce incidente către exterior cu z este mai mic decât n, rezultă că există un arc (z, w) astfel încât $f(z, w) > \left(6n^2\overline{r} - 2n\overline{r}\right)/n \geq 4n^2\overline{r}/n = 4n\overline{r}$.

(b) Conform Proprietăţii 9.15(c), algoritmul efectuează cel mult $2n$ măriri de flux în fiecare fază de $\overline{r}$ - scalare. Deoarece într-o fază de $\overline{r}$ - scalare, după fiecare mărire de flux, valoarea fluxului creşte cu $\overline{r}$ unităţi, rezultă că valoarea totală cu care se modifică fluxul pe arce până la sfârşitul execuţiei algoritmului este cel mult

$$2n\left(\overline{r} + \overline{r}/2 + \overline{r}/4 + \ldots + 1\right) < 4n\overline{r}.$$

Deci, dacă $f(z, w) > 4n\overline{r}$ la începutul fazei de $\overline{r}$ - scalare atunci $f(z, w) > 0$ la sfârşitul execuţiei algoritmului.

(c) Fie p un vector de potenţiale astfel încât f îndeplineşte condiţiile de optimalitate cu ecarturi complementare în raport cu potenţialele p. Deoarece $f(z, w) > 0$ rezultă că $b^p(z, w) = 0$. Conform Proprietăţii 8.8, dacă f îndeplineşte condiţiile de optimalitate cu ecarturi complementare (8.3b) în raport cu potenţialele p, atunci f îndeplineşte aceste condiţii în raport cu orice potenţiale optime. Deci, costul redus al arcului (z, w) va fi nul pentru orice potenţiale optime. ■

Fie P - problema fluxului de cost minim pe care dorim să o rezolvăm.

Algoritmul Goldberg - Tardos - Tarjan al scalării repetate constă în a aplica de mai multe ori, pe reţele din ce în ce mai mici, algoritmul Orlin al scalării

capacităţii modificat. Se iniţializează $\overline{c} = \max\{v(x) \mid x \in N \text{ şi } v(x) > 0\}$, $\overline{r} = 2^{\lfloor \log \overline{c} \rfloor}$. Fie z nodul pentru care $v(z) = \overline{c}$. Se aplică algoritmul Orlin al scalării capacităţii pentru reţele cu capacităţile infinite. După cel mult $q = \log(6n^2) = O(\log n)$ faze de scalare $\overline{r} = v(z)/2^q \leq v(z)/(6n^2)$. Dacă s-a obţinut un flux atunci algoritmul se termină; în caz contrar, conform Teoremei 9.16(a), există un arc (z, w) astfel încât $f(z, w) > 4n\overline{r}$. Conform Teoremei 9.16(b, c), costul redus al arcului (z, w) va fi nul, pentru orice potenţiale optime. Fie p vectorul curent de potenţiale obţinut prin aplicarea algoritmului Orlin al scalării capacităţii pentru problema P. Din condiţiile de optimalitate cu ecarturi complementare (8.3b) rezultă că

$$b^p(z, w) = 0$$

adică

$$b(z, w) - p(z) + p(w) = 0. \tag{9.6}$$

Fie P' - problema fluxului de cost minim obţinută din problema P prin înlocuirea costurilor arcelor cu $b'(x, y) = b(x, y) - p(x) + p(y)$, pentru orice $(x, y) \in A$.

Din relaţia (9.6) rezultă că

$$b'(z, w) = 0. \tag{9.7}$$

Deoarece P şi P' are aceeaşi soluţie optimă, iar $f(z, w) > 4n\overline{r}$, conform Teoremei 9.16(b, c) rezultă că şi în soluţia problemei P' costul redus al arcului (z, w) va fi nul. Fie p' - un vector de potenţiale optime pentru problema P'. Deci,

$$b'^{p'}(z, w) = 0$$

adică,

$$b'(z, w) - p'(z) + p'(w) = 0. \tag{9.8}$$

Din relaţiile (9.7) şi (9.8), rezultă că $p'(z) = p'(w)$. Deci, putem contracta nodurile z şi w într-un nod u astfel
(1) considerând $v(u) = v(z) + v(w)$
(2) înlocuind toate arcele (x, z) şi (x, w) cu (x, u)
(3) înlocuind toate arcele (z, x) şi (w, x) cu (u, x)
(4) considerând costurile arcelor nou introduse ca fiind egale cu costurile reduse ale arcelor pe care le-au înlocuit.

Apoi se recalculează $\overline{c}$ care este egal cu cea mai mare valoare a nodurilor din reţeaua obţinută prin contracţia nodurilor z şi w. Se aplică algoritmul Orlin al scalării capacităţii modificat. Se repetă acest proces până când (1) se obţine un flux aplicând algoritmul Orlin sau (2) reţeaua obţinută prin contracţia a 2 noduri conţine un singur nod u (cu $v(u) = 0$), problemă a cărei soluţie este, evident,

$f(u) = 0$. În final, expandăm nodurile în ordinea inversă ordinii în care au fost contractate. Dacă nodul u a fost obţinut prin contracţia nodurilor z şi w atunci considerăm potenţialele nodurilor z şi w ca fiind egale cu potenţialul nodului u şi adunăm p la potenţialele existente. Algoritmul Goldberg - Tardos - Tarjan al scalării repetate a capacităţii este următorul:

(1) PROGRAM GTT;
(2) BEGIN
(3) $f := 0$;
(4) $p := 0$
(5) WHILE există noduri cu exces şi $|N| = n > 0$ DO
(6) BEGIN
(7) $\overline{c} := \max\{v(x) \mid x \in N$ şi $v(x) > 0\}$;
(8) fie z astfel încât $v(z) = \overline{c}$;
(9) $\overline{r} = 2^{\lfloor \log \overline{c} \rfloor}$;
(10) WHILE $\overline{r} > v(z)/(6n^2)$ DO
(11) BEGIN
(12) se construieşte reţeaua $\tilde{G}(f, \overline{r})$;
(13) $S(\overline{r}) := \{x \in N \mid e(x) \geq \overline{r}\}$;
(14) $T(\overline{r}) := \{x \in N \mid e(x) \leq -\overline{r}\}$;
(15) WHILE $S(\overline{r}) \neq \emptyset$ şi $T(\overline{r}) \neq \emptyset$ DO
(16) BEGIN
(17) se selectează un nod $s \in S(\overline{r})$ şi un nod $T \in T(\overline{r})$;
(18) se determină distanţele minime $d(\cdot)$ de la s la celelalte noduri în $\tilde{G}(f, \overline{r})$ în raport cu costurile reduse;
(19) se determină drumul minim D de la s la t în $\tilde{G}(f, \overline{r})$;
(20) $p := p - d$;
(21) se măreşte cu $\overline{r}$ unităţi fluxul pe drumul D;
(22) se actualizează f, $S(\overline{r})$, $T(\overline{r})$ şi $\tilde{G}(f, \overline{r})$;
(23) END;
(24) $\overline{r} := \overline{r}/2$;
(25) END;
(26) IF există noduri cu exces
(27) THEN BEGIN
(28) se determină arcul (z, w) cu $f(z, w) > 4n\overline{r}$;
(29) se contractă nodurile z şi w;
(30) se actualizează $G = (N, A)$, n, f, v, b, b^p;
(31) END;
(32) ELSE IF $n = 0$ THEN $f := 0$
(33) END;
(34) se expandează nodurile în ordinea inversă contracţiei;

(35) END.

Exemplul 9.6. Ilustrăm algoritmul Goldberg - Tardos - Tarjan al scalării repetate a capacităţii pe reţeaua cu capacităţile arcelor infinite din figura 9.7(a). După iniţializările din liniile (3) - (9) obţinem $f := 0$, $p = 0$, $\overline{c} = 2^{100} - 1$, $z = 1$ şi $\overline{r} = 2^{99}$. Se construieşte reţeaua $\tilde{G}(f, 2^{99})$ care este reprezentată în figura 9.7(b)

Fig.9.7.

şi se determină mulţimile $S\left(2^{99}\right)=\{1,2\}$ şi $T\left(2^{99}\right)=\{3,4\}$. Se selectează nodul $2\in S\left(2^{99}\right)$ şi nodul $4\in T\left(2^{99}\right)$.

Vectorul distanţelor minime de la 2 la celelalte noduri în $\tilde{G}\left(f,2^{99}\right)$ este $d=(0,0,1,3)$. Se actualizează vectorul potenţialelor: $p=(0,0,-1,-3)$ şi se măreşte fluxul cu 2^{99} unităţi de-a lungul drumului minim $D=(2,1,3,4)$. Reţeaua astfel obţinută este reprezentată în figura 9.7(c). Avem $S\left(2^{99}\right)=\{1\}$ şi $T\left(2^{99}\right)=\{3\}$, deci se va determina vectorul distanţelor minime de la 1 la celelalte noduri în $\tilde{G}\left(f,2^{99}\right)$: $d=(0,0,0,0)$. Potenţialele rămân nemodificate şi se măreşte fluxul cu 2^{99} unităţi pe drumul minim $D=(1,3)$. Reţeaua obţinută este ilustrată în figura 9.7(d). Deoarece $S\left(2^{99}\right)=T\left(2^{99}\right)=\emptyset$ se efectuează $r:=2^{98}$ şi se trece la o nouă fază de scalare. Deoarece $\overline{r}\geq v(1)/(6n^2)=\frac{2^{100}-1}{24}$, se determină mulţimile $S\left(2^{98}\right)=\{1\},T\left(2^{98}\right)=\{3\}$. Vectorul distanţelor minime de la 1 la celelalte noduri în $\tilde{G}(f,2^{98})$ este $d=(0,0,0,0)$. În figura 9.7(e) este reprezentată reţeaua obţinută după mărirea fluxului cu 2^{98} unităţi pe drumul minim $D=(1,3)$. Cum $S\left(2^{98}\right)=T\left(2^{98}\right)=\emptyset$, se efectuează $\overline{r}=2^{97}$ şi se trece la o nouă fază de scalare. În continuare $\overline{r}\geq v(1)/\left(6n^2\right)=\left(2^{100}-1\right)/24$, deci se determină mulţimile $S\left(2^{97}\right)=\{1\},T\left(2^{97}\right)=\{3\}$. Vectorul distanţelor minime de la nodul 1 la celelalte noduri în $\tilde{G}\left(f,2^{97}\right)$ este $d=(0,0,0,0)$, deci potenţialele rămân nemodificate, iar fluxul se măreşte cu 2^{97} unităţi de-a lungul drumului minim $D=(1,3)$. Deoarece $S\left(2^{97}\right)=T\left(2^{97}\right)=\emptyset$, se efectuează $\overline{r}:=2^{96}$ şi se începe o nouă fază de scalare, care se va încheia după mărirea cu 2^{96} unităţi a fluxului pe dumul minim $D=(1,3)$. În faza de 2^{95} - scalare se va mări fluxul pe drumul minim $D=(1,3)$ cu 2^{95} unităţi, iar în faza de 2^{94} - scalare cu 2^{94} unităţi, obţinându-se astfel reţeaua din figura 9.7(f). Se efectuează $\overline{r}:=2^{93}$. Acum $\overline{r}<v(z)/\left(6n^2\right)=\left(2^{100}-1\right)/24$ rezultă că se va determina arcul $(1,3)$ cu $f(1,3)=2^{100}+2^{98}+2^{97}+2^{96}+2^{95}+2^{94}>4\cdot n\cdot\overline{r}$ şi se vor contracta nodurile 1 şi 3, obţinându-se astfel reţeaua reprezentată în figura 9.7(g). Se determină $\overline{c}=2^{99}$, $z=2$, $\overline{r}=2^{99}$, $S\left(2^{99}\right)=\{2\},T\left(2^{99}\right)=\{4\}$. Drumul minim de la 2 la 4 în $\tilde{G}\left(f,2^{99}\right)$ este $D=(2,5,4)$ vectorul distanţelor minime fiind $d=(0,0,0)$, iar reţeaua obţinută după mărirea fluxului cu 2^{99} unităţi de-a lungul drumului D este reprezentată în figura 9.7(h). În reţeaua din figura 9.7(h) nu mai există noduri cu exces, deci vom expanda nodurile obţinând reţeaua din figura 9.7(i), în care sunt reprezentate potenţialele optime ale nodurilor.

Pornind de la potenţialele optime determinăm fluxul de cost minim din figura 9.7(k), prin rezolvarea unei probleme de flux maxim (a se vedea paragraful 8.5) în reţeaua transformată reprezentată în figura 9.7(j).

Teorema 9.17. *Algoritmul Goldberg - Tardos - Tarjan al scalării repetate a*

capacităţii determină corect un flux de cost minim într-o reţea cu capacităţile arcelor infinite.
Demonstraţie. Evident, conform Teoremei 9.1. ■
Teorema 9.18. *Complexitatea algoritmul Goldberg - Tardos - Tarjan pentru determinarea unui flux de cost minim într-o reţea cu capacităţile arcelor infinite este* $O\left(n^2 \log nD(n,m)\right)$, *unde* $D(n,m)$ *este complexitatea algoritmului de determinare a drumului minim într-o reţea cu* n *noduri,* m *arce şi cu lungimile arcelor pozitive.*
Demonstraţie. Într-o fază de $\overline{r}$ - scalare se efectuează cel mult $2n$ măriri de flux pentru că la începutul fazei de $\overline{r}$ - scalare suma exceselor nodurilor este mai mică decât $2n\overline{r}$, iar la sfârşitul fazei de $\overline{r}$ - scalare este nulă şi fiecare mărire de flux se face cu $\overline{r}$ unităţi. Deoarece pentru fiecare mărire de flux se determină un drum minim într-o reţea cu lungimile arcelor pozitive, rezultă că fiecare fază de scalare are complexitatea $O\left(nD(n,m)\right)$. Deoarece, iniţial $\overline{r} = 2^{\lfloor \log \overline{c} \rfloor}$ şi $\overline{c} = \max\{v(x) \,|\, x \in N,\ v(x) > 0\}$ şi la sfârşitul fiecărei faze de scalare valoarea lui $\overline{r}$ se înjumătăţeşte, rezultă că, după $O(\log n)$ faze de scalare, $\overline{r} \leq v(z)/\left(6n^2\right)$ şi, în cazul în care nu s-a obţinut deja un flux de cost minim, se efectuează o contracţie, reducându-se astfel cu 1 numărul de noduri din reţea. Deci, algoritmul efectuează cel mult $O\left(n \log n\right)$ faze de scalare, iar complexitatea lui este $O\left(n^2 \log nD(n,m)\right)$. ■

Deoarece cel mai eficient algoritm de determinare a drumurilor minime într-o reţea cu n noduri, cu m arce şi cu lungimile arcelor pozitive este algoritmul Dijkstra implementat cu heap-uri Fibonacci, care are complexitatea $O\left(m + n \log n\right)$, rezultă că algoritmul Goldberg - Tardos - Tarjan pentru determinarea unui flux de cost minim într-o reţea cu capacităţile arcelor infinite are complexitatea $O(n^2 \log n(m + n \log n))$.
Teorema 9.19. *Compexitatea algoritmul Goldberg - Tardos - Tarjan pentru determinarea unui flux de cost minim într-o reţea oarecare este* $O\left(m^2 \log n (m + n \log n)\right)$.
Demonstraţie. Mai întâi transformăm reţeaua într-o reţea cu capacităţile arcelor infinite (a se vedea Exemplul 9.3). Reţeaua transformată va avea $n' = n + m$ noduri şi $m' = 2m$ arce. Algoritmul Goldberg - Tardos - Tarjan va efectua $O\left(n' \log n'\right) = O\left(m \log n\right)$ faze de scalare, iar la fiecare fază de scalare va determina $O\left(n'\right) = O(m)$ drumuri minime. Deoarece putem determina un drum minim în reţeaua transformată, care este o reţea bipartită, cu complexitatea $O\left(m + n \log n\right)$, rezultă că algoritmul Goldberg - Tardos - Tarjan pentru determinarea unui flux de cost minim într-o reţea oarecare are complexitatea $O\left(m^2 \log n(m + n \log n)\right)$. ■

9.6 Algoritmul Orlin al scalării intensive a capacităţii

Algoritmul Orlin al scalării intensive a capacităţii este o versiune îmbunătăţită a algoritmului Orlin al scalării capacităţii. Pentru simplitate, vom descrie algoritmul pentru determinarea unui flux de cost minim în reţele cu capacităţile arcelor infinite.

Algoritmul Orlin al scalării intensive a capacităţii efectuează faze de scalare pentru diferite valori ale parametrului $\overline{r}$. Într-o fază de $\overline{r}$ - scalare, spunem că fluxul pe arcul (x, y) este suficient de mare dacă $f(x, y) \geq 8n\overline{r}$. În acest caz spunem că arcul (x, y) este un *arc abundent*, în caz contrar arcul (x, y) este un *arc neabundent*. Se numeşte *subgraf abundent* al grafului $G = (N, A)$ subgraful $G' = (N, A')$, unde $A' = \{(x, y) \,|\, (x, y) \in A$ şi $(x, y) -$ arc abundent$\}$. În general, un subgraf abundent are mai multe componente conexe, care se numesc *componente abundente*. O componentă abundentă poate fi reprezentată prin mulţimea S a nodurilor pe care le conţine. Notăm $v(S) = \sum_{x \in S} v(x)$ şi $e(S) = \sum_{x \in S} e(x)$. Pentru fiecare componentă abundentă, se alege un nod care va fi rădăcină. Prin convenţie, rădăcina unei componente abundente este nodul cu cel mai mic număr.

Pe parcursul execuţiei algoritmului Orlin al scalării intensive a capacităţii sunt îndeplinite următoarele proprietăţi:

Proprietatea 9.20. (Proprietatea fluxului) *Într-o fază de $\overline{r}$ - scalare, fluxul pe arcele neabundente este multiplu de $\overline{r}$, iar fluxul pe arcele abundente poate avea orice valoare nenegativă.*

Proprietatea 9.21. (Proprietatea excesului/ deficitului) *Doar nodurile rădăcină pot avea exces sau deficit.*

Algoritmul porneşte de la un flux nul, deci în reţea nu există arce abundente, ceea ce implică faptul că subgraful abundent are n componente abundente, fiecare componentă constând într-un nod. Pe parcursul execuţiei sale, algoritmul determină arcele abundente şi le adaugă la subgraful abundent. Fie (x, y) un arc abundent. Există două cazuri:

c_1) x şi y se găsesc în aceeaşi componentă abundentă. Rezultă că prin adăugarea arcului (x, y) nu se creează o nouă componentă abundentă.

c_2) x şi y se află în componente abundente diferite. Fie S_x şi S_y componentele care-l conţin pe x şi respectiv, pe y. Prin adăugarea arcului (x, y) componentele S_x şi S_y sunt înlocuite de componenta abundentă $S_x \cup S_y$. Trebuie ca şi componenta $S_x \cup S_y$ să îndeplinească Proprietăţile 9.20 şi 9.21. Fie r_x - nodul rădăcină al componentei S_x şi r_y - nodul rădăcină al componentei S_y. Presupunem că $r_x < r_y$, deci r_x va fi rădăcina componentei $S_x \cup S_y$. Dacă $e(r_y) = 0$ atunci este îndeplinită Proprietatea 9.21. Dacă $e(r_y) > 0$ atunci se măreşte cu $e(r_y)$ unităţi fluxul de-a lungul unui drum oarecare de la r_y la x şi astfel se îndeplineşte Proprietatea 9.21. Iar dacă $e(r_y) < 0$ atunci se măreşte cu $-e(r_y)$ unităţi fluxul de-a lungul unui drum de la r_x la r_y, îndeplinindu-se astfel Proprietatea 9.21.

Algoritmul Orlin al scalării intensive a capacităţii este următorul:

(1) PROGRAM SCALARE - INTENSIVĂ;
(2) BEGIN
(3) $f := 0$;
(4) $p := 0$;
(5) $e := v$;
(6) $\overline{r} = \max\{|e(x)| \mid x \in N\}$;
(7) WHILE există un nod x cu $e(x) > 0$ în $\tilde{G}(f)$ DO
(8) BEGIN
(9) IF $\max\{e(x) \mid x \in N\} \leq \overline{r}/(8n)$ THEN $\overline{r} := \max\{e(x) \mid x \in N\}$
(10) FOR fiecare arc neabundent (x, y) DO
(11) IF$f(x, y) \geq 8n\overline{r}$
(12) THEN BEGIN
(13) se consideră arcul(x, y) ca fiind un arc abundent;
(14) fie S_x componenta abundentă astfel încât $x \in S_x$;
(15) fie S_y componenta abundentă astfel încât $y \in S_y$;

```
(16)                IF S_x ≠ S_y
(17)                   THEN BEGIN
(18)                            fie r_x rădăcina lui S_x ;
(19)                            fie r_y rădăcina lui S_y ;
(20)                            PRELUCRARE (min (r_x, r_y) , max (r_x, r_y))
(21)                            componentele abundente S_x şi S_y sunt
                                 înlocuite cu S_x ∪ S_y;
(22)                          END;
(23)        WHILE în G̃(f) există un nod z astfel încât |e(z)| ≥ (n − 1)r̄/n
(24)        BEGIN
(25)           se selectează nodurile z şi w astfel încât e(z) > (n − 1)r̄/n şi
                e(w) < −r̄/n sau e(z) > r̄/n şi e(w) < −(n − 1)r̄/n;
(26)           se determină vectorul distanţelor minime d(·) de la nodul
                z la celelalte noduri în G̃(f) considerând costurile reduse ca
                fiind lungimile arcelor;
(27)           p := p − d;
(28)           se determină drumul minim D de la z la w;
(29)           se măreşte cu r̄ unităţi fluxul de-a lungul drumului D;
(30)        END;
(31)        r̄ := r̄/2;
(32)     END;
(33) END.

(1) PROCEDURA PRELUCRARE (x, y)
(2) BEGIN
(3)     IF e(y) > 0
(4)        THEN BEGIN
(5)                se determină un drum D_x de la y la x în reţeaua
                    reziduală G̃(f);
(6)                se măreşte cu e(y) unităţi fluxul de-a lungul drumului
                    D_x ;
(7)             END;
(8)        ELSE IF e(y) < 0
(9)                THEN BEGIN
(10)                      se determină un drum D_y de la x la y în G̃(f);
(11)                      se măreşte cu − e(y) unităţi fluxul de-a lungul
                           drumului D_y;
(12)                    END;
(13) END;
```

Exemplul 9.6. Ilustrăm algoritmul Orlin al scalării intensive a capacităţii pe reţeaua cu capacităţile arcelor infinite care este reprezentată în figura 9.8.(a). După iniţializările din liniile (3) - (6) obţinem $f = 0,\ p = 0,\ \overline{r} = 2^{100}$. Reţeaua reziduală $\tilde{G}(f)$ este reprezentată în figura 9.8(b). Deoarece $f = 0$ nu există arce abundente, deci subgraful abundent are 4 componente abundente: $S_1 = \{1\},\ S_2 = \{2\},\ S_3 = \{3\},\ S_4 = \{4\}$. În reţeaua reziduală $\tilde{G}(f)$ există nodul $z = 1$ astfel încât $|e(1)| = 2^{100} \geq \frac{3}{4} \cdot 2^{100} = (n-1)\overline{r}/n$. Se selectează nodul $w = 4$, cu $-2^{100} = e(4) < -\overline{r}/n = -2^{100}/4$. Se determină vectorul distanţelor minime de la 1 la celelalte noduri în $\tilde{G}(f)$: $d = (0, 2, 4, 5)$. Se actualizează $p = (0, -2, -4, -5)$, se determină drumul minim $D = (1, 3, 4)$ şi se măreşte cu 2^{100} unităţi fluxul de-a lungul drumului D. Se actualizează $\overline{r} := 2^{99}$. Reţeaua reziduală astfel obţinută este reprezentată în figura 9.8(c). În reţeaua reziduală mai există un nod cu exces: nodul 2. Se efectuează $\overline{r} := 2^{90}$. Avem $f(1,3) = 2^{100} \geq 8n\overline{r}$, deci arcul $(1,3)$ este abundent. Deoarece $e(3) < 0$, se măreşte cu 2^{90} unităţi fluxul pe arcul $(1,3)$, se înlocuiesc componentele abundente S_1 şi S_3 cu $S_{13} = \{1, 3\}$. De asemenea, $f(3,4) = 2^{100} \geq 8n\overline{r}$, deci arcul $(3,4)$ este abundent. Deoarece $e(4) = 0$, se înlocuiesc componentele abundente S_{13} şi S_4 cu $S_{134} = \{1, 3, 4\}$. Reţeaua reziduală este reprezentată în figura 9.8(d). În reţeaua reziduală $\tilde{G}(f)$ există nodul $z = 2$ astfel încât $|e(2)| = 2^{90} \geq \frac{3}{4} \cdot 2^{90} = (n-1)\overline{r}/n$. Se selectează nodul $w = 1$, cu $e(1) = -2^{90} < -2^{90}/4 = -\overline{r}/n$. Se determină vectorul distanţelor minime de la nodul 2 la celelalte noduri în $\tilde{G}(f)$: $d = (1, 0, 1, 1)$. Se actualizează potenţialele: $p = (-1, -2, -5, -6)$, se determină drumul minim $D = (2, 3, 1)$ şi se măreşte cu 2^{90} unităţi fluxul pe drumul D. Reţeaua reziduală astfel obţinută este reprezentată în figura 9.8(e). Deoarece în $\tilde{G}(f)$ nu mai există nici un nod cu exces algoritmul se termină, iar fluxul de cost minim este ilustrat în figura 9.8(f).

Fig.9.8.

Teorema 9.22. *Algoritmul Orlin al scalării intensive a capacităţii determină corect un flux de cost minim într-o reţea cu capacităţile arcelor infinite.*
Demonstraţie. Pentru demonstraţie se recomandă cititorului monografiile precizate la bibliografie. ■

Teorema 9.23. *Algoritmul Orlin al scalării intensive a capacităţii determină un flux de cost minim într-o reţea cu capacităţile arcelor infinite după $O(n \log n)$ faze de scalare şi efectuează $O(n \log n)$ măriri de flux de-a lungul unor drumuri minime. Complexitatea algoritmului Orlin al scalării intensive a capacităţii pentru determinarea unui flux de cost minim într-o reţea cu capacităţile arcelor infinite este $O(n \log nD(n,m))$, unde $D(n,m)$ este complexitatea algoritmului de determinare a drumului minim într-o reţea cu n noduri, m arce şi cu lungimile arcelor pozitive.*

Demonstraţie. Pentru demonstraţie se recomandă cititorului monografiile precizate la bibliografie. ■

Teorema 9.24. *Complexitatea algoritmului Orlin al scalării intensive a capacităţii pentru determinarea unui flux de cost minim într-o reţea oarecare este* $O(m \log n(m + n \log n))$.
Demonstraţie. Mai întâi transformăm reţeaua oarecare într-o reţea cu capacităţile arcelor infinite (a se vedea Exemplul 9.3.). Reţeaua transformată va avea $n' = n + m$ noduri şi $m' = 2m$ arce. Rezultă că algoritmul Orlin al scalării intensive a capacităţii va efectua $O(n' \log n') = O(m \log n)$ faze de scalare şi $O(n' \log n') = O(m \log n)$ măriri de flux de-a lungul unor drumuri minime. Deoarece putem determina un drum minim în reţeaua transformată care este o reţea bipartită, în $O(m + n \log n)$, rezultă că algoritmul Orlin al scalării intensive a capacităţii pentru determinarea unui flux de cost minim într-o reţea oarecare are complexitatea $O(m \log n(m + n \log n))$. ■

9.7 Aplicaţii şi comentarii bibliografice

9.7.1 Încărcarea optimă a unui avion

O companie mică are un avion care poate transporta cel mult p pasageri. La un zbor se vizitează oraşele $1, 2, 3, ..., n$ în această ordine. Avionul poate lua şi lăsa pasageri în orice oraş. Fie v_{xy} numărul de pasageri care doresc să meargă de la oraşul x la oraşul y şi fie b_{xy} costul biletului unui pasager de la oraşul x la oraşul y. Compania doreşte să determine numărul de pasageri pe care ar trebui să-i transporte de la diferite oraşe de plecare la diferite oraşe de sosire astfel încât încasările provenite din vânzarea biletelor să fie maxime, dar să nu se depăşească niciodată capacitatea avionului. În figura 9.9 este prezentată o problemă de încărcare optimă a unui avion care trece prin oraşele 1,2,3 şi 4, formulată ca o problemă de flux de cost minim.

Fig.9.9.

Pentru cazul general al unui avion care trece prin oraşele $1, 2, \ldots, n$ vom construi reţeaua $G = (N, A, b, c)$ în modul următor $N = N_1 \cup N_2,\ N_1 = \{x \mid 1 \leq x \leq n\},\ N_2 = \{x - y \mid 1 \leq x < y \leq n\},\ A = A_1 \cup A_2 \cup A_3,\ A_1 = \{(x, x+1) \mid 1 \leq x \leq n-1\},\ A_2 = \{(x-y, x) \mid x-y \in N_2\},\ A_3 = \{(x-y, y) \mid x-y \in N_2\},\ b(x, x+1) = 0,\ c(x, x+1) = p,\ (x, x+1) \in A_1,\ b(x-y, x) = b_{xy},\ c(x-y, x) = \infty,\ (x-y, x) \in A_2,\ b(x-y, y) = 0,\ c(x-y, y) = \infty,\ (x-y, y) \in A_3,\ v(1) = 0,\ v(x) = -\sum_{z=1}^{x-1} v_{zx},\ z \in N_1 \setminus \{1\}, v(x-y) = v_{xy},\ x-y \in N_2$. Vom arăta că există o corespondenţă biunivocă între o încărcare oarecare a avionului şi un flux admisibil din reţeaua G. Fie f_{xy} numărul de pasageri care călătoresc cu avionul de la oraşul x la oraşul y. Efectuăm atribuirile $f(x-y, x) = f_{xy},\ f(x-y, y) = v_{xy} - f_{xy}$.

Fluxurile de pe arcele $(x, x+1) \in A_1$ pot fi determinate folosind constrângerile de conservare a fluxului. Evident, $f(x, x+1)$ va fi egal cu numărul de pasageri care se găsesc în avion când acesta zboară între oraşul x şi oraşul $x+1$. Deoarece $c(x, x+1) = p,\ (x, x+1) \in A_1$, rezultă că nu se depăşeşte niciodată capacitatea avionului. Deci, un flux admisibil în reţeaua G corespunde unei încărcări oarecare a avionului şi reciproc. Mai mult, costul fluxului este egal cu minus suma încasată din vânzarea biletelor. Deci, o încărcare optimă a avionului corespunde unui flux de cost minim în reţeaua G şi reciproc.

9.7.2 Planificarea lucrărilor care se execută cu întârziere

În unele probleme de planificare, lucrările nu au timpi ficşi de terminare, dar întârzierea execuţiei lor atrage după sine penalizări. O astfel de problemă de planificare a lucrărilor este următoarea: Sunt q maşini identice şi p lucrări. Fiecare lucrare poate fi executată de oricare dintre maşini. Fiecare lucrare j are un timp necesar prelucrării $\alpha(j)$ care nu depinde nici de maşina care o execută, nici de lucrările care sunt efectuate înaintea ei sau după ea. Fiecare lucrare j are o penalizare de întârziere $b_j(\tau)$, care este o funcţie monoton crescătoare de τ, timpul de terminare a execuţiei lucrării. Problema constă în a determina o planificare a lucrărilor astfel încât suma penalizărilor de întârziere $\sum_{j=1}^{p} b_j(\tau_j)$ să fie minimă, unde τ_j este timpul de terminare a lucrării $j,\ j = 1, \ldots, p$. Această problemă este dificilă în cazul în care lucrările au timpii necesari prelucrării lor diferiţi, însă în cazul particular în care $\alpha_j = \alpha,\ j = 1, \ldots, p$ putem să o transformăm într-o problemă de flux de cost minim.

Deoarece penalizările de întârziere sunt funcţii monoton crescătoare de timp, rezultă că în orice planificare optimă maşinile vor executa lucrările una imediat după alta astfel încât maşinile să nu aibă timpi nefolosiţi. Deci în orice planificare optimă, timpii de terminare a lucrărilor sunt de forma $k_\alpha,\ k \in \mathcal{N}$.

Fără a restrânge generalitatea, putem presupune că $r = \frac{p}{q}$ este un număr întreg (pentru că în caz contrar am putea adăuga lucrări fictive astfel încât $\frac{p}{q}$ să fie un număr întreg). Deci, fiecare maşină va executa r lucrări. Construim reţeaua $G = (N, A, b, c)$ care este reprezentată în figura 9.10 în modul următor $N = N_1 \cup N_2,\ N_1 = \{1, 2, \ldots, p\},\ N_2 = \{\overline{1}, \overline{2}, \ldots, \overline{r}\},\ A = \{(j, \overline{i}) \mid j \in N_1,\ i \in N_2\},\ b(j, \overline{i}) = b_j(i\alpha)$ (pentru că dacă lucrarea j este a i-a lucrare executată de o maşină atunci penalizarea de întârziere este $b_j(i\alpha)$), $c(j, \overline{i}) = 1,\ (j, \overline{i}) \in A,\ v(j) = 1,\ j \in N_1,\ v(\overline{i}) = -q,\ \overline{i} \in N_2\}$. Fluxul de pe arcul $(j, \overline{i})$ este 1 dacă lucrarea j este a i-a lucrare executată de o maşină şi 0 în caz contrar. Rezultă că o planificare corectă a lucrărilor corespunde unui flux admisibil din reţeaua G şi reciproc, iar suma penalizărilor de întârziere este egală cu costul fluxului. Deci, un flux de cost minim va furniza o planificare optimă a lucrărilor.

Fig.9.10.

9.7.3 Comentarii bibliografice

În acest capitol am prezentat şase algoritmi polinomiali pentru rezolvarea problemei fluxului de cost minim. Majoritatea algoritmilor unui flux de cost minim folosesc tehnica scalării. Edmons şi Karp (1972) au introdus tehnica scalării, obţinând astfel un algoritm slab polinomial pentru rezolvarea problemei fluxului de cost minim. Orlin (1988) a descris algoritmul de scalare a capacităţii, care este o variantă a algoritmului propus de Edmons şi Karp.

Goldberg şi Tarjan (1987) au descris algoritmul de scalare a costului folosind conceptul de $\epsilon-$ optimalitate introdus independent de Bertsekas (1979) şi Tardos(1985). Ahuja, Goldberg, Orlin şi Tarjan(1992) au obţinut algoritmul scalării duble, combinând scalarea capacităţii cu scalarea costului.

Sokkalingam, Ahuja şi Orlin au prezentat un algoritm polinomial care măreşte fluxul de-a lungul circuitului de cost negativ cu capacitatea reziduală suficient de mare şi o implementare tare polinomială a acestuia.

Goldberg, Tardos şi Tarjan (1989) au obţinut algoritmul de scalare repetată a capacităţii, care se bazează pe "fixarea" potenţialelor nodurilor. Algoritmul scalării intensive a capacităţii se datorează lui Orlin (1988).

Capitolul 10

Fluxuri dinamice

10.1 Problema fluxului dinamic

În precedentele cinci capitole au fost prezentate fluxuri care se supun anumitor cerinţe dictate de capacităţile de arc sau/şi costurile de arc. În acest capitol se consideră în plus o altă cerinţă, numită *timp de traversare* a arcului şi vom studia fluxurile care trebuie să se transfere de la nodul sursă s la nodul stoc t în cadrul unei cantităţi de timp date.

Fie digraful $G = (N, A,)$, $\mathcal{N}$ mulţimea numerelor naturale şi $P = \{0, 1, \ldots, q\}$ mulţimea perioadelor de timp. Fie funcţia timp $h : A \to \mathcal{N}$ şi funcţia capacitate $c : A \times P \to \mathcal{N}$, unde $h(x, y)$ reprezintă timpul de traversare a arcului şi $c(x, y; i)$ capacitatea arcului la timpul i pentru $(x, y) \in A$, $i \in P$.

Problema fluxului dinamic maximal pentru q perioade de timp constă în determinarea unui flux $\hat{f}, \hat{f}: A \times P \to \mathcal{N}$, care satisface următoarele condiţii:

$$\sum_{i=0}^{q} \left(\sum_{y} \hat{f}(s, y; i) - \sum_{y} f(y, s; j) \right) = \hat{v}(P) \tag{10.1.a}$$

$$\sum_{y} \hat{f}(x, y; i) - \sum_{y} \hat{f}(y, x; j) = 0,\ x \neq s, t;\ i \in P \tag{10.1.b}$$

$$\sum_{i=0}^{q} \left(\sum_{y} \hat{f}(t, y; i) - \sum_{y} \hat{f}(y, t; i - j) \right) = -\hat{v}(P) \tag{10.1.c}$$

$$0 \leq \hat{f}(x, y; i) \leq c(x, y; i),\ (x, y) \in A,\ i \in P \tag{10.2}$$

$$\max \hat{v}(P), \tag{10.3}$$

unde $j = i - h(y,x)$, $\hat{v}(P) = \sum_{i=0}^{q} \hat{v}_i$, $\hat{v}_i$ este valoarea fluxului la timpul i.

Deci, un flux dinamic $\hat{f}$ de la nodul sursă s la nodul stoc t este orice flux de la s la t pentru care cel mult $c(x,y;\, i)$ unităţi de flux intră în arcul (x,y) la startul perioadei de timp i pentru toate arcele (x,y) şi toţi i. Notăm că într-un flux dinamic, unităţile de flux pot pleca de la nodul sursă s la timpul $0, 1, \ldots$. Un flux dinamic maxim $\hat{f}^{*}$ de la s la t este orice flux dinamic de la s la t în care numărul maxim posibil de unităţi de flux ajung la stocul t în timpul primelor q perioade de timp.

Evident, problema determinării unui flux dinamic maxim $\hat{f}$ este mult mai complexă decât problema determinării unui flux static maxim f deoarece problema fluxului dinamic cere să urmărim la fiecare timp să fie verificate constrângerile (10.2).

Problema fluxului dinamic pentru q perioade de timp în reţeaua dinamică $G = (N, A, c, h, q)$ este echivalentă cu problema fluxului static în reţeaua statică $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{c}, q)$ unde:

$$\tilde{N} = \{x_i \,|\, x \in N,\ i \in P\}$$
$$\tilde{A} = \{(x_i, y_j) \,|\, (x,y) \in A;\ i, j \in P,\ j = i + h(x,y)\},$$
$$\tilde{c}(x_i, y_j) = c(x,y;\, i),\ (x,y) \in A;\ i, j \in P,\ j = i + h(x,y).$$

Nodurile x_i din $\tilde{N}$ se obţin prin duplicarea fiecărui nod x din N o dată pentru fiecare perioadă de timp i, $i = 0, 1, \ldots, q$ şi (x_i, y_j) este un arc din $\tilde{A}$ numai dacă timpul de traversare al arcului (x,y) din A este $h(x,y) = j - i$.

Exemplul 10.1. Fie reţeaua dinamică $G = (N, A, c, h, q)$ din figura 10.1 cu $q = 6$, $c(x,y;\, i) = 1$, $(x,y) \in A$, $i \in P$, fiecare timp de traversare $h(x,y)$ este specificat pe arcul (x,y), $(x,y) \in A$, $s = 1$, $t = 4$.

Fig.10.1

Reţeaua statică $\tilde{G} = \left(\tilde{N}, \tilde{A}, \tilde{c}, 6\right)$ este reprezentată în figura 10.2. Evident că există noduri x_i şi arce (x_i, y_j) din reţeaua statică $\tilde{G}$ care nu aparţin nici unui drum de la oricare nod sursă s_i la oricare nod stoc t_j. Se poate construi reţeaua statică $\tilde{G}' = \left(\tilde{N}', \tilde{A}', \tilde{c}', q\right)$ care elimină acest neajuns. Această reţea se construieşte în modul următor. Fie d_x distanţa de la nodul sursă s la nodul x şi $\overline{d}_x$ distanţa de la nodul x la nodul stoc t în raport cu funcţia timp h. Fie mulţimile

$$P_x = \{i \,|\, i \in P,\ d_x \le i \le q - \overline{d}_x\},\ x \in N,$$
$$P_{xy} = \left\{i \,|\, i \in P,\ d_x \le i \le q - \left(h(x,y) + \overline{d}_y\right)\right\},\ (x,y) \in A.$$

Construim N' şi A' în modul următor:

$N' = \{x \,|\, x \in N,\ P_x \neq \emptyset\}$,
$A' = \{(x,y) \,|\, (x,y) \in A,\ P_{xy} \neq \emptyset.\}$

Fie reţeaua dinamică $G' = (N', A', h', c', q,)$ unde funcţiile h' şi c' sunt restricţiile funcţiilor h şi respectiv c la A'. Evident că un flux dinamic din reţeaua $G = (N, A, c, h, q)$ este echivalent cu un flux dinamic din reţeaua $G' = (N', A', h', c', q)$.

Fig.10.2

Reţeaua statică $\tilde{G}' = \left(\tilde{N}', \tilde{A}', \tilde{c}', q\right)$ este construită în modul următor:

$\tilde{N}' = \{x_i \,|\, x \in N',\ i \in P_x\}$,
$\tilde{A}' = \{(x_i, y_j) \,|\, (x,y) \in A',\ i \in P_{xy},\ j = i + h(x,y)\}$,
$\tilde{c}'(x_i, y_j) = c'(x, y;\, i),\ (x,y) \in A',\ i \in P_{xy},\ j = i + h(x,y)$.

Reţeaua G' este, în general, o subreţea a reţelei G şi reţeaua $\tilde{G}'$ este întotdeauna o subreţea a reţelei $\tilde{G}$.

Exemplul 10.2. Fie reţeaua dinamică $G = (N, A, c, h, q)$ reprezentată în figura 10.1. Se obţine: $P_1 = \{0,1,2,3\}$, $P_2 = \{1,2,3,4\}$, $P_3 = \{2,3,4,5\}$, $P_4 = \{3,4,5,6\}$, $P_{12} = \{0,1,2,3,4\}$, $P_{13} = \{0,1,2\}$, $P_{23} = \{1,2,3,4\}$, $P_{24} = \{1,2,3\}$, $P_{34} = \{2,3,4,5\}$. Deoarece $P_x \neq \emptyset, x \in N$, $P_{xy} \neq \emptyset, (x,y) \in A$ rezultă că $G' = G$. Reţeaua statică $\tilde{G}'$ este reprezentată în figura 10.3. Este evident că reţeaua $\tilde{G}'$ este o subreţea a reţelei $\tilde{G}$ reprezentată în figura 10.2.

Fig.10.3

În tabelul 10.1 se prezintă echivalenţa dintre un flux dinamic din reţeaua dinamică $G = (N, A, c, h, q)$ reprezentată în figura 10.1 şi un flux static din reţeaua statică $\tilde{G} = \left(\tilde{N}, \tilde{A}, \tilde{c}, 6\right)$ reprezentată în figura 10.2 sau din reţeaua statică $\tilde{G}' = \left(\tilde{N}', \tilde{A}', \tilde{c}', 6\right)$ reprezentată în figura 10.3.

Flux dinamic			Flux static	
Drumul D	i	$g(D)$	Drumul D^*	$g(D^*)$
1,2,3,4	0	1	$1_0, 2_1, 3_2, 4_3$	1
1,2,3,4	1	1	$1_1, 2_2, 3_3, 4_4$	1
1,2,4	2	1	$1_2, 2_3, 4_6$	1

Tabelul 10.1

Deci un flux dinamic maxim $\hat{f}$ pentru q perioade de timp în reţeaua dinamică $G = (N, A, c, h, q)$ se poate determina aplicând un algoritm de flux maxim, prezentat în capitolul 5 sau 6, pe reţeaua statică $\tilde{G} = \left(\tilde{N}, \tilde{A}, \tilde{c}, q\right)$ sau $\tilde{G} = \left(\tilde{N}', \tilde{A}', \tilde{c}', q\right)$.

10.2 Fluxuri dinamice staţionare

Dacă numărul de perioade q este foarte mare, atunci numărul nodurilor din $\tilde{N}$ şi numărul arcelor din $\tilde{A}$ devin foarte mari. Acest lucru este valabil chiar şi pentru reţeaua $\tilde{G}' = \left(\tilde{N}', \tilde{A}', \tilde{c}', q\right)$. În acest caz, determinarea unui flux static maxim în reţeaua $\tilde{G}$ sau $\tilde{G}'$ nu mai este avantajos.

Se spune că un flux dinamic este *staţionar* dacă funcţiile capacitate c şi timp h ale reţelei dinamice $G = (N, A, c, h, q)$ sunt constante în timp ($c: A \to \mathcal{N}$, $h: A \to \mathcal{N}$). În cazul staţionar, problema fluxului dinamic se poate rezolva eficient fără a utiliza reţeaua statică $\tilde{G}$ sau $\tilde{G}'$. În algoritmul care determină un flux dinamic staţionar maxim se utilizează ca procedură algoritmul fluxului de timp minim. Acest algoritm este unul din algoritmii prezentaţi în capitolul 8 sau capitolul 9 în care funcţia cost b se înlocuieşte cu funcţia timp h. Algoritmul fluxului dinamic staţionar maxim este următorul:

(1) PROGRAM FDSM;
(2) BEGIN
(3) PROCEDURA FTM (G^*, f^*);
(4) PROCEDURA DF $(G^*, f^*, \mathcal{D}^*, R^*)$;
(5) PROCEDURA RF $\left(\mathcal{D}^*, R^*, \tilde{G}^*\right)$;
(6) END.

PROCEDURA FTM determină un flux static de timp minim f^* în reţeaua dinamică G^*. Dacă $G^* = G$ atunci se determină un flux static de timp minim $f^* = f$ cu $d_t \leq q$. Dacă $G^* = G'$ atunci se determină un flux static maxim de timp minim $f^* = f'$.

PROCEDURA DF descompune fluxul f^* în fluxuri de drum; fie $\mathcal{D}^* = \{D_1^*, \ldots, D_r^*\}$ mulţimea drumurilor de la nodul sursă s la nodul stoc t care au flux nenul şi $R^* = \{r(D_1^*), \ldots, r(D_r^*)\}$ mulţimea valorilor fluxurilor drum.

PROCEDURA RF repetă fluxurile drum $g(D_k^*)$ în reţeaua $\tilde{G}^* = \tilde{G}$ respectiv $\tilde{G}^* = \tilde{G}'$ pentru perioadele de timp $i = 0, 1 \ldots, q - \tau(D_k^*)$, unde $\tau(D_k^*)$ este timpul de parcurs al drumului D_k^*, $k = 1, \ldots, r$. Datorită acestei repetări, fluxul dinamic determinat cu algoritmul FDSM se numeşte *flux dinamic repetat în timp.*

Exemplul 10.3 Fie reţeaua dinamică $G = (N, A, c, h, q)$ reprezentată în figura 10.1. În exemplul 10.2 s-a precizat că $G' = G$. Aplicând PROCEDURA FTM se obţine fluxul f' unde $f'(x, y)$ este reprezentat pe arcul (x, y), $(x, y) \in A$ din figura 10.4.

Fig.10.4

Aplicând PROCEDURA DF se obţine $\mathcal{D}' = \{D'_1 = (1,2,4),\ D'_2 = (1,3,4)\}$, $\mathcal{G}' = \{g(D'_1) = 1,\ g(D'_2) = 1\}$, $\tau(D'_1) = 4, \tau(D'_2) = 4$. PROCEDURA RF trimite de la $s = 1$ câte o unitate de flux de-a lungul drumurilor D'_1 şi D'_2 la timpii 0,1,2 şi ajung la $t = 4$ la timpii 4,5,6. Valoarea fluxului dinamic maxim este $v(P) = v(4) + v(5) + v(6) = 2 + 2 + 2 = 6$. Acest flux este reprezentat pe reţeaua statică $\tilde{G}'$ din figura 10.5.

Fig.10.5

Fie v^* valoarea fluxului static de timp minim f^*. Evident că $v^* = v = v'$.

Teorema 10.1 (Teorema de corectitudine). *Fluxul static de timp minim f^* generează un flux dinamic maxim repetat în timp pentru q perioade de timp şi are valoarea:*

$$v(P) = (q+1)v^* - \sum_A h(x,y) f^*(x,y). \tag{10.4}$$

Demonstraţie. Algoritmul trimite unităţi de flux de la nodul sursă s la nodul stoc t de-a lungul drumurilor din $\mathcal{D}^*$. Deci sunt verificate condiţiile 10.1 şi 10.2. De asemenea, toate unităţile de flux pleacă din nodul sursă s şi aceste unităţi ajung în nodul sursă t înainte de q perioade de timp. Deci algoritmul generează un flux dinamic.

Pentru a demonstra că fluxul dinamic generat de algoritm este un flux dinamic maxim, se consideră echivalentul lui în reţeaua statică $\tilde{G} = \left(\tilde{N}, \tilde{A}, \tilde{c}, q\right)$. În funcţie de algoritmul care se utilizează în PROCEDURA FTM se defineşte o tăietură în $\tilde{G}$ care separă sursele $s_0, \ldots, s_q$ de $t_0, \ldots, t_q$ şi se arată că aceasta este o tăietură minimă. Rezultă că fluxul static $\tilde{f}$ din $\tilde{G}$ este un flux maxim şi deci echivalentul lui în reţeaua dinamică G este un flux dinamic maxim $\hat{f}$ pentru q perioade de timp. Evident, că algoritmul converge într-un număr finit de iteraţii deoarece fiecare procedură converge într-un număr finit de iteraţii.

Din PROCEDURA RF rezultă:

$$v(P) = \sum_{k=1}^{r} \left(q + 1 - \tau(D_k^*)\right) g(D_k^*) = (q+1)v^* - \sum_A h(x,y) f^*(x,y). \qquad \blacksquare$$

Pentru o demonstraţie în detaliu se poate consulta bibliografia indicată.

Teorema 10.2 (Teorema de complexitate a algoritmului). *Algoritmul FDSM are complexitatea procedurii FTM.*

Demonstraţie. Este evident faptul că algoritmul FDSM are complexitatea procedurii FTM. Complexitatea acestei proceduri depinde de algoritmul fluxului de timp minim utilizat. $\blacksquare$

10.3 Fluxuri dinamice lexicografice

În acest paragraf considerăm cazul staţionar. Deci funcţia capacitate c şi funcţia timp h ale reţelei dinamice $G = (N, A, c, h, q)$ sunt constante în timp ($c : A \to \mathcal{N}$, $h : A \to \mathcal{N}$).

O clasificare a fluxurilor dinamice poate fi făcută în raport cu plecarea de la nodul sursă s şi sosirea la nodul stoc t. În tabelul 10.2 sunt prezentate categoriile de fluxuri clasificate conform acestui criteriu.

plecarea\ sosirea	cea mai devreme	oarecare	cea mai târzie
cea mai devreme	da	da	da
oarecare	da	da	da
cea mai târzie	da	da	da

Tabelul 10.2

Dacă pentru plecare sau/şi sosire nu se specifică nimic atunci înseamnă că ele sunt oarecare. Astfel un flux dinamic cu sosirea cea mai devreme şi plecarea oarecare îl vom identifica prin flux dinamic cu sosirea cea mai devreme. În paragraful 2 au fost prezentate fluxurile dinamice maxime cu plecarea oarecare şi sosirea oarecare.

Fie $\overline{P}_k = \{0, 1, \ldots, k\}$ mulţimea perioadelor de timp cu k perioade, $k = 1, \ldots, q$. Evident că $\overline{P}_q = P$. Fie f_k fluxul de timp minim, $d_t \leq k$, cu valoarea v_k în reţeaua dinamică $G_k = (N, A, c, h, k)$, f'_k fluxul maxim de timp minim cu valoarea v'_k în reţeaua dinamică $G'_k = (N', A', c', h', k)$. De asemenea, fie $\hat{f}_k$ fluxul dinamic maxim ($\tilde{f}_k$ fluxul static maxim în reţeaua statică $\tilde{G}_k$ generat de f_k) şi $\overset{\wedge}{f'}_k$ fluxul dinamic maxim ($\tilde{f'}_k$ fluxul static maxim din reţeaua statică $\tilde{G}'_k$ generat de f'_k) pentru k perioade de timp. Conform precizărilor anterioare avem $v^*_k = v_k = v'_k$ şi $\hat{v}(P_k)$ este:

$$\hat{v}(\overline{P}_k) = (k+1)v^*_k - \sum_A h(x,y) f^*_k(x,y).$$

Din această relaţie rezultă că $\hat{v}(\overline{P}_k) - \hat{v}(\overline{P}_{k-1}) = v^*_k$. Evident că $v^*_k \geq v^*_{k-1}$ şi $v^*_k = v^*_{k-1}$ dacă $f^*_k = f^*_{k-1}$. În consecinţă când k creşte valoarea fluxului dinamic maxim creşte cu cel puţin v^*_{k-1} unităţi.

Un flux cu sosirea cea mai devreme pentru q perioade de timp este un flux dinamic maxim pentru q perioade de timp cu proprietatea că $v(\overline{P}_k)$ unităţi de flux sosesc la stocul t, pentru $k = 0, 1 \ldots, q$. Un astfel de flux este cunoscut în literatura de specialitate şi ca flux dinamic maxim universal deoarece acest flux este maxim pentru oricare $\overline{P}_k$, $k = 0, 1, \ldots, q$.

Teorema 10.3 (Teorema de existenţă). *În reţeaua dinamică $G = (N, A, c, h, q)$ există un flux dinamic maxim cu sosirea cea mai devreme pentru q perioade de timp.*

Demonstraţie. Se utilizează un algoritm de flux maxim pentru determinarea secvenţei de fluxuri statice maxime $\tilde{f}_0, \ldots, \tilde{f}_q$ în reţeaua statică $\tilde{G}$. Fluxul static maxim $\tilde{f}_k$ este de la nodurile sursă $s_0, s_1, \ldots, s_k$ la nodurile stoc $t_0, \ldots, t_k$ şi se determină plecând cu fluxul iniţial $\tilde{f}_{k-1}$, $k = 1, \ldots, q$. Este evident că după determinarea lui $\tilde{f}_k$ valorile $\tilde{v}_i$ la nodurile stoc t_i, $i = 0, \ldots, k-1$ rămân nemodificate dar, în general, se modifică repartizarea fluxului pe alte drumuri de la nodurile sursă s_i, $i = 0, \ldots, k-1$ la nodurile stoc t_i, $i = 0, \ldots, k-1$. Fluxul $\tilde{f}_k$ din reţeaua statică $\tilde{G}$ corespunde unui flux dinamic maxim $\hat{f}_k$ din reţeaua dinamică G pentru k perioade de timp cu $\hat{v}(\overline{P}_k) = \hat{v}(\overline{P}_{k-1}) + \tilde{v}_k$, $k = 1, \ldots, q$. Rezultă că $\hat{f} = \hat{f}_q$ este un flux dinamic maxim cu sosirea cea mai devreme pentru q perioade de timp. ■

Demonstraţia Teoremei 10.3 nu arată numai faptul că un flux dinamic maxim cu sosirea cea mai devreme există întotdeauna pentru orice reţea dinamică $G = (N, A, c, h, q)$ şi pentru orice $q = 0, 1, \ldots$, dar de asemenea poate reprezenta o procedură pentru determinarea unui astfel de flux. Dacă m, n, q au valori foarte mari atunci această procedură nu este eficientă. În cazul staţionar un flux dinamic maxim cu sosirea cea mai devreme pentru q perioade de timp se poate determina ca un flux dinamic repetat în timp. În continuare se va prezenta un algoritm pentru determinarea unui flux dinamic maxim cu sosirea cea mai devreme.

În capitolele 8 şi 9 prin $p(x)$ am notat variabila duală ataşată nodului x, pentru toate nodurile $x \in N$. Valoarea lui $p(x)$ se numeşte *potenţialul nodului* x. Fie $h_p(x, y) = h(x, y) + p(x) - p(y)$ timpul redus al arcului (x, y), pentru fiecare arc $(x, y) \in A$.

În continuare prezentăm o variantă a algoritmului Wilkinson pentru determinarea unui flux dinamic maxim cu sosirea cea mai devreme. Acest algoritm se va numi *algoritmul VW*. Presupunem că dacă $(x, y) \in A$ atunci $(y, x) \notin A$. Această condiţie nu micşorează generalitatea deoarece se poate introduce un nod z pe arcul (y, x) dacă reţeaua conţine de asemenea arcul (x, y). Algoritmul VW utilizează conceptul de reţea reziduală. Reamintim că reţeaua reziduală $G(f)$ corespunzătoare fluxului static f este definită în modul următor. Înlocuim fiecare arc $(x, y) \in A$ prin două arce (x, y) şi (y, x). Arcul (x, y) are timpul de tranzit $h(x, y)$ şi capacitatea reziduală $r(x, y) = c(x, y) - f(x, y)$ şi arcul (y, x) are timpul de tranzit $h(y, x) = -h(x, y)$ şi capacitatea reziduală $r(y, x) = f(x, y)$. Reţeaua reziduală $G(f)$ constă din nodurile N şi toate arcele (x, y) şi (y, x) cu $r(x, y) > 0$ şi $r(y, x) > 0$. Algoritmul VW este următorul:

(1) PROGRAMUL VW;
(2) BEGIN
(3) $f := 0$; $\hat{f} := 0$; $p := 0$;
(4) se determină reţeaua reziduală $G(f)$;
(5) WHILE $p(t) \leq q$ DO
(6) BEGIN
(7) se determină distanţele d_x în $G(f)$ în raport cu h_p;
(8) se determină drumul minim D de la s la t;
(9) $p := p + d$;
(10) IF $p(t) \leq q$
(11) THEN BEGIN
(12) $r(D) := \min\{r(x,y) \mid (x,y) \in D\}$;
(13) MĂRIRE FS$(G, f, D, r(D))$;
(14) MĂRIRE FD$(G, \hat{f}, D, r(D))$;
(15) se actualizează $G(f)$;
(16) END;
(17) END;
(18) END.

(1) PROCEDURA MĂRIRE FS$(G, f, D, r(D))$;
(2) BEGIN
(3) fie L lanţul din G corespunzător drumului D din $G(f)$;
(4) FOR $(x,y) \in L$ DO
(5) IF $(x,y) \in L^+$
(6) THEN $f(x,y) := f(x,y) + r(D)$
(7) ELSE $f(x,y) := f(x,y) - r(D)$;
(8) END;

(1) PROCEDURA MĂRIRE FD$(G, \hat{f}, D, r(D))$;
(2) BEGIN
(3) fie L lanţul din G corespunzător drumului D din $G(f)$;
(4) FOR $(x,y) \in L$ DO
(5) IF $(x,y) \in L^+$
(6) THEN $\hat{f}(x,y;i) := \hat{f}(x,y;i) + r(D)$
pentru $i = p(x), \ldots, p(x) + q - p(t)$
(7) ELSE $\hat{f}(x,y;i) = \hat{f}(x,y;i) - r(D)$
pentru $i = p(x), \ldots, p(x) + q - p(t)$;
(8) END;

Exemplul 10.4 Fie reţeaua dinamică $G = (N, A, c, h, q)$ reprezentată în figura 10.1. Să determinăm un flux dinamic maxim cu sosirea cea mai devreme utilizând algoritmul VW.

Iniţial $f = 0$, $\hat{f} = 0$, $p = 0$, $G(0) = G$.

Iteraţia 1. $h_p = (h_p(1,2),\ h_p(1,3),\ h_p(2,3), h_p(2,4),\ h_p(3,4)) = (1,3,1,3,1)$, $d = (0,1,2,3)$, $D = (1,2,3,4)$, $p = (0,1,2,3)$, $r(D) = 1$.
Fluxul static f obţinut cu PROCEDURA MĂRIRE FS este reprezentat în figura 10.6.

Fig.10.6

Fluxul dinamic $\hat{f}$ obţinut cu PROCEDURA MĂRIRE FD este reprezentat prin fluxul static $\tilde{f}$ în figura 10.7.

Fig.10.7

Reţeaua reziduală $G(f)$ este reprezentată în figura 10.8.

Fig.10.8

Iteraţia 2. $h_p = (h_p(1,3),\ h_p(2,1),\ h_p(2,4), h_p(3,2),\ h_p(4,3)) = (1,0,1,0,0)$.
$d = (0,1,1,2)$, $D = (1,3,2,4)$, $p = (0,2,3,5)$, $r(D) = 1$,
Fluxul static f obţinut cu PROCEDURA MĂRIRE FS este reprezentat în figura 10.9.

Fluxul dinamic $\hat{f}$ obţinut cu PROCEDURA MĂRIRE FD este reprezentat prin fluxul static $\tilde{f}$ în figura 10.10.

Reţeaua reziduală $G(f)$ este reprezentată în figura 10.11.

Iteraţia 3. $h_p = (h_p(2,1),\ h_p(2,3),\ h_p(3,1), h_p(4,2),\ h_p(4,3)) = (1,0,0,0,1)$, $d = (0,\infty,\infty,\infty)$, D nu există , $p = (0,\infty,\infty,\infty)$. Algoritmul se termină şi fluxul dinamic reprezentat în figura 10.10 este un flux dinamic maxim cu sosirea cea mai devreme.

Teorema 10.4 (Teorema de corectitudine). *Algoritmul VW generează un flux dinamic maxim cu sosirea cea mai devreme.*

Demonstraţie. La fel ca în demonstraţia Teoremei 10.1 se arată că algoritmul VW generează un flux dinamic maxim. Evident că acest flux este un flux repetat în timp. Din enunţul algoritmului rezultă că, mai întâi, $f := f + g_k$, fluxul g_k este de-a lungul lanţului de timp minim L_k de la nodul sursă s la nodul stoc t cu $\tau(L_k) = j$, $k = 1, \ldots, r$ şi $\hat{f} := \hat{f} + g_k(i)$, $g_k(i)$ pleacă de la s la timpul i, $i = 0, \ldots, q - j$ şi soseşte la t respectiv la timpul i, $i = j, \ldots, q$. Apoi $f = f + g'_k$, fluxul g'_k este de-a lungul lanţului cu al doilea timp minim L'_k cu $\tau(L'_k) = j'$, $k = 1, \ldots, r'$, $j' \geq j+1$ şi $\hat{f} := \hat{f} + g'_k(i)$, g'_k pleacă de la s la timpul i, $i = 0, \ldots, q - j'$ şi soseşte la t

Fig.10.9

Fig.10.10

respectiv la timpul i, $i = j', \ldots, q$. Acest procedeu continuă cât timp $\tau(L) \leq q$. Evident că nici un flux $g'_k(i)$ nu modifică valorile $\hat{v}_0, \ldots, \hat{v}_{k'}$, $k' = j' - 1$. Această afirmaţie este adevărată pentru toate clasele de lanţuri $L_k, L'_k, \ldots$. Prin urmare fluxul dinamic maxim este cu sosirea cea mai devreme. ■

Teorema 10.5 (Teorema de complexitate). *Algoritmul VW are complexitatea* $O(nmq^2\overline{c})$, $\overline{c} = \max\{c(x,y) \,|\, (x,y) \in A\}$.

Demonstraţie. Pentru fiecare $p(t)$, $0 \leq p(t) \leq q$, pot exista cel mult n drumuri D în $G(f)$. Deoarece $1 \leq r(D) \leq \overline{c}$ rezultă că algoritmul execută cel mult $nq\overline{c}$ iteraţii. Complexitatea unei iteraţii este determinată de PROCEDURA MĂRIRE FD. Această procedură are complexitatea $O(mq)$. Deci algoritmul are complexitatea $O(nmq^2\overline{c})$. ■

Fie $\overline{P}'_k = \{k, \ldots, q\}$ mulţimea ultimelor $q-k+1$ perioade de timp şi $\hat{v}(\overline{P}'_k) = \sum_{i=k}^{q} \hat{v}_k$, $k = 0, \ldots, q$. Deoarece $\overline{P}_k = \{0, \ldots, k\}$ rezultă că $\overline{P}_q = \overline{P}'_0 = P$.

Un flux cu sosirea cea mai târzie pentru q perioade de timp este un flux dinamic maxim cu proprietatea că este maxim pentru fiecare $\overline{P}'_k$, $k = 0, \ldots, q$.

Teorema 10.6 (Teorema de existenţă). *În reţeaua dinamică* $G = (N, A, c, h, q)$ *există un flux dinamic maxim cu sosirea cea mai târzie pentru* q *perioade de timp.*

Demonstraţie. Teorema se demonstrează prin construcţie în acelaşi mod ca demonstraţia Teoremei 10.3 a existenţei fluxului dinamic maxim cu sosirea cea mai devreme pentru q perioade de timp, cu deosebirea că în acest caz unităţile de flux sunt trimise mai întâi la nodul stoc t_q, apoi la nodul stoc $t_{q-1}, \ldots$. ■

Pentru determinarea unui flux dinamic maxim cu sosirea cea mai târzie pentru q perioade de timp se poate elabora un algoritm similar cu algoritmul VW cu deosebirea că mai întâi se determină fluxul g_k de-a lungul lanţului de timp maxim L_k, $j = \tau(L_k) \leq q$, $k = 1, \ldots, r$ şi se repetă în timp, apoi fluxurile g'_k de-a lungul lanţului L'_k, $\tau(L'_k) = j' \leq j - 1$ şi se repetă în timp, ...

Într-un mod similar, se pot defini un flux dinamic maxim cu plecarea cea mai devreme pentru q perioade de timp şi un flux dinamic maxim cu plecarea cea mai târzie pentru q perioade de timp. Existenţa fiecăruia din aceste fluxuri poate fi demonstrată constructiv într-un mod similar cu demonstraţia pentru fluxul cu sosirea cea mai devreme respectiv cea mai târzie în care se substituie nodurile stoc prin nodurile sursă.

Fie $G^{-1} = (N, A^{-1})$, $A^{-1} = \{(y,x) \,|\, (x,y) \in A\}$ digraful invers digrafului $G = (N, A)$ şi $G^{-1} = (N, A^{-1}, c^{-1}, h^{-1}, q)$, cu t nodul sursă şi s nodul stoc, reţeaua dinamică inversă a reţelei dinamice $G = (N, A, c, h, q)$ cu $c^{-1}(y,x) = c(x,y)$, $h^{-1}(y,x) = h(x,y)$, $(y,x) \in A^{-1}$. Evident că reţelei statice $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{c}, q)$

Fig.10.11

îi corespunde o reţea statică inversă $\tilde{G}^{-1} = (\tilde{N}, \tilde{A}^{-1}, \tilde{c}^{-1}, q)$. Analog pentru reţeaua statică $\tilde{G}' = (\tilde{N}', \tilde{A}', \tilde{c}')$. Dacă $\hat{f}$ este un flux dinamic de la nodul sursă s la nodul stoc t pentru q perioade de timp în reţeaua dinamică G, atunci $\hat{f}^{-1}$ cu $\hat{f}^{-1}(y,x) = \hat{f}(x,y)$, $(y,x) \in A^{-1}$ este un flux dinamic de la nodul sursă t la nodul stoc s pentru q perioade de timp în reţeaua dinamică inversă G^{-1}. Unităţile de flux ale fluxului dinamic $\hat{f}^{-1}$ pleacă de la nodul sursă t la timpul i, $i = q, q-1, \ldots, 0$. Deci se poate considera că unităţile fluxului dinamic $\hat{f}^{-1}$ circulă înapoi în timp.

Teorema 10.7. *Dacă $\hat{f}$ este un flux dinamic maxim pentru q perioade de timp în reţeaua dinamică G pentru care $\hat{w}_i$ unităţi de flux pleacă de la nodul sursă s în timpul perioadei i şi $\hat{v}_i$ unităţi de flux sosesc la nodul stoc t în timpul perioadei i, pentru $i = 0, \ldots, q$, atunci $\hat{f}^{-1}$ este un flux dinamic maxim pentru q perioade de timp în reţeaua dinamică inversă G^{-1} pentru care $\hat{v}_i$ unităţi de flux pleacă de la nodul sursă t în timpul perioadei $q-i$ şi $\hat{w}_i$ unităţi de flux sosesc la nodul stoc s în timpul perioadei $q-i$, pentru $i = 0, \ldots, q$.*
Demonstraţie. Afirmaţia teoremei rezultă din precizările de mai sus. ■

Din Teorema 10.7 rezultă că unui flux dinamic maxim $\hat{f}$ cu sosirea cea mai devreme pentru q perioade de timp în reţeaua dinamică G îi corespunde un flux dinamic maxim $\hat{f}^{-1}$ cu plecarea cea mai târzie pentru q perioade de timp în reţeaua dinamică inversă G^{-1}. Deci, unui flux dinamic maxim $\hat{f}^{-1}$ cu sosirea cea mai devreme pentru q perioade de timp în reţeaua dinamică inversă G^{-1} îi corespunde un flux dinamic maxim $\hat{f}$ cu plecarea cea mai târzie pentru q perioade de timp în reţeaua dinamică $\left(G^{-1}\right)^{-1} = G$.

Aceste precizări ne conduc la un algoritm pentru determinarea unui flux dinamic maxim cu plecarea cea mai târzie pentru q perioade de timp în reţeaua dinamică G. În continuare vom arăta că acest algoritm nu este necesar.

Teorema 10.8. *Dacă fluxul dinamic maxim $\hat{f}$ cu sosirea cea mai devreme pentru q perioade de timp în reţeaua dinamică G are $\hat{v}_i$ unităţi de flux care sosesc la nodul stoc t în timpul perioadei i, $i = 0, \ldots, q$, atunci fluxul dinamic maxim $\overset{\wedge}{\varphi}$ cu plecarea cea mai târzie pentru q perioade de timp în reţeaua dinamică G are $\hat{v}_i$ unităţi de flux care pleacă de la nodul sursă s în timpul perioadei $q-i$, pentru $i = 0, \ldots, q$.*

Demonstraţie. Fie perioada de timp k oarecare, $0 \leq k \leq q$. Pentru fluxul $\hat{f}$, respectiv fluxul $\overset{\wedge}{\varphi}$ avem:

$$\hat{v}\left(\overline{P}_k\right) = \sum_{i=0}^{k} \hat{v}_i, \text{ respectiv } \hat{w}\left(\overline{P}'_{q-k}\right) = \sum_{i=q-k}^{q} \hat{w}_i = \sum_{i=0}^{k} \hat{w}_{q-i}$$

Din Teorema 10.7 rezultă că $\hat{w}\left(\overline{P}'_{q-k}\right) = \hat{v}\left(\overline{P}_k\right)$. Deoarece k este oarecare obţinem că $\hat{w}_{q-i} = \hat{v}_i$ pentru $i = 0, \ldots, q$.

Teorema 10.9. *Fluxul dinamic maxim $\hat{f}$ cu sosirea cea mai devreme pentru q perioade de timp în reţeaua dinamică G construit cu algoritmul VW este un flux dinamic maxim cu plecarea cea mai târzie pentru q perioade de timp în reţeaua dinamică G.*

Demonstraţie. Fluxul dinamic maxim $\hat{f}$ cu sosirea cea mai devreme pentru q perioade de timp în reţeaua dinamică G construit cu algoritmul VW are $\hat{w}_i$ unităţi de flux care pleacă de la nodul sursă s la timpul i, $i = 0, \ldots, q$ şi $\hat{v}_i$ unităţi de flux care sosesc la nodul stoc t la timpul i, $i = 0, \ldots, q$. Evident că

$$\hat{v}(P) = \sum_{i=0}^{q} \hat{w}_i = \sum_{i=0}^{q} \hat{v}_i \ .$$

Fie două clase oarecare L'_k, $k = 1, \ldots, r'$ şi L''_k, $k = 1, \ldots, r''$ de lanţuri de mărire a fluxului determinate cu algoritmul VW care au timpul de parcurs $\tau(L'_k) = j'$, $k = 1, \ldots, r'$ şi $\tau(L''_k) = j''$, $k = 1, \ldots, r''$, $j' < j''$. Algoritmul VW trimite $r(L'_k)$, $k = 1, \ldots, r'$, respectiv $r(L''_k)$, $k = 1, \ldots, r''$, unităţi de flux de la nodul sursă s la timpul i, $i = 0, \ldots, q - j'$, respectiv $i = 0, \ldots, q - j''$ şi sosesc la nodul stoc t la timpul i, $i = j', \ldots, q$, respectiv $i = j'', \ldots, q$. Evident că $r(L''_k)$, $k = 1, \ldots, r''$, nu modifică valorile $\hat{v}_0, \ldots, \hat{v}_u$, $u = j'' - 1$ şi valorile $\hat{w}_u, \ldots, \hat{w}_q$, $u = q - j'' + 1$. Deoarece $\hat{w}_q = \hat{v}_0 = 0$ şi cele două clase de lanţuri sunt oarecare rezultă că $\hat{w}_{q-i} = \hat{v}_i$, $i = 0, \ldots, q$. Conform Teoremei 10.8 fluxul dinamic maxim $\hat{f}$ cu sosirea cea mai devreme pentru q perioade de timp este şi un flux dinamic maxim cu plecarea cea mai târzie pentru acelaşi număr de perioade. ■

Exemplul 10.5. Fie fluxul dinamic maxim $\hat{f}$ cu sosirea cea mai devreme pentru 6 perioade de timp reprezentat prin fluxul static maxim $\tilde{f}$ în figura 10.7. Acest flux este şi un flux dinamic maxim cu plecarea cea mai târzie. Simetria opusă în timp dintre plecări şi sosiri este prezentată în tabelul 10.3.

i	0	1	2	3	4	5	6
$\hat{w}_i$	2	2	1	1	0	0	0
$\hat{v}_i$	0	0	0	1	1	2	2

Tabelul 10.3

Teorema 10.10. *Dacă $\tilde{f}_1$ şi $\tilde{f}_2$ sunt două fluxuri statice maxime în reţeaua statică $\tilde{G}$ cu proprietatea că fluxul $\tilde{f}_1$ are valoarea $\tilde{w}_i$ la sursa s_i, $i = 0, \ldots, q$ şi fluxul $\tilde{f}_2$ are valoarea $\tilde{v}_i$ la stocul t_i, $i = 0, \ldots, q$, atunci există un flux static maxim $\tilde{f}_3$ în reţeaua statică $\tilde{G}$ cu proprietatea că are valoarea $\tilde{w}_i$ la sursa s_i, $i = 0, \ldots, q$ şi valoarea $\tilde{v}_i$ la stocul t_i, $i = 0, \ldots, q$.*

Demonstraţie. Fie $\left[\tilde{X}, \overline{\tilde{X}}\right]$ o tăietură minimă $s_i - t_j$, $i = 0, \ldots, q,\ j = 0, \ldots, q$ în reţeaua $\tilde{G}$. Evident că fluxurile statice maxime $\tilde{f}_1$ şi $\tilde{f}_2$ saturează această tăietură. Fie $\tilde{A} = \tilde{A}_1 \cup \tilde{A}_2 \cup \tilde{A}_3$, $\tilde{A}_1 = \{(x, y) \,|\, (x, y) \in A,\ x, y \in \tilde{X}\}$, $\tilde{A}_2 = \{(x, y) \,|\, (x, y) \in \tilde{A},\ x \in \tilde{X}, y \in \overline{\tilde{X}}$ sau $x \in \overline{\tilde{X}}, y \in \tilde{X}\}$, $\tilde{A}_3 = \{(x, y) \,|\, (x, y) \in A,\ x, y \in \overline{\tilde{X}}\}$. Construim fluxul static maxim $\tilde{f}_3$ în modul următor: $\tilde{f}_3(x, y) = \tilde{f}_1(x, y)$ pentru fiecare arc $(x, y) \in \tilde{A}_1$, $\tilde{f}_3(x, y) = \tilde{f}_1(x, y) = \tilde{f}_2(x, y)$ pentru fiecare arc $(x, y) \in \tilde{A}_2$, $\tilde{f}_3(x, y) = \tilde{f}_2(x, y)$ pentru fiecare arc $(x, y) \in \tilde{A}_3$. ■

Conform tabelului 10.2 clasele de fluxuri dinamice, referitoare la plecare şi sosire, sunt următoarele:
(1) plecarea cea mai devreme şi sosirea cea mai devreme;
(2) plecarea cea mai devreme şi sosirea oarecare;
(3) plecarea cea mai devreme şi sosirea cea mai târzie;
(4) plecarea oarecare şi sosirea cea mai devreme;
(5) plecarea oarecare şi sosirea oarecare;
(6) plecarea oarecare şi sosirea cea mai târzie;
(7) plecarea cea mai târzie şi sosirea cea mai devreme;
(8) plecarea cea mai târzie şi sosirea oarecare;
(9) plecarea cea mai târzie şi sosirea cea mai târzie.

Aceste clase de fluxuri dinamice, cu excepţia clasei (5), se numesc *fluxuri dinamice lexicografice* deoarece există o preferinţă lexicografică privind plecarea de la nodurile sursă $s_0, \ldots, s_q$ sau/şi sosirea la nodurile stoc $t_0, \ldots, t_q$.

Teorema 10.11. *În orice reţea dinamică $G = (N, A, c, h, q)$ cu $d_t \leq q$ există oricare flux dinamic lexicografic maxim prezentat mai sus.*

Demonstraţie. Un flux dinamic lexicografic maxim $\hat{f}_3$ se construieşte cum s-a specificat în Teorema 10.10 unde $\hat{f}_1$ este un flux dinamic lexicografic maxim care verifică condiţia de plecare sau/şi $\hat{f}_2$ este un flux dinamic lexicografic maxim care verifică condiţia de sosire. ■

10.4 Aplicaţii şi comentarii bibliografice

Modelul fluxului dinamic apare în problemele de luarea deciziilor în timp. Astfel de probleme apar în următoarele domenii: sistemele de transport, sistemele de evacuare, planificarea economică, sistemele de producţie şi distribuţie, sistemele de comunicare, sistemele energetice etc. În continuare vom prezenta două probleme dinamice concrete, una de transport şi a doua de evacuare.

10.4.1 Problema dinamică de transport

În Bucureşti se constată, la un moment dat, că apa de la robinete este suficient de poluată pentru a nu mai fi recomandată ca apă potabilă. Primăria capitalei ia decizia, ca până la depoluarea apei care alimentează populaţia, să aducă apă minerală de la cea mai apropiată localitate care îmbuteliază această apă. Localitatea este Zizin lângă municipiul Braşov. Transportul este efectuat cu tiruri pe şosele europene şi naţionale. Reţeaua de transport de la Braşov la Bucureşti este reprezentată în figura 10.12, unde pe fiecare arc este trecut numărul perioadelor de timp de parcurgere a rutei reprezentată de arc şi fiecare perioadă de timp este de 15 minute.

Fig.10.12
1 - Braşov, 2 - Sinaia, 3 - Vălenii de Munte,
4 - Câmpulung Muşcel, 5 - Ploieşti, 6 - Târgovişte,
7 - Piteşti, 8 - Bucureşti.

În funcţie de condiţiile concrete de transport se stabileşte o capacitate pe fiecare arc. Se cere trimiterea, cât mai urgent posibil, a unui număr maxim de tiruri încărcate cu apă minerală în primele 12 ore. Rezolvarea problemei constă în determinarea unui flux dinamic maxim cu sosirea cea mai devreme pentru $q = 12 \times 4 = 48$ perioade de timp.

10.4.2 Problema evacuării unei clădiri

În marile oraşe, printre criteriile utilizate la proiectarea unei clădiri mari (bancă, universitate etc.), arhitecţii trebuie să asigure capacităţi suficiente la evacuarea rapidă a clădirii pentru a răspunde, de exemplu: la un incendiu, la un cutremur, la o scurgere de gaz natural sau toxic, la o ameninţare cu bomba etc.

În continuare arătăm cum problema evacuării unei clădiri se modelează ca o problemă de flux dinamic. Se construieşte o reţea dinamică pentru clădire. Nodurile acestei reţele reprezintă sălile de lucru, oficiile, casele scărilor, holurile de la uşile liftului şi ieşirile. Arcele reprezintă pasajele dintre aceste locuri. Nodurile

care reprezintă locurile ce găzduiesc un număr semnificativ de persoane sunt nodurile sursă şi ieşirile clădirii sunt nodurile stoc în reţea.

Capacitatea unui arc este numărul de persoane care pot trece prin pasajul corespunzător arcului pe unitate de timp. De exemplu, dacă anticipăm că 30 de persoane pe minut pot trece prin fiecare punct al pasajului şi lungimea perioadei de timp este 10 secunde, capacitatea arcului respectiv este $30/6 = 5$ unităţi. Se estimează timpul de traversare al fiecărui arc (pasaj) şi reprezintă un număr întreg de perioade de timp. Reamintim că problema fluxului cu surse şi stocuri multiple se poate transforma într-o problemă echivalentă de flux cu o singură sursă şi un singur stoc. Rezolvarea problemei evacuării unei clădiri constă în determinarea unui flux dinamic maxim cu sosirea cea mai devreme pentru un număr de perioade de timp specificat în reţeaua dinamică asociată clădirii.

10.4.3 Comentarii bibliografice

O extensie netrivială a problemei fluxului static este problema fluxului dinamic formulată şi rezolvată de Ford şi Fulkerson (1985a, 1962). Autorii arată echivalenţa dintre un flux dinamic maxim în reţeaua dinamică $G = (N, A, c, h, q)$ şi un flux static maxim în reţeaua statică $\tilde{G} = \left(\tilde{N}, \tilde{A}, \tilde{c}, q\right)$. Pentru cazul staţionar Ford şi Fulkerson au elaborat algoritmul, prezentat în paragraful 2, pentru determinarea unui flux dinamic maxim în reţeaua dinamică G. Acest flux este un flux repetat în timp.

Gale (1959) introduce conceptul de flux dinamic maxim universal (flux dinamic maxim cu sosirea cea mai devreme). În cazul staţionar, independent, Wilkinson (1971) şi Minieka (1973) modifică algoritmul Ford-Fulkerson pentru fluxul dinamic maxim şi obţin algoritmul fluxului dinamic maxim universal prezentat în paragraful 3 sub o formă îmbunătăţită. Fluxurile dinamice lexicografice au fost studiate de Minieka (1973, 1992) şi Ciurea (1985).

Independent, Halpern (1979) şi Ciurea (1979) au arătat că un flux dinamic maxim în reţeaua dinamică $G = (N, A, h, c, q)$ este echivalent cu un flux dinamic maxim în reţeaua statică $G' = (N', A', c', h', q)$, prezentată în paragraful 1, care este echivalent cu un flux static maxim în reţeaua statică $\tilde{G}' = (\tilde{N}', \tilde{A}', \tilde{c}', q)$ (vezi paragraful 1).

În literatura de specialitate au fost prezentate multe generalizări ale problemei fluxului dinamic. Cazul nestaţionar a fost studiat de Halpern (1979), Ciurea (1984, 1987, 1995), Cai, Sha şi Wong (2001) şi alţii. Problema fluxului dinamic cu arcele şi nodurile capacitate apare în lucrările: Ford şi Fulkerson (1962), Halpern (1979), Ciurea (1997b), Cai, Sha şi Wong (2001) etc. Powell, Sheffi şi Thiriez (1984), Ciurea (1982, 1984) studiază unele aspecte în reţele dinamice aleatoare. Problema fluxului dinamic minim a fost abordată de Ciurea (2000a). O excelentă trecere în revistă a problemelor fluxurilor dinamice se găseşte în lucrările:

Leretskiy şi Melamed (1986), Aronson (1989) etc.

Capitolul 11

Algoritmul simplex pentru fluxuri

11.1 Algoritmul simplex pentru fluxul de cost minim

Fie $G = (N, A, c, b, v)$ o reţea orientată cu N mulţimea nodurilor, A mulţimea arcelor, $c : A \to \Re^+$ funcţia capacitate, $b : A \to \Re$ funcţia cost şi $v : N \to \Re$ funcţia valoare. Problema fluxului de cost minim constă în determinarea funcţiei flux $f : A \to \Re^+$ care verifică condiţiile:

$$\text{Minimizează } z(f) = \sum_A b(x,y) f(x,y), \tag{11.1.a}$$

$$\sum_y f(x,y) - \sum_y f(y,x) \;=\; v(x), \;\; x \in N, \tag{11.1.b}$$

$$0 \le f(x,y) \le c(x,y), \;\; (x,y) \in A \tag{11.1.c}$$

cu $\sum_N v(x) = 0$.

Fie $\overline{c} = \max\{c(x,y) \,|\, (x,y) \in A\}$, $\overline{b} = \max\{b(x,y) \,|\, (x,y) \in A\}, \overline{v} = \max\{|v(x)| \mid x \in N\}$ şi f o soluţie admisibilă pentru problema (11.1).

Definiţia 11.1. Se spune că arcul (x,y) este *liber* dacă $0 < f(x,y) < c(x,y)$ şi este *restricţionat* dacă $f(x,y) = 0$ sau $f(x,y) = c(x,y)$.

Definiţia 11.2. Soluţia admisibilă f se numeşte *soluţie fără ciclu* dacă reţeaua G nu conţine cicluri compuse numai din arce libere.

În algoritmul simplex pentru problema fluxului de cost minim, unei soluţii admisibile f i se asociază un arbore parţial $G' = (N, A')$ al lui $G = (N, A)$.

Definiţia 11.3. Soluţia admisibilă f se numeşte *soluţie arbore parţial* dacă toate arcele nonarbore din $\overline{A}'$, $\overline{A}' = A - A'$, sunt restricţionate.

Remarcăm faptul că arcele arbore din A' pot fi fie libere, fie restricţionate.

Exemplul 11.1. Reţeaua din figura 11.1(a) conţine un flux admisibil de-a lungul ciclului $\overset{\circ}{L}= (1, 2, 3, 4, 5, 1)$.

(a) (b)

Fig.11.1. Schimbarea fluxului de-a lungul unui ciclu

Dacă considerăm că $c(x, y) = 7$, $(x, y) \in A$ atunci $r = 3$. De asemenea, $b(\overset{\circ}{L}) = 2-4+1+3-3 = -1$ şi valoarea funcţiei obiectiv va scădea cu $rb(\overset{\circ}{L}) = -3$. Dacă considerăm că $b(1, 2) = 5$ atunci $b(\overset{\circ}{L}) = 2$. În ambele cazuri, noul flux admisibil f' va avea $f'(2, 3) = 0$, respectiv $f'(3, 4) = 0$. În general, pe cel puţin un arc $(x, y) \in \overset{\circ}{L}$ are fluxul $f'(x, y) = 0$ sau $f'(x, y) = c(x, y)$ şi f' este o soluţie fără cicluri.

Algoritmul simplex pentru problema fluxului de cost minim utilizează următorul rezultat fundamental.

Teorema 11.1. *Dacă funcţia obiectiv a problemei fluxului de cost minim este mărginită inferior pe domeniul admisibil, atunci problema are întotdeauna o soluţie fără ciclu optimă.* ■

O soluţie fără ciclu se poate converti într-o soluţie arbore parţial. Arcele libere corespunzătoare unei soluţii fără ciclu definesc o pădure. Dacă această pădure este un arbore parţial, atunci soluţia fără ciclu este o soluţie arbore parţial. În caz contrar, se adaugă la această pădure arce restricţionate pentru a obţine un arbore parţial.

Exemplul 11.2. Fluxul din reţeaua prezentată în figura 11.2(a) este o soluţie fără cicluri. Mulţimea arcelor libere defineşte pădurea prezentată în figura 11.2(b). În figura 11.2(c) sunt prezentate două soluţii arbore parţial corespunzătoare soluţiei fără ciclu. Deci, unei soluţii fără ciclu date poate să-i corespundă mai multe soluţii arbore parţial.

Fig.11.2. Convertirea unei soluţii fără ciclu într-o soluţie arbore parţial.

Avem următorul rezultat fundamental.

Teorema 11.2. *Dacă funcţia obiectiv a problemei fluxului de cost minim este mărginită inferior pe domeniul admisibil, atunci problema are întotdeauna o soluţie arbore parţial optimă.* ■

O soluţie arbore parţial partiţionează mulţimea arcelor A în trei submulţimi:

1. A', mulţimea arcelor din arborele parţial $G' = (N, A')$;
2. $\overline{A}_0{}'$, mulţimea arcelor nonarbore (x, y) pentru care $f(x, y) = 0$;

3. $\overline{A}_c{}'$, mulţimea arcelor nonarbore (x, y) pentru care $f(x, y) = c(x, y)$.

Definiţia 11.4. Se numeşte *structură arbore parţial* tripletul $\left(A', \overline{A}_0{}', \overline{A}_c{}'\right)$.

Mulţimea arcelor arbore A' conţine toate arcele libere şi eventual şi arce restricţionate în cazul când arcele libere nu formează un arbore parţial.

Am arătat că fiecărei soluţii arbore parţial i se poate asocia o structură arbore parţial. Se poate proceda şi invers, unei structuri arbore parţial să-i punem în corespondenţă o soluţie arbore parţial f în modul următor: $f(x, y) = 0$ pentru fiecare arc (x, y) din $\overline{A}_0{}'$, $f(x, y) = c(x, y)$ pentru fiecare arc (x, y) din $\overline{A}_c{}'$ şi $f(x, y)$ pentru fiecare arc (x, y) din A' se obţine rezolvând sistemul de restricţii de conservare a fluxului.

Definiţia 11.5. Se spune că o structură arbore parţial este *admisibilă* dacă soluţia arbore parţial corespunzătoare este admisibilă.

Definiţia 11.6. Se spune că arborele parţial $G' = (N, A')$ este *nedegenerat* dacă toate arcele din A sunt libere; în caz contrar se spune că arborele parţial este *degenerat.*

Definiţia 11.7. Se spune că o structură arbore parţial este *optimală* dacă soluţia arbore parţial corespunzătoare este o soluţie optimă a problemei fluxului de cost minim.

În capitolele anterioare s-au definit costurile reduse prin $b^p(x, y) = b(x, y) - p(x) + p(y)$ pentru fiecare arc $(x, y) \in A$, unde $p(x)$ este potenţialul nodului x, $x \in N$.

Teorema 11.3. *O structură arbore parţial* $\left(A', \overline{A}_0{}', \overline{A}_c{}'\right)$ *este optimală pentru problema fluxului de cost minim dacă este admisibilă şi dacă există un vector potenţial* p *pentru care costurile reduse* $b^p(x, y)$ *satisfac condiţiile:*

$$b^p(x, y) = 0 \quad \textit{pentru toate arcele } (x, y) \in A' \tag{11.2.a}$$

$$b^p(x, y) \geq 0 \quad \textit{pentru toate arcele } (x, y) \in \overline{A}_0{}' \tag{11.2.b}$$

$$b^p(x, y) \leq 0 \quad \textit{pentru toate arcele } (x, y) \in \overline{A}_c{}'. \tag{11.2.c}$$

Demonstraţie. Fie f^* soluţia asociată cu structura arbore parţial $\left(A', \overline{A}_0{}', \overline{A}_c{}'\right)$. Cunoaştem că un vector potenţial p împreună cu structura arbore parţial $\left(A', \overline{A}_0{}', \overline{A}_c{}'\right)$ satisfac condiţiile (11.2). Trebuie să arătăm că f^* este o soluţie optimă a problemei fluxului de cost minim.

În capitolele anterioare s-a arătat că expresia

$$\text{minimizează } \sum_A b(x, y) f(x, y)$$

este echivalentă cu expresia

$$\text{minimizează } \sum_{A} b^p(x,y) f(x,y).$$

Condiţiile (11.2) implică faptul că expresia

$$\text{minimizează } \sum_{A} b^p(x,y) f(x,y)$$

este echivalentă cu expresia

$$\text{minimizează } \sum_{\overline{A}_0'} b^p(x,y) f(x,y) - \sum_{\overline{A}_c'} |b^p(x,y)|\, f(x,y). \tag{11.3}$$

Definiţia soluţiei f^* implică faptul că pentru oricare soluţie arbitrară f, $f^*(x,y) \leq f(x,y)$ pentru toate arcele $(x,y) \in \overline{A}_0'$ şi $f^*(x,y) \geq f(x,y)$ pentru toate arcele $(x,y) \in A_c'$. Din (11.3) rezultă că $z(f^*) \leq z(f)$ şi f^* este o soluţie optimă a problemei fluxului de cost minim. ■

Algoritmul simplex va menţine pe parcursul execuţiei o structură arbore parţial admisibilă şi iterativ o va transforma într-o structură arbore parţial optimă. La fiecare iteraţie, algoritmul adaugă un arc la arborele parţial în locul unuia din arcele curente. Arcul de intrare (adăugat la arborele parţial) este un arc nonarbore care nu verifică condiţiile (11.2). Algoritmul execută următoarele operaţii:

1. adaugă un arc nonarbore la arborele parţial care creează un ciclu de cost negativ, eventual cu capacitatea reziduală zero;

2. trimite cantitatea de flux maximă posibilă de-a lungul ciclului după care cel puţin un arc al ciclului devine arc restricţionat;

3. elimină un arc restricţionat din ciclu obţinând o nouă structură arbore parţial.

Din cauza relaţiei cu algoritmul simplex pentru problema programării liniare, operaţia de transformare a unei structuri arbore parţial în alta este numită *operaţie pivot*, şi două structuri arbori parţiali obţinute în iteraţii consecutive sunt numite *structuri arbori parţiali adiacenţi.*

Înainte de a prezenta în detaliu algoritmul simplex pentru problema fluxului de cost minim vom arăta cum se calculează vectorul potenţial şi fluxul.

Deoarece algoritmul simplex generează o secvenţă de soluţii arbore parţial, pentru implementarea algoritmului trebuie să putem reprezenta convenabil arborii parţiali într-un calculator. Considerăm că arborele are un nod rădăcină şi

vom nota cu r nodul rădăcină. Sunt utilizate trei tablouri unidimensionale fiecare cu n componente.

Tabloul *predecesor,* notat π, se defineşte în modul următor: dacă $x \in N$, atunci $\pi(x) = y$, unde y este penultimul nod din lanţul unic de la nodul rădăcină r la nodul x în arborele parţial; prin convenţie $\pi(r) = 0$.

Tabloul *adâncime,* notat η, se defineşte în modul următor: dacă $x \in N$, atunci $\eta(x) = k$, unde k este numărul de arce din lanţul unic de la nodul rădăcină r la nodul x în arborele parţial; prin convenţie $\eta(r) = 0$.

Tabloul *conductor,* notat ω, se defineşte în modul următor: dacă $x \in N$, atunci $\omega(x) = y$, unde y este următorul nod al lui x în parcurgerea în preordine a arborelui parţial; prin convenţie $y = r$ dacă x este ultimul nod din secvenţă.

În arborele parţial $G' = (N, A')$, un nod z se numeşte *succesorul* lui x dacă $\pi(z) = x$, indiferent dacă $(x, z) \in A'$ sau $(z, x) \in A'$.

Exemplul 11.3. În figura 11.3(a) este reprezentat arborele parţial şi în figura 11.3(b) sunt reprezentate tablourile π, η şi ω. Fie $x = 6$, lanţul unic de la nodul rădăcină $r = 1$ la nodul $x = 6$ este $L = (1, 2, 5, 6)$ şi parcurgerea arborelui în preordine este 1, 2, 5, 6, 8, 9, 7, 3, 4. Avem $\pi(6) = 5,\ \eta(6) = 3,\ \omega(6) = 8$; nodurile 6 şi 7 sunt succesori ai nodului 5.

(a) (b)

Fig.11.3. Reprezentarea unui arbore

Obţinerea unei structuri arbore parţial iniţială se poate realiza în două moduri. Un prim mod constă în determinarea unui flux admisibil prin rezolvarea problemei fluxului maxim în reţea şi construim soluţia arbore parţial aşa cum s-a precizat mai sus. Un al doilea mod constă în următoarele. Presupunem că pentru oricare nod y din N, $y \neq r = 1$, avem $(1, y) \in A$ şi $(y, 1) \in A$. Această ipoteză nu micşorează generalitatea problemei, deoarece dacă unele din aceste arce nu există în reţea, atunci adăugăm aceste arce reţelei cu costuri suficient de mari astfel încât ele nu vor putea să apară în soluţia optimă. Pentru fiecare nod y din N, $y \neq 1$ adăugăm mulţimii A' arcul $(1, y)$ sau $(y, 1)$ în modul următor: dacă $v(y) \geq 0$, atunci adăugăm arcul $(y, 1)$ cu $f(y, 1) = v(y)$, altfel adăugăm arcul $(1, y)$ cu $f(1, y) = -v(y)$. Mulţimea $\overline{A}_0{}'$ conţine arcele rămase şi mulţimea $\overline{A}_c{}'$ este mulţimea vidă.

În continuare trebuie calculat vectorul p. Deoarece $b^p(x, y) = b(x, y) - p(x) + p(y) = b(x, y) - (p(x) + k) + (p(y) + k)$, rezultă că vectorul p este unic determinat abstracţie făcând de o translaţie de o constantă k. De aceea putem considera că $p(1) = 0$. Condiţiile de optimalitate impun pentru $(x, y) \in A'$ ca $b^p(x, y) = 0$. Astfel putem determina vectorul p rezolvând sistemul:

$$b(x,y) - p(x) + p(y) = 0, \ \ (x,y) \in A'. \tag{11.4}$$

Deoarece $|A'| = n-1$ şi $p(1) = 0$, sistemul are $n-1$ ecuaţii şi $n-1$ necunoscute şi determinantul lui este nenul. Deci sistemul este compatibil determinat şi se poate determina vectorul p pentru care sunt verificate condiţiile de optimalitate. Sistemul poate fi rezolvat cu PROCEDURA POTENŢIAL1.

```
(1) PROCEDURA POTENŢIAL1;
(2) BEGIN
(3)     p(1) := 0;
(4)     y  := ω(1);
(5)     WHILE y ≠ 1 DO
(6)     BEGIN
(7)         x := π(y);
(8)         IF(x,y) ∈ A THEN p(y) := p(x) − b(x,y);
(9)         IF(y,x) ∈ A THEN p(y) := p(x) + b(y,x);
(10)        y := ω(y);
(11)    END;
(12) END;
```

Exemplul 11.4. Aplicăm procedura pentru arborele parţial reprezentat în figura 11.4. Numărul asociat fiecărui arc reprezintă costul acestui arc.

Fig.11.4

Iniţial avem: $\pi = (0,1,2,3,2,5,5)$, $\omega = (2,5,4,1,6,7,3)$, $p(1) = 0$,
$y = \omega(1) = 2$.

Prin execuţia ciclului WHILE se obţin următoarele:

$x = \pi(2) = 1,\ (1,2) \in A,\ p(2) = p(1) - b(1,2) = \ \ 0 - 5 = -5,\ y = \omega(2) = 5;$
$x = \pi(5) = 2,\ (5,2) \in A,\ p(5) = p(2) + b(5,2) = -5 + 2 = -3,\ y = \omega(5) = 6;$
$x = \pi(6) = 5,\ (5,6) \in A,\ p(6) = p(5) - b(5,6) = -3 - 4 = -7,\ y = \omega(6) = 7;$
$x = \pi(7) = 5,\ (7,5) \in A,\ p(7) = p(5) + b(7,5) = -3 + 1 = -2,\ y = \omega(7) = 3;$
$x = \pi(3) = 2,\ (3,2) \in A,\ p(3) = p(2) + b(3,2) = -5 + 3 = -2,\ y = \omega(3) = 4;$
$x = \pi(4) = 3,\ (4,3) \in A,\ p(4) = p(3) + b(4,3) = -2 + 1 = -1,\ y = \omega(4) = 1;$
STOP.

Teorema 11.4. *PROCEDURA POTENŢIAL1 calculează corect vectorul p.*
Demonstraţie. Evident. ■

Teorema 11.5. *PROCEDURA POTENŢIAL1 are complexitatea $O(n)$.*
Demonstraţie. Evident. ■

Teorema 11.6. *Dacă toate costurile sunt întregi, atunci problema fluxului de cost minim are întotdeauna potenţialele nod optimale întregi.*

Demonstraţie. Evident. ■

Dacă se dă o structură arbore parţial $\left(A', \overline{A}_0', \overline{A}_c'\right)$, atunci determinarea unui flux pentru această structură se face în modul următor. Mai întâi precizăm faptul că dacă eliminăm arcul (x, y) din arborele parţial $G' = (N, A')$ atunci obţinem doi subarbori $G_1' = (N_1, A_1')$ şi $G_2' = (N_2, A_2')$ cu $x \in N_1$ şi $y \in N_2$. Deoarece

$$\sum_{N_1} v(z) + \sum_{N_2} v(z) = \sum_{N} v(z) = 0$$

rezultă că în arborele G' pe arcul (x, y) se transportă de la subarborele G_1 la subarborele G_2, $\sum_{N_1} v(z) = -\sum_{N_2} v(z)$ unităţi de flux. Dacă presupunem că $N_2 = \{y\}$ atunci pe arcul (x, y) se transportă $-v(y)$ unităţi de flux. Dacă $f(x, y) = -v(y)$, atunci $v(x) := v(x) + v(y)$ şi $v(y) := v(y) - v(y) = 0$. Putem elimina nodul y şi arcul (x, y) din G' şi să continuăm procedeul. Această idee stă la baza PROCEDURII FLUX.

```
(1) PROCEDURA FLUX;
(2) BEGIN
(3)     FOR x ∈ N DO v'(x) := v(x);
(4)     FOR (x,y) ∈ Ā0' DO f(x,y) := 0;
(5)     FOR (x,y) ∈ Āc' DO BEGIN f(x,y) := c(x,y);
(6)                               v'(x) := v'(x) - c(x,y);
(7)                               v'(y) := v'(y) + c(x,y);
(8)                            END;
(9)     N' := N;  A'' := A';
(10)    WHILE N' :≠ {1} DO
(11)    BEGIN
(12)        se selectează un nod terminal y din G'' = (N', A'')
(13)        x := π(y);
(14)        IF (x,y) ∈ A'' THEN f(x,y) := -v'(x)
(15)                       ELSE f(y,x) := -v'(y);
(16)        v'(x) := v'(x) + v'(y);
(17)        N' := N' - {y};  A'' := A'' - {(x,y)};
(18)    END;
(19) END;
```

Exemplul 11.5 Considerăm reţeaua reprezentată în figura 11.5(a). Structura $\left(A', \overline{A}_0', \overline{A}_c'\right)$ este reprezentată în figura 11.5(b), unde $A' = \{(1,2), (1,3), (2,4), (2,5), (5,6)\}$ şi fiecare arc este reprezentat prin linie continuă, $\overline{A}_0' = \{(2,3), (5,4)\}$ şi fiecare arc este reprezentat prin linie întreruptă, $\overline{A}_c' = \{(3,5), (4,6)\}$ şi fiecare arc este reprezentat prin linie punctată. Aplicând PROCEDURA FLUX rezultă

următoarele:

(a) (b)

Fig.11.5

Liniile (3) - (9) au efectul: $v' = (9, 0, 0, 0, 0, -9)$; $f(2,3) = 0$, $f(5,4) = 0$; $f(3,5) = 3$, $f(4,6) = 5$, $v' = (9, 0, -3, -5, 3, -4)$, $N' = \{1, 2, 3, 4, 5, 6\}$, $A'' = \{(1,2), (1,3), (2,4), (2,5), (5,6)\}$; de asemenea, avem $\pi = (0, 1, 1, 2, 2, 5)$.
Iteraţiile instrucţiunii WHILE sunt următoarele:

Iteraţia 1.
$y = 6$, $x = 5$, $(5,6) \in A''$, $f(5,6) = 4$, $v'(5) = -1$, $N' = \{1, 2, 3, 4, 5\}$, $A'' = \{(1,2), (1,3), (2,4), (2,5)\}$.

Iteraţia 2.
$y = 5$, $x = 2$, $(2,5) \in A''$, $f(2,5) = 1$, $v'(2) = -1$, $N' = \{1, 2, 3, 4\}$, $A'' = \{(1,2), (1,3), (2,4)\}$.

Iteraţia 3.
$y = 4$, $x = 2, (2,4) \in A'', f(2,4) = 5, v'(2) = -6, N' = \{1, 2, 3\}, A'' = \{(1,2), (1,3)\}$.

Iteraţia 4.
$y = 3$, $x = 1$, $(1,3) \in A''$, $f(1,3) = 3$, $N' = \{1, 2\}$, $A'' = \{(1,2)\}$.

Iteraţia 5.
$y = 2$, $x = 1$, $(1,2) \in A''$, $f(1,2) = 6$, $N' = \{1\}$, $A'' = \emptyset$ STOP.

Fluxurile de arc $f(x,y)$ obţinute sunt numerele ataşate arcelor reţelei reprezentate în figura 11.5(b).

Teorema 11.7. *PROCEDURA FLUX calculează corect fluxurile de arc* $f(x,y)$, $(x,y) \in A$, *pentru o structură dată* $\left(A', \overline{A}_0', \overline{A}_c'\right)$.

Demonstraţie. Demonstraţia teoremei rezultă din faptul că la fiecare iteraţie se poate determina un nod terminal din G' deoarece, după fiecare iteraţie rămâne arbore şi are cel puţin un nod terminal şi din modul cum sunt calculate fluxurile de arc $f(x,y)$ în procedură. ■

Teorema 11.8. *PROCEDURA FLUX are complexitatea* $O(m)$.

Demonstraţie. Partea de procedură formată din liniile (3) - (9) are complexitatea $O(m)$ şi partea de procedură formată din liniile (10) - (18) are complexitatea $O(n)$. Deci procedura are complexitatea $O(m) + O(n) = O(m)$. ■

Algoritmul simplex pentru problema fluxului de cost minim menţine o structură arbore parţial admisibilă la fiecare iteraţie şi o transformă succesiv într-o structură arbore parţial îmbunătăţită până când devine optimă. Acest algoritm este prezentat în PROGRAM SIMPLEX.

(1) PROGRAM SIMPLEX;
(2) BEGIN
(3) se determină o structură arbore admisibilă iniţială $\left(A', \overline{A}_0', \overline{A}_c'\right)$;

(4) fie f fluxul şi p vectorul potenţial care corespund acestei structuri;
(5) WHILE există arc nonarbore care nu verifică condiţiile de optimilate DO
(6) BEGIN
(7) se selectează un arc (x', y') care nu verifică condiţiile de optimalitate;
(8) adaugă arcul de intrare (x', y') la arborele curent $G' = (N, A')$;
(9) se determină ciclul $\overset{\circ}{L}$ creat în digraful $\tilde{G}' = \left(N, \tilde{A}'\right)$, $\tilde{A}' = A' \cup \{(x', y')\}$;
(10) se actualizează f pe $\overset{\circ}{L}$ şi se determină arcul de ieşire (x'', y'') din $\tilde{G}'$;
(11) se actualizează $\left(A', \overline{A}_0', \overline{A}_c'\right)$ şi p;
(12) END
(13) END.

O structură arbore parţial admisibilă $\left(A', \overline{A}_0', \overline{A}_c'\right)$ verifică întotdeauna condiţiile de optimalitate (11.2a). Pentru ca o astfel de structură să fie optimă trebuie să verifice şi condiţiile (11.2b) şi (11.2c).

Definiţia 11.8. Un arc (x, y) se numeşte *eligibil* dacă verifică una din condiţiile:
(e_1) $b^p(x, y) < 0$, dacă $(x, y) \in \overline{A}_0'$,
(e_2) $b^p(x, y) > 0$, dacă $(x, y) \in \overline{A}_c'$.

Arcul de intrare (x', y') poate fi orice arc eligibil. Una din metodele clasice pentru determinarea arcului de intrare (x', y') constă în alegerea arcului eligibil cu abaterea de la optimalitate maximă, adică cu valoarea $|b^p(x', y')|$ maximă. Principalul dezavantaj al acestei metode este dat de modul de implementare în care trebuie parcurse toate arcele eligibile, dovedindu-se practic că nu este cea mai avantajoasă, chiar dacă ea îmbunătăţeşte soluţia cu valoarea cea mai bună raportat la o iteraţie.

Arcul de intrare (x', y') creează un ciclu unic $\overset{\circ}{L}$ în digraful obţinut $\tilde{G}' = \left(N, \tilde{A}'\right)$, $\tilde{A}' = A' \cup \{(x', y')\}$ pe care îl vom numi *ciclu pivot.* Dacă $(x, y) \in \overline{A}_0'$, atunci definim orientarea ciclului $\overset{\circ}{L}$ aceeaşi cu orientarea arcului (x', y') şi invers orientării acestui arc dacă $(x', y') \in \overline{A}_c'$. Fie $\overset{\circ}{L}{}^+$ mulţimea arcelor directe şi $\overset{\circ}{L}{}^-$ mulţimea arcelor inverse ale ciclului $\overset{\circ}{L}$. Fie capacităţile reziduale

$$r(x, y) = \begin{cases} c(x, y) - f(x, y), & \text{dacă} \quad (x, y) \in \overset{\circ}{L}{}^+, \\ f(x, y), & \text{dacă} \quad (x, y) \in \overset{\circ}{L}{}^- \end{cases}$$

şi capacitatea reziduală a ciclului $r(\overset{\circ}{L}) = \min\{r(x, y) \mid (x, y) \in \overset{\circ}{L}\}$.

Fluxul de-a lungul ciclului se modifică astfel:

$$f(x, y) = \begin{cases} f(x, y) + r(\overset{\circ}{L}), & \text{dacă} \quad (x, y) \in \overset{\circ}{L}{}^+, \\ f(x, y) - r(\overset{\circ}{L}), & \text{dacă} \quad (x, y) \in \overset{\circ}{L}{}^- . \end{cases}$$

Definiţia 11.9. Un arc $(x,y) \in \overset{\circ}{L}$ cu $r(x,y) = r(\overset{\circ}{L})$ se numeşte *arc de blocare*. O iteraţie a algoritmului se numeşte *nedegenerată* dacă $r(\overset{\circ}{L}) > 0$ şi *degenerată* dacă $r(\overset{\circ}{L}) = 0$.

Evident că, o iteraţie este degenerată numai dacă G' este degenerat.

Arcul de ieşire (x'', y''), care se elimină din digraful $\tilde{G}'$, este orice arc de blocare. Pentru a determina arcul de ieşire (x'', y'') trebuie să identificăm mai întâi ciclul $\overset{\circ}{L}$. Identificarea ciclului $\overset{\circ}{L}$ se poate realiza cu PROCEDURA CICLU.

```
(1) PROCEDURA CICLU;
(2) BEGIN
(3)     x := x';  y := y';
(4)     WHILE x ≠ y DO
(5)         BEGIN
(6)             IF η(x) > η(y)
(7)                 THEN x := π(x)
(8)                 ELSE IF η(y) > η(x)
(9)                         THEN y := π(y)
(10)                        ELSE BEGIN x := π(x); y := π(y); END;
(11)        END;
(12) END;
```

Exemplul 11.6. Fie arborele $G' = (N, A')$ reprezentat în figura 11.6(a) şi ciclul $\overset{\circ}{L}$ al digrafului $\tilde{G}' = \left(N, \tilde{A}'\right)$ obţinut prin introducerea arcului $(3,5) \in \overline{A}_c{}'$ este reprezentat în figuta 11.6(b). Avem $\pi = (0,1,1,2,2,5),\ \eta = (0,1,1,2,2,3)$. Orientarea ciclului $\overset{\circ}{L}$ va fi invers orientării arcului $(3,5)$. Aplicând PROCEDURA CICLU obţinem următoarele: $x = 3,\ y = 5$;

(a) (b)

Fig.11.6

Iteraţia 1: $\eta(5) > \eta(3),\ y = \pi(5) = 2$
Iteraţia 2: $\eta(2) = \eta(3),\ x = \pi(3) = 1,\ y = \pi(2) = 1$; STOP

Ciclul este $\overset{\circ}{L}= (1,2,5,3,1)$.

Precizăm faptul că, modificarea fluxului pe ciclul $\overset{\circ}{L}$ va scădea valoarea funcţiei obiectiv cu $r(\overset{\circ}{L}) \mid b^p(x', y') \mid$.

Teorema 11.9. *PROCEDURA CICLU determină corect ciclul* $\overset{\circ}{L}$ *al digrafului* $\tilde{G}' = \left(N, \tilde{A}'\right),\ \tilde{A}' = A' \cup \{(x', y')\}$.

Demonstraţie. Corectitudinea algoritmului rezultă din modul cum au fost definiţi vectorii π, η şi din modul cum este determinat ciclul $\overset{\circ}{L}$ în algoritm. ■

Teorema 11.10. *PROCEDURA CICLU are complexitatea* $O(n)$.

Demonstraţie. Evident. ■

Pentru actualizarea structurii arbore parţial $\left(A', \overline{A}_0', \overline{A}_c'\right)$ şi a vectorului potenţial p există două cazuri:

$c_1)$ $(x'', y'') = (x', y') \left(r(\overset{\circ}{L}) = r(x', y') = c(x', y')\right)$.

Deoarece $(x'', y'') = (x', y')$ A' rămâne nemodificată şi arcul (x', y') trece din $\overline{A}_0'$ în $\overline{A}_c'$ sau viceversa. Vectorul potenţial p rămâne neschimbat.

$c_2)$ $(x'', y'') \neq (x', y')$.

În acest caz $A' := A' \cup \{(x', y')\} - \{(x'', y'')\}$ şi arcul (x'', y'') este introdus în $\overline{A}_0'$ sau $\overline{A}_c'$ după cum $f(x'', y'') = 0$ respectiv $f(x'', y'') = c(x'', y'')$. Vectorul potenţial p trebuie modificat. Arcul de ieşire (x'', y'') partiţionează mulţimea nodurilor N în două submulţimi: N_1 care conţine nodul rădăcină şi $N_2 = N - N_1$; arcul de intrare (x', y') are o extremitate în N_1 şi cealaltă extremitate în N_2. Condiţiile $p(1) = 0, b^p(x, y) = b(x, y) - p(x) + p(y) = 0$ sunt verificate în subarborele $G_1' = (N_1, A_1')$. Pentru ca $b^p(x, y) = b(x, y) - p(x) + p(y) = 0$ să fie verificate şi în subarborele $G_2' = \left(N_2, \overline{A}_2'\right)$ trebuie să modificăm potenţialele nod după cum urmează:

- dacă $x' \in N_1$ şi $y' \in N_2$, atunci $p(y) := p(y) - b^p(x', y'),\ y \in N_2$;
- dacă $x' \in N_2$ şi $y' \in N_1$, atunci $p(y) := p(y) + b^p(x', y'),\ y \in N_2$.

PROCEDURA POTENŢIAL2 calculează $p(y)$ pentru toate nodurile y din N_2.

```
(1) PROCEDURA POTENŢIAL2;
(2) BEGIN
(3)     IF y'' ∈ N2 THEN y := y''
(4)                 ELSE y := x'';
(5)     IF x' ∈ N1 THEN β := −b^p(x', y')
(6)                 ELSE β := b^p(x', y');
(7)      p(y) := p(y) + β;
(8)      z := ω(y);
(9)      WHILE η(z) > η(y) DO
(10)     BEGIN
(11)         p(z) := p(z) + β;
(12)         z := ω(z);
(13)     END;
(14) END;
```

Exemplul 11.7. Fie arborele $G' = (N, A')$ reprezentat în figura 11.7(a) cu

$\eta = (0, 1, 1, 2, 2, 3)$, $\omega = (0, 5, 2, 1, 6, 4)$, $\overline{A}_0' = \{(2, 3), (5, 4)\}$, $\overline{A}_c' = \{(3, 5), (4, 6)\}$.

(a) (b)

Fig.11.7

Arcul de intrare este $(x', y') = (3, 5)$ şi arcul de ieşire este $(x'', y'') = (2, 5)$. Se obţine noul arbore $G' = (N, A')$ reprezentat în figura 11.7(b). Aplicăm PROCEDURA POTENŢIAL pentru calcularea vectorului p. Arcul de ieşire $(x'', y'') = (2, 5)$ partiţionează mulţimea nodurilor $N = \{1, 2, 3, 4, 5, 6\}$ în $N_1 = \{1, 2, 3, 4\}$ şi $N_2 = \{5, 6\}$. Iniţializările sunt următoarele: $j = 5,\ 3 \in N_1$ şi $\beta = b^p(3, 5) = b(3, 5) - p(3) + p(5) = 2 + 3 - 6 = -1,\ p(5) := p(5) + \beta = -5 - 1 = -6,\ z = \omega(5) = 6$; execuţia instrucţiunii WHILE produce următoarele:
Iteraţia 1: $\eta(6) > \eta(5); p(6) := p(6) + \beta = -9 - 1 = -10,\ z = \omega(6) = 4$
Iteraţia 2: $\eta(4) = \eta(5)$ STOP

Teorema 11.11. *PROCEDURA POTENŢIAL2 calculează corect potenţialele nod corespunzătoare noului arbore* $G' = (N, A')$.
Demonstraţie. Corectitudinea algoritmului rezultă din precizările făcute mai sus şi modul de calcul al potenţialelor nod $p(y)$ pentru toate nodurile y din N_2. ■
Teorema 11.12. *PROCEDURA POTENŢIAL2 are complexitatea* $O(n)$.
Demonstraţie. Evident. ■

După explicitarea algoritmului simplex pentru problema fluxului de cost minim putem să prezentăm un exemplu.
Exemplul 11.8. Fie problema fluxului de cost minim din reţeaua reprezentată în figura 11.8(a). Transportând 9 unităţi de flux de la nodul $s = 1$ la nodul $t = 6$ se obţine fluxul f şi structura arbore parţial $\left(A', \overline{A}_0', \overline{A}_c'\right)$ reprezentate în figura 11.8(b) unde $A' = \{(1, 2), (1, 3), (2, 4), (2, 5), (5, 6)\}$ şi fiecare arc este reprezentat prin linie continuă, $\overline{A}_0' = \{(2, 3), (5, 4)\}$ şi fiecare arc este reprezentat prin linie întreruptă, $\overline{A}_c' = \{(3, 5), (4, 6)\}$ şi fiecare arc este reprezentat prin linie punctată. Vectorul $p = (0, -3, -2, -8, -5, -9)$ se obţine cu PROCEDURA POTENŢIAL1. Calculăm costurile reduse: $b^p(2, 3) = 3$, $b^p(5, 4) = 2$, $b^p(3, 5) = 1$, $b^p(4, 6) = 2$. Rezultă că arcele din $\overline{A}_c'$ sunt eligibile.
Iteraţia 1. Fie arcul de intrare $(x', y') = (3, 5)$. Cu PROCEDURA CICLU determinăm ciclul $\overset{\circ}{L} = (1, 2, 5, 3, 1)$ cu $r(\overset{\circ}{L}) = \min\{2, 1, 3, 3\} = 1$ şi se modifică fluxul pe acest ciclu. Arcul $(2, 5)$ este unicul arc de blocare şi deci arcul de ieşire este $(x'', y'') = (2, 5)$. Noua structură arbore parţial $\left(A', \overline{A}_0', \overline{A}_c'\right)$ este reprezentată în figura 11.8(c), unde vectorul p a fost actualizat cu PROCEDURA POTENŢIAL2.

Fig.11.8

Calculăm costurile reduse: $b^p(2,3) = 3$, $b^p(5,4) = 3$, $b^p(2,5) = -1$, $b^p(4,6) = 1$.

Iteraţia 2. $(x', y') = (4,6)$, $\overset{\circ}{L}$= $(1,3,5,6,4,2,1)$, $r(\overset{\circ}{L}) = 1$, $(x'', y'') = (3,5)$. Noua structură arbore parţial $\left(A', \overline{A}_0{}', \overline{A}_c{}'\right)$ este reprezentată în figura 11.8(d).

Calculăm costurile reduse: $b^p(2,3) = 3$, $b^p(5,4) = 4$, $b^p(2,5) = -2$, $b^p(3,5) = -1$. Condiţiile de optimalitate sunt verificate şi fluxul din reţeaua reprezentată în figura 11.8(d) este de cost minim.

Algoritmul simplex pentru fluxul de cost minim nu se termină în mod necesar într-un număr finit de iteraţii. Problema degenerării şi ciclării nu este numai o problemă teoretică ci de asemenea una practică. Studiile au arătat că mai mult de 90% din operaţiile pivot pot fi degenerate în cazul în care $r(\overset{\circ}{L}) = 0$ adică atunci când valoarea funcţiei obiectiv rămâne neschimbată. Acest lucru se poate întâmpla atunci când arborele parţial este degenerat, adică conţine arce restricţionate şi orientarea ciclului $\overset{\circ}{L}$ coincide cu orientarea unui arc $(x,y) \in \overset{\circ}{L}$ cu $f(x,y) = c(x,y)$ sau orientarea ciclului $\overset{\circ}{L}$ este inversă unui arc $(x,y) \in \overset{\circ}{L}$ cu $f(x,y) = 0$.

În aceste condiţii arcul (x,y) este arc de blocare şi $r(\overset{\circ}{L}) = 0$.

Degenerarea conduce la ciclare, adică la repetarea unor secvenţe pentru care $r(\overset{\circ}{L}) = 0$ şi deci la imposibilitatea de a ieşi din algoritm. Pentru a se evita degenerarea şi deci ciclarea se va menţine o structură arbore parţial specială numită *structură arbore parţial tare admisibilă.* În continuare vom prezenta cum se poate obţine o astfel de structură iniţială şi cum se poate menţine acest tip de structură pe parcursul iteraţiilor.

Fie structura arbore parţial $\left(A', \overline{A}_0{}', \overline{A}_c{}'\right)$ cu r nodul rădăcină. Fiecare arc arbore din A' este direcţionat fie către nodul rădăcină r şi îl numim *arc interior*, fie în sens contrar nodului rădăcină r şi îl numim *arc exterior.*

Definiţia 11.10. Arborele parţial $G' = (N, A')$ se numeşte *tare admisibil* dacă fiecare arc arbore (x,y) cu $f(x,y) = 0$ este arc interior şi fiecare arc arbore (x,y) cu $f(x,y) = c(x,y)$ este arc exterior.

Analizând definiţia de mai sus putem obţine o exprimare echivalentă a arborelui parţial tare admisibil.

Definiţia 11.11. Arborele parţial $G' = (N, A')$ se numeşte *tare admisibil* dacă putem trimite o cantitate $\delta > 0$ de flux de la orice nod x la nodul rădăcină r de-a lungul unui lanţ arbore L fără a încălca restricţiile de mărginire a fluxului.

Definiţia 11.12 Structura arbore parţial $\left(A', \overline{A}_0{}', \overline{A}_c{}'\right)$ se numeşte *tare admisibilă* dacă arborele parţial $G' = (N, A')$ este tare admisibil.

Pentru implementarea algoritmului simplex trebuie să determinăm un arbore parţial tare admisibil iniţial. Folosind metoda prezentată anterior pentru construirea structurii arborelui parţial iniţial se obţine un arbore parţial tare admisibil.

Precizăm faptul că un arbore parţial nedegenerat este întotdeauna tare admisibil şi că un arbore parţial degenerat poate sau nu poate să fie tare admisibil. Algoritmul simplex creează un arbore parţial degenerat dintr-un arbore parţial nedegenerat ori de câte ori două sau mai multe arce pot fi arce de ieşire şi selectăm unul dintre acestea. Deci regula arcului de ieşire trebuie să asigure că următorul arbore parţial este tare admisibil.

Presupunem că avem un arbore parţial tare admisibil şi, în timpul operaţiei pivot, arcul (x', y') este adăugat arborelui parţial. Considerăm cazul când $f(x', y') = 0$. Fie $\overset{\circ}{L}$ ciclul format de arcul de intrare (x', y') şi z nodul ciclului cel mai apropiat de nodul rădăcină r (z este primul nod ascendent comun al nodurilor x' şi y'); nodul z îl numim *nod pisc.* Definim orientarea ciclului $\overset{\circ}{L}$ aceeaşi cu a arcului de intrare (x', y'). După schimbarea fluxului pe ciclul $\overset{\circ}{L}$ cu $r(\overset{\circ}{L})$, algoritmul identifică arcele de blocare (arcele (x, y) cu $r(x, y) = r(\overset{\circ}{L})$). Dacă ciclul $\overset{\circ}{L}$ conţine un singur arc de blocare (x'', y''), arcul de ieşire va fi acest arc. Dacă ciclul $\overset{\circ}{L}$ conţine mai multe arce de blocare, atunci următorul arbore parţial va fi degenerat (conţine arce (x, y) cu $f(x, y) = 0$ sau $f(x, y) = c(x, y)$). În acest caz algoritmul selectează arcul de ieşire (x'', y'') conform regulii următoare:

- se va selecta arcul de ieşire (x'', y'') ca fiind ultimul arc de blocare întâlnit la parcurgerea ciclului $\overset{\circ}{L}$ în sensul orientării sale, plecând de la nodul pisc z.

Fie $\overset{\circ}{L} = (z, \ldots, x'', y'', \ldots, z)$, $L_1 = (z, \ldots, x'')$, $L_2 = (y'', \ldots, z)$. Dacă considerăm $\overset{\circ}{L}, L_1, L_2$ ca secvenţe de arce atunci $\overset{\circ}{L} = L_1 \cup \{(x'', y'')\} \cup L_2$.

Teorema 11.13(a). *De la fiecare nod x din L_1 se poate transporta o cantitate strict pozitivă de flux la nodul rădăcină în următorul arbore parţial.*

(b). *De la fiecare nod x din L_2 se poate transporta o cantitate strict pozitivă de flux la nodul rădăcină în următorul arbore parţial.*

Demonstraţie. **(a)** Considerăm două cazuri:

(a_1) $r(\overset{\circ}{L}) > 0$; când se efectuează schimbarea de flux pe $\overset{\circ}{L}$ se trimit $r(\overset{\circ}{L})$ unităţi de flux pe L_1 de la z la x'' şi deci cel puţin $r(\overset{\circ}{L})$ unităţi de flux se pot trimite de la x'' la z. Deoarece arborele parţial a fost tare admisibil se pot trimite $\delta > 0$ unităţi de flux de la z la nodul rădăcină r. Rezultă că cel puţin $\delta' = \min\{r(\overset{\circ}{L}), \delta\}$ unităţi de flux se pot trimite de la fiecare nod x din L_1 la nodul rădăcină în următorul arbore parţial.

(a_2) $r(\overset{\circ}{L}) = 0$; în acest caz $x'' = x'$, deoarece proprietatea de tare admisibilitate a arborelui parţial anterior implică faptul că de la fiecare nod al lanţului de la nodul y' la nodul z se poate trimite o cantitate strict pozitivă de flux la nodul rădăcină r şi astfel nici un arc al acestui lanţ nu poate fi un arc de blocare, deoarece $r(\overset{\circ}{L}) = 0$. De asemenea, înainte de schimbarea fluxului, de la fiecare nod din L_1 se poate trimite o cantitate strict pozitivă de flux la nodul

rădăcină r, afirmaţie care rămâne valabilă şi după schimbarea de flux, deoarece $r(\overset{\circ}{L}) = 0$.

(b) Deoarece (x'', y'') este ultimul arc de blocare din $\overset{\circ}{L}$ rezultă că nici un arc din L_2 nu este arc de blocare şi deci de la fiecare nod y din L_2 se poate trimite o cantitate strict pozitivă de flux la nodul rădăcină r via nodul pisc z. ■

Teorema 11.14. *Noul arbore parţial* $G'' = (N, A'')$, *obţinut din arborele parţial tare admisibil* $G' = (N, A')$ *prin* $A'' = A' \cup \{(x', y')\} - \{(x'', y'')\}$, *unde arcul de ieşire* (x'', y'') *este determinat cu regula de mai sus, este tare admisibil.*

Demonstraţie. Considerăm două cazuri:

(a) $x \in \overset{\circ}{L}$; în acest caz, conform teoremei anterioare, de la fiecare nod x al lui $\overset{\circ}{L}$ se poate trimite o cantitate de flux strict pozitivă la nodul rădăcină r.

(b) $x \in N - \overset{\circ}{L}$; în acest caz, conform proprietăţii de tare admisibilitate a arborelui parţial anterior, de la fiecare nod x din $N - \overset{\circ}{L}$ se poate trimite o cantitate de flux strict pozitivă la nodul rădăcină r.

Deci de la fiecare nod x din G'' se poate trimite o cantitate de flux strict pozitivă la nodul rădăcină r, ceea ce implică faptul că G'' este un arbore parţial tare admisibil. ■

Exemplul 11.9. Fie reţeaua $\tilde{G}' = \left(N, \tilde{A}'\right)$ reprezentată în figura 11.9. Digraful $G' = (N, A')$ cu $A' = \tilde{A}' - \{(9, 10)\}$ este un arbore parţial tare admisibil, deoarece de la orice nod x din G' se poate transporta o cantitate $\delta > 0$ de flux la nodul rădăcină $r = 1$. Fie arcul $(x', y') = (9, 10)$ arcul de intrare. Acest arc creează în $\tilde{G}' = \left(N, \tilde{A}'\right)$ ciclul pivot $\overset{\circ}{L} = (2, 3, 5, 7, 9, 10, 8, 6, 4, 2)$ şi nodul 2 este nodul pisc. Deoarece $f(9, 10) = 0$ definim orientarea ciclului $\overset{\circ}{L}$ aceeaşi cu a arcului $(9, 10)$. Acest ciclu pivot este degenerat deoarece arcele $(2, 3)$ şi $(7, 5)$ blochează orice flux adiţional în ciclul pivot $\overset{\circ}{L}$. Parcurgând ciclul $\overset{\circ}{L}$ în sensul orientării sale, plecând de la nodul pisc $z = 2$, întâlnim arcul $(7, 5)$ după arcul $(2, 3)$; astfel selectăm arcul $(x'', y'') = (7, 5)$ ca arc de ieşire. Lanţurile L_1 şi L_2 sunt $L_1 = (2, 3, 5)$, $L_2 = (7, 9, 10, 8, 6, 4, 2)$.

Fig.11.9

Teorema 11.15. *Algoritmul simplex pentru problema fluxului de cost minim converge într-un număr finit de iteraţii.*

Demonstraţie. Deoarece fiecare iteraţie nedegenerată descreşte strict valoarea funcţiei obiectiv, rezultă că numărul iteraţiilor nedegenerate este finit. Algoritmul poate executa iteraţii degenerate. Vom arăta că numărul de iteraţii degenerate între oricare două iteraţii nedegenerate este finit. Considerăm două cazuri:

(c_1) $f(x', y') = 0$; în acest caz orientarea ciclului $\overset{\circ}{L}$ este aceeaşi cu a

arcului de intrare (x', y'). Arcul de ieşire (x'', y'') aparţine lanţului arbore de la nodul x' la nodul pisc z. Din prezentarea PROCEDURII POTENŢIAL2 rezultă că nodul x' se află în subarborele G_2 şi potenţialele tuturor nodurilor din G_2 se schimbă cu cantitatea $b^p(x', y')$. Deoarece $b^p(x', y') < 0$, această iteraţie degenerată descreşte strict suma tuturor potenţialelor nod, care prin ipoteza noastră anterioară este întreagă. Deoarece nici un potenţial nod nu poate scădea sub $-n\bar{b}$, rezultă că numărul succesiv de iteraţii degenerate este finit.

(c_2) $f(x', y') = c(x', y')$; în acest caz orientarea ciclului $\overset{\circ}{L}$ este definită ca fiind opusă orientării arcului de intrare (x', y'). Criteriul de selecţie a arcului de ieşire (x'', y'') rămâne neschimbat, adică regula prezentată mai sus. În acest caz nodul y' este conţinut în subarborele G_2 şi astfel după iteraţie, potenţialele tuturor nodurilor din G_2 descresc cu cantitatea $b^p(x', y') > 0$; în consecinţă, iteraţia descreşte strict suma tuturor potenţialelor nod. ■

11.2 Algoritmul simplex pentru fluxul maxim

Problema fluxului de cost minim (11.1) poate avea mai multe noduri sursă x $(v(x) > 0)$ şi mai multe noduri stoc x $(v(x) < 0)$. Fără a restrânge generalitatea putem presupune că problema are un singur nod sursă s şi un singur nod t.

Problema fluxului maxim se poate obţine ca o versiune particulară a problemei fluxului de cost minim. Se construieşte reţeaua $\tilde{G} = \left(N, \tilde{A}\right)$ cu $\tilde{A} = A \cup \{(t, s)\}$, $\tilde{c}(x, y) = c(x, y)$, $\tilde{b}(x, y) = 0$, $(x, y) \in A$, $\tilde{c}(t, s) = \infty$, $\tilde{b}(t, s) = -1$. Introducerea arcului (t, s) trimite fluxul de la nodul stoc t înapoi la nodul sursă s, obţinându-se astfel o circulaţie ($v(x) = 0$ pentru orice nod $x \in N$). Maximizarea valorii fluxului de la nodul sursă s la nodul stoc t este deci echivalentă cu maximizarea fluxului pe arcul (t, s). Dacă v este valoarea fluxului de la s la t, atunci $\tilde{f}(t, s) = v$. Deoarece problema fluxului de cost minim este o problemă de minim, în problema fluxului maxim în loc să maximizăm pe v vom minimiza pe $-v$. Astfel obţinem problema fluxului maxim sub forma

$$\text{Minimizează } (-\tilde{f}(t, s)) \tag{11.5.a}$$

$$\sum_y \tilde{f}(x, y) - \sum_y \tilde{f}(y, x) = 0, \ x \in N, \tag{11.5.b}$$

$$0 \leq \tilde{f}(x, y) \leq \tilde{c}(x, y), \ (x, y) \in \tilde{A}. \tag{11.5.c}$$

Algoritmul simplex pentru problema fluxului maxim este similar cu algoritmul simplex pentru problema fluxului de cost minim: se determină o structură arbore

parţial admisibilă iniţială; se testează optimalitatea structurii şi dacă nu este optimă se îmbunătăţeşte structura.

Pentru problema fluxului maxim în orice structură arbore parţial admisibilă arcul (t,s) aparţine arborelui parţial $\tilde{G}' = \left(N, \tilde{A}'\right)$ deoarece $0 < \tilde{f}(t,s) < \tilde{c}(t,s)$. Acest arc partiţionează mulţimea nodurilor N în două submulţimi N_s care conţine nodul sursă şi N_t care conţine nodul stoc t. Deci arcul (t,s) uneşte doi subarbori $\tilde{G}'_s = \left(N_s, \tilde{A}'_s\right)$ şi $\tilde{G}'_t = \left(N_t, \tilde{A}'_t\right)$ aşa cum se prezintă în figura 11.10.

Fig.11.10

Vectorul potenţial nod p corespunzător unui arbore parţial admisibil al problemei fluxului maxim se obţine în modul următor. Deoarece putem stabili un potenţial nod arbitrar, considerăm $p(t) = 0$. Având în vedere condiţiile de optimalitate:

$$\tilde{b}^p(x,y) = 0 \text{ pentru toate arcele } (x,y) \in \tilde{A}' \tag{11.6.a}$$

$$\tilde{b}^p(x,y) \geq 0 \text{ pentru toate arcele } (x,y) \in \overline{\tilde{A}}_0{}' \tag{11.6.b}$$

$$\tilde{b}^p(x,y) \leq 0 \text{ pentru toate arcele } (x,y) \in \overline{\tilde{A}}_c{}' \tag{11.6.c}$$

şi ţinând cont că $\tilde{b}(x,y) = 0,\ (x,y) \in \tilde{A}',\ \tilde{b}(t,s) = -1$ vom obţine:

$$p(s) = 1,\ p(x) = 1,\ x \in N_s,\ p(x) = 0,\ x \in N_t.$$

Fiecare soluţie arbore parţial a problemei fluxului maxim defineşte o tăietură $s-t$, $[N_s, N_t] = (N_s, N_t) \cup (N_t, N_s)$ în reţeaua $G = (N, A)$. Fiecare arc din această tăietură este un arc nonarbore. Conform cu potenţialele nod obţinute mai sus rezultă că $\tilde{b}^p(x,y) = -1,\ (x,y) \in (N_s, N_t),\ \tilde{b}^p(x,y) = 1,\ (x,y) \in (N_t, N_s)$ şi $\tilde{b}^p(x,y) = 0,\ (x,y) \notin [N_s, N_t]$. În consecinţă, pentru ca să fie verificate condiţiile 11.6 trebuie să avem $f(x,y) = c(x,y),\ (x,y) \in (N_s, N_t)$ şi $f(x,y) = 0,\ (x,y) \in (N_t, N_s)$ în reţeaua originară $G = (N, A)$. Această observaţie demonstrează încă o dată teorema fluxului maxim şi a tăieturii minime, deoarece algoritmul va determina o structură optimă atunci când fiecare arc direct al tăieturii $s - t$ $[N_s, N_t]$ are fluxul egal cu capacitatea şi fiecare arc invers are fluxul egal cu zero. Fie mulţimea de arce

$$\overline{\tilde{A}}{}'' = \{(N_s, N_t) \cap \overline{\tilde{A}}_0{}'\} \cup \{(N_t, N_s) \cap \overline{\tilde{A}}_c{}'\}.$$

Evident că arcul de intrare (x', y') este orice arc din $\overline{\tilde{A}}{}''$. Introducând arcul de intrare (x', y') în arborele parţial $\tilde{G}' = \left(N, \tilde{A}'\right)$ se creează un ciclu $\overset{\circ}{\tilde{L}}$ care conţine

arcul (t, s) ca un arc direct. Algoritmul va trimite pe acest lanţ o cantitate maximă de flux şi va identifica un arc de blocare (x'', y'') care este arcul de ieşire. Astfel se creează un nou arbore parţial $\tilde{G}' = \left(N, \tilde{A}'\right)$ care conţine doi subarbori $\tilde{G}'_s$ şi $\tilde{G}'_t$ uniţi prin arcul (t, s).

Remarcăm faptul că algoritmul simplex pentru fluxul maxim este un algoritm cu lanţ de mărire a fluxului. Structura arbore permite determinarea lanţului de mărire a fluxului de la nodul sursă s la nodul stoc t foarte uşor. În acest sens algoritmul simplex are un avantaj faţă de alţi algoritmi cu lanţ de mărire a fluxului pentru problema fluxului maxim. Totuşi, datorită degenerării, algoritmul simplex poate să nu trimită o cantitate strict pozitivă de flux de la nodul sursă s la nodul stoc t în fiecare iteraţie.

Pentru evitarea degenerării şi ciclării algoritmului simplex pentru problema fluxului maxim se vor determina structurile arbore parţial tare admisibile ca la algoritmul simplex pentru problema fluxului de cost minim. Se consideră nodul rădăcină r ca fiind nodul stoc t. Conform definiţiei arborelui parţial tare admisibil se poate trimite flux de la fiecare nod din subarborele $\tilde{G}'_t$ la nodul rădăcină $r = t$ fără să încălcăm restricţiile de mărginire a fluxului. În consecinţă fiecare arc $(x, y) \in \tilde{A}'_t$ cu $\tilde{f}(x, y) = 0$ are orientarea către nodul t şi fiecare arc $(x, y) \in \tilde{A}'_t$ cu $\tilde{f}(x, y) = \tilde{c}(x, y)$ nu are orientarea către nodul t. În acest caz, regula arcului de ieşire constă în selectarea arcului de blocare din ciclul pivot care este cel mai depărtat de nodul stoc t atunci când parcurgem ciclul în direcţia arcului (t, s) plecând de la nodul stoc t. Remarcăm faptul că fiecare iteraţie degenerată selectează un arc de intrare incident unui nod din $\tilde{G}'_t$ şi conform precizărilor de mai sus fiecare arc de blocare trebuie să fie din $\tilde{G}'_s$. Ca urmare fiecare iteraţie degenerată măreşte pe $\tilde{G}'_t$, astfel algoritmul poate executa cel mult n iteraţii degenerate consecutive. Precizăm că problema fluxului de cost minim nu satisface această proprietate.

Teorema 11.16. *Algoritmul simplex pentru problema fluxului maxim determină un flux maxim de la nodul sursă s la nodul stoc t într-o reţea $G = (N, A, c)$.*

Demonstraţie. Afirmaţia teoremei rezultă din corectitudinea algoritmului simplex pentru fluxul de cost minim şi precizările făcute în acest paragraf.

Teorema 11.17. *Algoritmul simplex pentru problema fluxului maxim are complexitatea $O(n^2\,\overline{c})$.*

Demonstraţie. Algoritmul poate executa cel mult $n\,\overline{c}$ iteraţii nedegenerate, deoarece $n\,\overline{c}$ este o margine superioară pentru valoarea fluxului maxim. Între două iteraţii nedegenerate consecutive algoritmul poate executa cel mult n iteraţii degenerate. Deci algoritmul are complexitatea $O(n^2\,\overline{c})$. ■

Conform teoremei de complexitate rezultă că algoritmul simplex pentru problema fluxului maxim este pseudopolinomial.

În continuare se va prezenta un algoritm simplex polinomial pentru problema

fluxului maxim. Vom utiliza următoarele notaţii pentru iteraţia k a acestui algoritm:

$\tilde{f}_k$ fluxul obţinut până la această iteraţie;
$\tilde{G}'_k = \left(N, \tilde{A}'_k\right)$ arborele parţial corespunzător iteraţiei;
$[N^k_s, N^k_t] = (N^k_s, N^k_t) \cup (N^k_t, N^k_s)$ tăietura generată de $\tilde{G}'_k$ şi arcul (t, s);
$\overline{\tilde{A}}''_k$ mulţimea de arce din care se selectează arcul de intrare $a'_k = (x', y')$;
$\overset{\circ}{\tilde{L}}_k$ ciclul generat de arcul de intrare a'_k;
$a''_k = (x'', y'')$ arcul de ieşire din $\overset{\circ}{\tilde{L}}_k$.

Definiţia 11.13. Se numeşte *pseudolanţ de mărire a fluxului* relativ la $\tilde{f}_k$ şi $\tilde{G}'_k$, lanţul $P_k(u, z)$ de la nodul u la nodul z din G astfel încât dacă $(x, y) \in P_k(u, z)$ şi $(x, y) \notin \tilde{A}'_k$, atunci $\tilde{f}_k(x, y) = 0$ dacă (x, y) este arc direct în $P_k(u, z)$ şi $\tilde{f}_k(x, y) = \tilde{c}(x, y)$ dacă (x, y) este arc invers în $P_k(u, z)$.

Numărul de arce ale pseudolanţului de mărire a fluxului $P_k(u, z)$ reprezintă lungimea acestui pseudolanţ şi o notăm cu $\ell_k(u, z)$. Se va nota $P^*_k(u, z)$ pseudolanţul de mărire a fluxului cu lungimea minimă şi cu $\ell^*_k(u, z)$ lungimea acestuia. Dacă nu există pseudolanţ de mărire a fluxului $P_k(u, z)$ atunci prin definiţie se consideră $\ell_k(u, z) = \infty$.

Definiţia 11.14. Se numeşte *eticheta* nodului z valoarea $d_k(z) = \ell^*_k(s, z)$.

Etichetele $d_k(z),\ z \in N$ se pot calcula cu o parcurgere BF (breadth first) în reţeaua G, de-a lungul pseudolanţurilor de mărire a fluxului $P_k(s, z)$, plecând de la nodul sursă s.

Definiţia 11.15. Se numeşte *eticheta arcului nonarbore* $(x, y),\ (x, y) \notin \tilde{A}'_k$, valoarea $e_k(x, y) = d_k(x)$ dacă $\tilde{f}_k(x, y) = 0$ şi $e_k(x, y) = d_k(y)$ dacă $\tilde{f}_k(x, y) = \tilde{c}(x, y)$. Se numeşte *eticheta arcului arbore* $(x, y),\ (x, y) \in \tilde{A}'_k$, valoarea $e_k(x, y) = 0$ dacă $(x, y) \in \tilde{A}'_i,\ i = 1, \ldots, k-1$ şi $e_k(x, y) = e_{k-1}(x, y)$, dacă există $i \in \{1, \ldots, k-1\}$ astfel încât $(x, y) \notin \tilde{A}'_i$.

Definiţia 11.16. Se numeşte *arc admisibil* relativ la $\tilde{f}_k$ şi $\tilde{G}'_k$ un arc (x, y) care verifică condiţiile:

$$d_k(y) = d_k(x) + 1 \ \text{ dacă } \tilde{f}_k(x, y) = 0 \tag{11.7.a}$$

$$d_k(x) = d_k(y) + 1 \ \text{ dacă } \tilde{f}_k(x, y) = \tilde{c}(x, y). \tag{11.7.b}$$

Dacă arcul de intrare a'_k se selectează din $\overline{\tilde{A}}''_k$ după regula că trebuie să fie admisibil şi $e_k(a'_k) < e_k(a''_{k-1})$, atunci se obţine algoritmul simplex cu eticheta mai mică pentru problema fluxului maxim (ASEM). Acest algoritm este prezentat mai jos.

(1) PROGRAM ASEM;
(2) BEGIN
(3) $\tilde{f}_1 := 0$;
(4) se determină un arbore parţial $\tilde{G}'_1 = \left(N, \tilde{A}'_1\right)$, N_s^1, N_t^1, $\overline{\tilde{A}}_1''$;
(5) se calculează d_1;
(6) se selectează un nod y'' din N_t^1 cu $d_1(y'') = \min\{d_1(z) \mid z \in N_t^1\}$;
(7) $k := 1$;
(8) WHILE $d_k(t) < \infty$ DO
(9) BEGIN
(10) se selectează un arc admisibil a'_k din $\overline{\tilde{A}}''$ cu $e_k(a'_k) \leq d_k(y'') - 1$;
(11) se determină ciclul $\overset{\circ}{\tilde{L}}$ creat de arcul de intrare a'_k;
(12) se măreşte fluxul de-a lungul ciclului $\overset{\circ}{\tilde{L}}$ şi se obţine $\tilde{f}_{k+1}$;
(13) se determină arcul de ieşire a''_k;
(14) se determină $\tilde{A}'_{k+1}$, N_s^{k+1}, N_t^{k+1}, $\overline{\tilde{A}}''_{k+1}$;
(15) se precizează nodul y'' extremitatea din N_t^{k+1} a arcului a''_k;
(16) se calculează d_{k+1};
(17) $k := k + 1$;
(18) END;
(19) $\tilde{f}_k$ este fluxul maxim şi $[N_s^k, N_t^k]$ este tăietura minimă;
(20) END.

Fie arcul de ieşire $a''_{k-1} = (x'', y'')$ la iteraţia $k-1$. Dacă arcul (x'', y'') este arc direct al ciclului $\overset{\circ}{\tilde{L}}_{k-1}$, atunci $\tilde{f}_k(x'', y'') = \tilde{c}(x'', y'')$ şi $(x'', y'') \in (N_s^k, N_t^k)$. Deci $e_k(x'', y'') = d_k(y'')$ cu $y'' \in N_t^k$. Dacă arcul (x'', y'') este arc invers al ciclului $\overset{\circ}{\tilde{L}}_{k-1}$, atunci $\tilde{f}_k(x'', y'') = 0$ şi $(x'', y'') \in (N_t^k, N_s^k)$. Deci $e_k(x'', y'') = d_k(x'')$ cu $x'' \in N_t^k$. Rezultă faptul că inegalitatea $e_k(a'_k) < e_k(a''_{k-1})$, prezentată mai sus, este echivalentă cu inegalitatea $e_k(a'_k) \leq d_k(y'') - 1$ cu $y'' \in N_t^k$, care apare în linia (10) a algoritmului.

Exemplul 11.11. Considerăm reţeaua $\tilde{G} = \left(N, \tilde{A}, \tilde{c}\right)$ reprezentată în figura 11.11(a) cu nodul sursă $s = 1$, nodul stoc $t = 6$ şi fluxul iniţial $\tilde{f}_1 = 0$.
Iniţializări:
Arborele parţial iniţial $\tilde{G}_1'$ cu $\tilde{A}_1' = \{(1,2),(1,3),(4,6),(5,6),(6,1)\}$, $N_1^1 = \{1,2,3\}$, $N_6^1 = \{4,5,6\}$ este reprezentat în figura 11.11(a'); $\overline{\tilde{A}}_1'' = \{(2,4),(3,5)\}$, $d_1 = (0,1,1,2,2,3)$, $\min\{d_1(4), d_1(5), d_1(6)\} = \min\{2,2,3\} = 2$ şi selectăm $y'' = 4$.

Fig.11.11

$k = 1$
$d_1(6) < \infty$ şi se execută instrucţiunea WHILE
$d_1(4) = d_1(2) + 1, d_1(5) = d_1(3) + 1$ şi ambele arce din $\overline{\tilde{A}}_1''$ sunt admisibile; $e_1(2,4) = d_1(2) = 1, e_1(2,4) \leq d_1(4) - 1$ şi $e_1(3,5) = d_1(3) = 1, e_1(3,5) \leq d_1(4) - 1$; deci ambele arce din $\overline{\tilde{A}}_1''$ verifică condiţia din linia (10) şi selectăm arcul $a_1' = (2,4)$; $\overset{\circ}{\tilde{L}}_1 = (2,4,6,1,2), r(\overset{\circ}{\tilde{L}}_1) = \min\{6,4,\infty,10\} = 4$; se măreşte fluxul de-a lungul ciclului $\overset{\circ}{\tilde{L}}_1$ şi se obţine fluxul $\tilde{f}_2$ din reţeaua reprezentată în figura 11.11(b); singurul arc de blocare este arcul (4,6) şi deci arcul de ieşire este $a_1'' = (4,6)$; se obţine arborele parţial $\tilde{G}_2'$ cu $\tilde{A}_2' = \{(1,2),(1,3),(2,4),(5,6),(6,1)\}$, $N_1^2 = \{1,2,3,4\}, N_6^2 = \{5,6\}$ reprezentat în figura 11.11(b'); $\overline{\tilde{A}}_2'' = \{(3,5),(4,5)\}$, $y'' = 6$, $d_2 = (0,1,1,2,2,3)$.
$k = 2$
$d_2(6) < \infty$ şi se execută instrucţiunea WHILE
$d_2(5) = d_2(3) + 1, d_2(5) \neq d_2(4) + 1$; arcul (3,5) este singurul arc admisibil din $\overline{\tilde{A}}_2''$; $e_2(3,5) = d_2(3) = 1, e_2(3,5) \leq d_2(6) - 1$ şi deci $a_2' = (3,5)$; $\overset{\circ}{\tilde{L}}_2 = (3,5,6,1,3), r(\overset{\circ}{\tilde{L}}_2) = 2$; se execută mărirea de flux şi se obţine fluxul $\tilde{f}_3$ din reţeaua reprezentată în figura 11.11(c); $a_2'' = (1,3)$; se obţine arborele parţial $\tilde{G}_3'$ cu $\tilde{A}_3' = \{(1,2),(2,4),(3,5),(5,6),(6,1)\}$, $N_1^3 = \{1,2,4\}, N_6^3 = \{3,5,6\}$ reprezentat în figura 11.11(c'); $\overline{\tilde{A}}_3'' = \{(4,3),(4,5)\}$, $y'' = 3$, $d_3 = (0,1,3,2,3,4)$.
$k = 3$
$d_3(6) < \infty$ şi se execută instrucţiunea WHILE
$d_3(3) = d_3(4) + 1, d_3(5) = d_3(4) + 1$ şi ambele arce $\overline{\tilde{A}}_3''$ sunt admisibile; $e_3(4,3) = d_3(4) = 2, e_3(4,3) \leq d_3(3) - 1, e_3(4,5) = d_3(4) = 2, e_3(4,5) \leq d_3(3) - 1$; se selectează arcul $a_3' = (4,5)$; $\overset{\circ}{\tilde{L}}_3 = (4,5,6,1,2,4), r(\overset{\circ}{\tilde{L}}_3) = 2$; se obţine fluxul $\tilde{f}_4$ din reţeaua reprezentată în figura 11.11(d); $a_3'' = (2,4)$ şi se obţine arborele parţial $\tilde{G}_4'$ cu $\tilde{A}_4' = \{(1,2),(3,5),(4,5),(5,6),(6,1)\}$, $N_1^4 = \{1,2\}, N_6^4 = \{3,4,5,6\}$ reprezentat în figura 11.11(d'); $\overline{\tilde{A}}_4'' = \emptyset, y'' = 4$, $d_4 = (0,1,\infty,\infty,\infty,\infty)$.

$k = 4$
$d_4(6) = \infty$, STOP; $\tilde{f}_4$ este fluxul maxim cu $v = \tilde{f}_4(6,1) = 8$ şi $[N_1^4, N_6^4] = [\{1,2\},\{3,4,5,6\}]$ este o tăietură minimă; într-adevăr, avem $c[\{1,2\},\{3,4,5,6\}] = c(\{1,2\},\{3,4,5,6\}) = c(1,3) + c(2,4) = 2 + 6 = 8 = v$.

Teorema 11.18. *Algoritmul simplex cu eticheta mai mare pentru problema fluxului maxim determină un flux maxim de la nodul sursă s la nodul stoc t într-o reţea $G = (N, A, c)$.*

Demonstraţie. Afirmaţia teoremei rezultă din Teorema 11.16 şi precizările de mai sus. ■

Teorema 11.19. *Algoritmul simplex cu eticheta mai mare pentru problema fluxului maxim are complexitatea $O(n^2m)$.*

Demonstraţie. Pentru demonstraţie se recomandă cititorului monografiile prezentate la bibliografie. ■

Dacă arcul de intrare a'_k se selectează din $\overline{\tilde{A}}''$ după regula că trebuie să fie admisibil şi $e_k(a'_k) = \min\{e_k(a_k) \mid a_k \in \overline{\tilde{A}}''\}$ (este cel mai apropiat de nodul sursă s) atunci obţinem algoritmul simplex cu eticheta cea mai mică pentru problema fluxului maxim (ASEM2).

11.3 Aplicaţii şi comentarii bibliografice

11.3.1 Repartizarea proiectelor la studenţi

Fiecare student trebuie să participe la un proiect al unei firme. Pentru a face o repartizare corectă a proiectelor ţinând cont de preferinţele studenţilor, profesorul responsabil cu proiectele propune următoarea abordare: fiecare student îşi numerotează proiectele în ordinea crescătoare a preferinţelor sale. Obiectivul acestei probleme de repartizare este ca suma numerelor asociate proiectelor repartizate să fie maximă. Problema repartizării proiectelor la studenţi are două constrângeri: fiecare student trebuie să lucreze mai exact un proiect şi pentru fiecare proiect există un număr maxim de studenţi care pot participa la el.

Problema repartizării a NP proiecte la NS studenţi se poate modela ca o problemă de flux de cost minim în reţeaua $G = (N, A, c, b)$ construită în modul următor: $N = N_1 \cup N_2 \cup \{t\}$, $N_1 = \{i \mid i = 1, \ldots, NS\}$, $N_2 = \{j' \mid j' = 1, \ldots, NP\}$, $A = A_1 \cup A_2$, $A_1 = \{(i, j') \mid i \in N_1,\ j' \in N_2\}$, $A_2 = \{(j', t) \mid j \in N_2\}$, $c(i, j') = 1$, $(i, j') \in A_1$, $c(j', t) =$ numărul maxim de studenţi care pot lucra la proiectul j, $(j', t) \in A_2$, $b(i, j') =$ opusul numărului acordat de studentul i proiectului j, $(i, j') \in A_1$, $b(j', t) = 0$, $(j', t) \in A_2$, $v(i) = 1$, $i \in N_1$, $v(j') = 0$, $j' \in N_2$, $v(t) = -NS$. Apoi se determină un flux de cost minim în reţeaua G. Evident, un flux de cost minim corespunde unei repartizări optime a proiectelor la studenţi.

Exemplul 11.12. Considerăm că avem $NP = 3$ proiecte şi $NS = 4$ studenţi care şi-au exprimat preferinţele pentru proiecte în felul următor:

Student\ Proiect	1	2	3
1	1	3	2
2	1	2	3
3	2	3	1
4	1	3	2

La proiectul 1 pot lucra maxim 3 studenţi, iar la proiectele 2 şi 3 pot lucra maxim 2 studenţi.

Reţeaua G corespunzătoare acestei probleme este reprezentată în figura 11.12.

Fig.11.12
Fig.11.13

Fluxul de cost minim din reţeaua G este reprezentat în figura 11.13.

Repartizarea optimă a proiectelor corespunzătoare fluxului de cost minim din figura 11.13 este următoarea: studenţii 1 şi 3 lucrează la proiectul 2, iar studenţii 2 şi 4 la proiectul 3.

11.3.2 Calcul distribuit pe două procesoare

Această aplicaţie constă în repatizarea modulelor unui program la două procesoare astfel încât costul total al comunicaţiei între procesoare şi al calculului să fie minim. Considerăm un calculator cu două procesoare, nu neapărat identice şi un program complex care trebuie executat pe acest calculator. Programul conţine mai multe module care interacţionează între ele pe parcursul execuţiei lui. Se cunoaşte costul execuţiei fiecărui modul pe fiecare procesor. Fie $p_1(i)$, respectiv $p_2(i)$ costul procesării modulului i de către procesorul 1, respectiv 2. Repartizarea diferitelor module la diferite procesoare implică unele costuri suplimentare cauzate de comunicaţia între procesoare. Fie c_{ij} costul comunicaţiei între procesoare în cazul în care modulele i şi j sunt repartizate la procesoare diferite. Problema calculului distribuit pe două procesoare constă în repartizarea modulelor programului la cele două procesoare astfel încât să se minimizeze costul total al procesării şi al comunicaţiei între procesoare.

Formulăm această problemă ca o problemă de determinare a unei tăieturi minime într-o reţea neorientată. Considerăm că programul conţine n module. Construim reţeaua $G = (N, A, c, s, t)$, $N = N_1 \cup \{s, t\}$, $N_1 = \{1, 2, \ldots, n\}$, $A = A_1 \cup A_2 \cup A_3$, $A_1 = \{(s, i) \mid i \in N_1\}$, $A_2 = \{(i, j) \mid$ modulul i interacţionează cu modulul j pe parcursul execuţiei programului $\}$, $A_3 = \{(i, t)\ i \in N_1\}$, $c(s, i) = p_2(i)$, $(s, i) \in A_1$, $c(i, j) = c_{ij}$, $(i, j) \in A_2$, $c(i, t) = p_1(i)$, $(i, t) \in A_3$.

În figura 11.14(a) sunt ilustrate costurile procesării celor 4 module ale unui program de către cele două procesoare, iar în figura 11.14(b) sunt prezentate costurile comunicaţiei între programe.

(a) (b)

Figura 11.14

În figura 11.15 este reprezentată reţeaua G.

Fig.11.15

Observăm că există o corespondenţă biunivocă între tăieturile $s-t$ din reţeaua G şi repartizările modulelor la cele două procesoare. Mai mult, capacitatea unei tăieturi este egală cu costul repartizării modulelor la cele două procesoare corespunzătoare tăieturii. Fie $N' = \{i \in N_1 \mid \text{modulul } i \text{ a fost repartizat procesorului } 1\}$ şi $N'' = \{i \in N_1 \mid \text{modulul } i \text{ a fost repartizat procesorului } 2\}$. Costul acestei repartizări este $\sum_{i \in N'} p_1(i) + \sum_{i \in N''} p_2(i) + \sum_{(i,j) \in (N', N'') \cap A} c(i,j)$.

Tăietura $s-t$ care corespunde acestei repartizări este $[\{s\} \cup N',\ \{t\} \cup N'']$. În consecinţă, o tăietură $s-t$ minimă în reţeaua G va furniza o repartizare optimă a modulelor programului la cele două procesoare.

În figura 11.15 este reprezentată tăietura $s-t$ minimă $[\{s,1,2\},\{3,4,t\}]$ al cărei cost este 30. Repartizarea optimă corespunzătoare tăieturii $s-t$ minime este: procesorul 1 execută modulele 1 şi 2, iar procesorul 2 execută modulele 3 şi 4. Costul acestei repartizări este egal cu costul tăieturii $s-t$ minime corespunzătoare, adică 30.

11.4 Comentarii bibliografice

Pornind de la metoda simplex de programare liniară, Dantzig (1951) a dezvoltat primul algoritm simplex pentru rezolvarea unei probleme de transport într-o reţea cu capacităţile arcelor infinite. Mai târziu, Dantzig (1962) a obţinut algoritmul simplex pentru fluxul de cost minim. În anii care au urmat, au fost dezvoltaţi şi testaţi algoritmi simplex pentru problema fluxului de cost minim care folosesc structuri arbore parţial.

Algoritmul prezentat în paragraful 11.1 foloseşte o operaţie pivot care se datorează lui Mulvey (1978). Studiile au arătat că mai mult de 90 % dintre operaţiile pivot pot fi degenerate. Numărul acestora poate fi redus prin utilizarea structurilor arbore parţial tare admisibile care au fost descrise de Cunningham (1976).

Capitolul 12

Fluxuri cu multiplicatori şi fluxuri cu mai multe produse

12.1 Fluxuri cu multiplicatori

Fie $G = (N, A, c, b, \mu, v)$ o reţea orientată cu N mulţimea nodurilor, A mulţimea arcelor, $c : A \to \Re^+$ funcţia capacitate, $b : A \to \Re$ funcţia cost, $\mu : A \to \mathcal{Q}^+$ funcţia multiplicativă şi $v : N \to \Re$ funcţia valoare. Problema fluxului cu multiplicatori de cost minim constă în determinarea funcţiei flux $f : A \to \Re^+$ care verifică condiţiile:

$$\text{Minimizează } z(f) = \sum_A b(x,y)f(x,y), \tag{12.1.a}$$

$$\sum_y f(x,y) - \sum_y \mu(y,x)f(y,x) \;=\; v(x), \;\; x \in N, \tag{12.1.b}$$

$$0 \le f(x,y) \le c(x,y), \;\; (x,y) \in A \tag{12.1.c}$$

cu $\sum_N v(x) = 0$.

Această problemă se mai numeşte şi *problema fluxului generalizat*.

În această problemă $\mu(x,y)$ este multiplicatorul arcului (x,y) şi presupunem că dacă trimitem 1 unitate de flux de la nodul x la nodul y pe arcul (x,y), atunci $\mu(x,y)$ unităţi de flux sosesc la nodul y. Presupunem că fiecare multiplicator $\mu(x,y)$ este un număr raţional. Dacă $\mu(x,y) < 1$ atunci arcul (x,y) este *cu pierdere*; dacă $\mu(x,y) > 1$ atunci arcul (x,y) este *cu câştig*.

Fie L un lanţ de la nodul s la nodul t. Fie L^+ mulţimea arcelor directe şi L^- mulţimea arcelor inverse ale lanţului L. Definim multiplicatorul lanţului L, notat $\mu(L)$, prin

$$\mu(L) = \left(\prod_{L^+} \mu(x,y)\right) / \left(\prod_{L^-} \mu(x,y)\right) \tag{12.2}$$

Evident că dacă lanţul L este un drum D atunci $\mu(D) = \prod_D \mu(x,y)$.

Exemplul 12.1. Se consideră drumul şi lanţul din figura 12.1.

(a) (b)

Fig.12.1

Presupunem că din nodul 1 al drumului $D = (1, 2, 3, 4, 5)$ din figura 12.1(a) pleacă 2 unităţi de flux. În nodul 2 sosesc $3 \cdot 2 = 6$ unităţi de flux care pleacă la nodul 3 şi sosesc 3 unităţi de flux, în nodul 4 sosesc 12 unităţi de flux şi în nodul 5 sosesc 12 unităţi de flux. Deci dacă se trimit 2 unităţi de flux pe drumul D atunci în nodul 5 sosesc $2\mu(D) = 2 \cdot 6 = 12$ unităţi de flux.

Presupunem că din nodul 1 al lanţului $L = (1, 2, 3, 4)$ din figura 12.1.(b) pleacă 1 unitate de flux şi în nodul 2 ajung 2 unităţi de flux. Deoarece $2\,f(1,2) + 6\,f(3,2) = 0$ înseamnă că din nodul 3 trebuie să plece $-1/3$ unităţi de flux la nodul 2. Deci din nodul 3 trebuie să plece $1/3$ unităţi de flux la nodul 4 şi la nodul 4 sosesc $2/3$ unităţi de flux. Rezultă că dacă se trimite pe lanţul L 1 unitate de flux atunci în nodul 4 sosesc $1 \cdot \mu(L) = 1 \cdot 2/3 = 2/3$ unităţi de flux.

Din acest exemplu rezultă următoarele afirmaţii:

1. dacă se trimit r unităţi de flux pe un arc direct (x, y), atunci la nodul y sosesc $r\mu(x, y)$ unităţi de flux;

2. dacă se trimit r unităţi de flux din nodul x pe un arc invers (x, y), fluxul pe arc este $-r/\mu(x, y)$ unităţi de flux şi $r/\mu(x, y)$ unităţi de flux devin disponibile la nodul y.

Proprietatea următoare este o consecinţă a celor discutate mai sus.

Proprietatea 12.1. *Dacă se trimite 1 unitate de flux de la nodul s la nodul t de-a lungul lanţului L, atunci $\mu(L)$ unităţi de flux devin disponibile la nodul t.*

Fie $\overset{\circ}{L}$ un ciclu cu $\overset{\circ}{L}^+$ mulţimea arcelor directe şi $\overset{\circ}{L}^-$ mulţimea arcelor inverse ale acestui ciclu. În raport cu orientarea ciclului se defineşte multiplicatorul ciclului, notat cu $\mu(\overset{\circ}{L})$ prin

$$\mu\left(\overset{\circ}{L}\right) = \left(\prod_{\overset{\circ}{L}^+} \mu(x,y)\right) / \left(\prod_{\overset{\circ}{L}^-} \mu(x,y)\right) \tag{12.3}$$

Proprietatea 12.1 implică faptul că dacă se trimite 1 unitate de flux de-a lungul ciclului $\overset{\circ}{L}$ care pleacă din nodul s, atunci $\mu(\overset{\circ}{L})$ unităţi de flux ajung în nodul s. Dacă $\mu(\overset{\circ}{L}) > 1$, atunci se creează un exces la nodul s; în acest caz se spune că ciclul $\overset{\circ}{L}$ este un *ciclu cu câştig*. Dacă $\mu(\overset{\circ}{L}) < 1$, atunci se creează un deficit la nodul s; în acest caz se spune că ciclul $\overset{\circ}{L}$ este un *ciclu cu pierdere*. Dacă $\mu(\overset{\circ}{L}) = 1$, atunci fluxul de-a lungul acestui ciclu conservă fluxul la toate nodurile lui; în acest caz se spune că ciclul $\overset{\circ}{L}$ este un *ciclu de conservare*.

Sunt evidente următoarele proprietăţi:

Proprietatea 12.2. *Dacă $\mu(\overset{\circ}{L})$ este multiplicatorul ciclului $\overset{\circ}{L}$ cu orientarea specificată, atunci $1/\mu(\overset{\circ}{L})$ este multiplicatorul aceluiaşi ciclu $\overset{\circ}{L}$ cu orientarea opusă.*

Proprietatea 12.3. *Prin trimiterea a r unităţi de flux de-a lungul ciclului $\overset{\circ}{L}$ cu plecarea din nodul s, se creează un exces/deficit de $r\left(\mu(\overset{\circ}{L}) - 1\right)$ unităţi de flux în nodul s.*

Proprietatea 12.4. *Fie s un nod al unui ciclu $\overset{\circ}{L}$ cu $\mu(\overset{\circ}{L}) \neq 1$. Pentru a creea un exces/deficit de r unităţi de flux în nodul s (care verifică constrângerile de conservare a fluxului în toate celelalte noduri), trebuie trimise $r/\left(\mu(\overset{\circ}{L}) - 1\right)$ unităţi de flux de-a lungul ciclului $\overset{\circ}{L}$ cu plecarea din nodul s.*

Definiţia 12.1. Fie $G'_a = (N', A'_a)$ un subgraf al grafului $G = (N, A)$ astfel încât $N' \subseteq N$ şi $A'_a \subseteq A$. Se spune că G'_a este *un arbore extins* dacă $G' = (N', A')$, $A' = A'_a - \{a\}$, $a = (x', y')$ este un arbore.

Evident că arcul $a = (x', y') \in A'_a$ creează cu arcele din A' un ciclu unic $\overset{\circ}{L}{}'_a$ în subgraful G'_a.

Definiţia 12.2. Arcul $a = (x', y') \in A'_a$ care creează ciclul unic $\overset{\circ}{L}{}'_a$ în subgraful G'_a se numeşte *extraarc* şi ciclul creat $\overset{\circ}{L}{}'_a$ se numeşte *extraciclu.*

Remarcăm faptul că orice arc al ciclului $\overset{\circ}{L}{}'_a$ poate fi considerat extraarc. De asemenea, se consideră că oricare arbore extins G'_a are un nod special numit *nod rădăcină* şi că G'_a este agăţat de acest nod.

Definiţia 12.3. Se numeşte *pădure extinsă* a grafului $G = (N, A)$ un subgraf $G'_p = (N, P')$ cu $P' \subseteq A$ o colecţie de arbori extinşi disjuncţi.

Definiţia 12.4. Se numeşte *arbore extins bun* al grafului $G = (N, A)$ un arbore extins $G'_a = (N', A'_a)$ cu proprietatea că extraciclul $\overset{\circ}{L}{}'_a$ are multiplicatorul $\mu(\overset{\circ}{L}{}'_a) \neq 1$.

Definiţia 12.5. Se numeşte *pădure extinsă bună* a grafului $G = (N, A)$ o pădure extinsă $G'_p = (N, P')$ cu proprietatea că fiecare componentă este un arbore extins bun.

Definiţia 12.6. Arcele din mulţimea P' se numesc *arce pădure extinsă* şi arcele din $\overline{P}' = A - P'$ se numesc *arce nonpădure extinsă.*

Exemplul 12.2. În figura 12.2(b) se reprezintă un arbore extins $G'_a = (N', A'_a)$ şi în figura 12.2(c) se reprezintă o pădure $G'_p = (N, P')$ pentru graful reprezentat în figura 12.2(a). Extraarcele sunt reprezentate prin linii întrerupte.

O pădure extinsă se poate memora într-un calculator ca o colecţie de arbori extinşi. Fiecare arbore extins este un arbore plus un extraarc. Un arbore se poate memora prin utilizarea a trei tablouri unidimensionale fiecare cu n componente: tabloul predecesor π, tabloul adâncime η şi tabloul conductor ω. Aceste tablouri au fost definite în capitolul 11.

Fie o partiţie a mulţimii arcelor A de forma $A = \left(P', \overline{P}'_0, \overline{P}'_c\right)$ unde:

1. P' este mulţimea arcelor pădure extinsă bună;
2. $\overline{P}'_0$ este mulţimea arcelor nonpădure extinsă bună (x, y) pentru care $f(x, y) = 0$;
3. $\overline{P}'_c$ este mulţimea arcelor nonpădure extinsă bună (x, y) pentru care $f(x, y) = c(x, y)$.

Definiţia 12.7. Se numeşte *structură pădure extinsă* tripletul $\left(P', \overline{P}'_0, \overline{P}'_c\right)$.

Unei structuri pădure extinsă $\left(P', \overline{P}'_0, \overline{P}'_c\right)$ îi putem pune în corespondenţă o soluţie pădure extinsă în modul următor: $f(x, y) = 0$ pentru fiecare arc (x, y) din $\overline{P}'_0$, $f(x, y) = c(x, y)$ pentru fiecare arc (x, y) din $\overline{P}'_c$ şi $f(x, y)$ pentru fiecare arc (x, y) din P' se obţine rezolvând sistemul de restricţii de conservare a fluxului.

Definiţia 12.8. Se spune că structura pădure extinsă $\left(P', \overline{P}'_0, \overline{P}'_c\right)$ este *admisibilă* dacă soluţia pădure extinsă corespunzătoare este admisibilă; în caz contrar se spune că structura pădure extinsă $\left(P', \overline{P}'_0, \overline{P}'_c\right)$ este *inadmisibilă.*

Definiţia 12.9. Se spune că structura pădure extinsă $\left(P', \overline{P}'_0, \overline{P}'_c\right)$ este *nedegenerată* dacă toate arcele din P' sunt libere $(0 < f(x, y) < c(x, y))$; în caz contrar se spune că structura pădure extinsă $\left(P', \overline{P}'_0, \overline{P}'_c\right)$ este *degenerată.*

Fig.12.2

Definiţia 12.10. Se spune că structura pădure extinsă $\left(P', \overline{P}'_0, \overline{P}'_c\right)$ este *optimală* dacă soluţia pădure extinsă corespunzătoare este soluţie optimă a problemei fluxului cu multiplicatori de cost minim.

Pentru problema fluxului cu multiplicatori de cost minim costurile reduse se definesc în modul următor: $b^p(x, y) = b(x, y) - p(x) + \mu(x, y)\, p(y)$ pentru fiecare arc $(x, y) \in A$, unde $p(x)$ este potenţialul nodului x, $x \in N$.

Teorema 12.5. *Un flux f^* este o soluţie optimă a problemei fluxului cu multiplicatori de cost minim dacă este admisibilă şi pentru un vector potenţial p oarecare,*

perechea (f^*, p) *verifică următoarele condiţii de optimalitate:*

$$\text{Dacă } 0 < f^*(x,y) < c(x,y), \text{ atunci } b^p(x,y) = 0. \tag{12.4.a}$$

$$\text{Dacă } f^*(x,y) = 0, \text{ atunci } b^p(x,y) \geq 0. \tag{12.4.b}$$

$$\text{Dacă } f^*(x,y) = c(x,y), \text{ atunci } b^p(x,y) \leq 0. \tag{12.4.c}$$

Demonstraţie. Similar ca într-un capitol anterior se poate arăta că minimizarea funcţiei obiectiv $\sum_A b(x,y)\, f(x,y)$ este echivalentă cu minimizarea funcţiei obiectiv $\sum_A b^p(x,y)\, f(x,y)$. Fie p un vector care împreună cu fluxul f^* verifică condiţiile (12.4) şi fie f un flux oarecare. Stabilim că fiecare termen din suma

$$\sum_A b^p(x,y)\,(f(x,y) - f^*(x,y))$$

este nenegativ. Considerăm următoarele trei cazuri:

$c_1)$ $0 < f^*(x,y) < c(x,y)$; în acest caz $(12.4.a)$ implică $b^p(x,y) = 0$, astfel termenul $b^p(x,y)\,(f(x,y) - f^*(x,y))$ este zero.

$c_2)$ $f^*(x,y) = 0$; în acest caz $f(x,y) \geq f^*(x,y) = 0$ şi $(12.4.b)$ implică $b^p(x,y) \geq 0$, astfel termenul $b^p(x,y)\,(f(x,y) - f^*(x,y))$ este nenegativ.

$c_3)$ $f^*(x,y) = c(x,y)$; în acest caz $f(x,y) \leq f^*(x,y) = c(x,y)$ şi $(12.4.c)$ implică $b^p(x,y) \leq 0$, astfel termenul $b^p(x,y)\,(f(x,y) - f^*(x,y))$ este nenegativ.

Am arătat că $b^p(f - f^*) = b^p f - b^p f^* \geq 0$, sau $b^p f^* \leq b^p f$, care justifică demonstraţia teoremei. ■

Teorema 12.6. *O structură pădure extinsă admisibilă* $\left(P', \overline{P}'_0, \overline{P}'_c\right)$ *cu fluxul asociat* f^* *este o structură pădure extinsă optimă pentru un vector potenţial* p *oarecare, atunci perechea* (f^*, p) *verifică următoarele condiţii de optimalitate:*

$$b^p(x,y) = 0 \text{ pentru toate arcele } (x,y) \in P'. \tag{12.5.a}$$

$$b^p(x,y) \geq 0 \text{ pentru toate arcele } (x,y) \in \overline{P}'_0. \tag{12.5.b}$$

$$b^p(x,y) \leq 0 \text{ pentru toate arcele } (x,y) \in \overline{P}'_c. \tag{12.5.c}$$

Demonstraţie. Rezultă imediat din teorema 12.5. ■

În continuare se va prezenta un algoritm pentru determinarea potenţialelor nodurilor unui arbore extins $G' = (N', A'_a)$. Aplicând acest algoritm pentru fiecare arbore extins al pădurii extinse $G'_p = (N, P')$ se obţin potenţialele nod pentru fiecare nod din reţea. Fie $A'_a = A' \cup \{a\},\ a = (x', y')$ cu $G' = (N', A')$ un arbore parţial al grafului $G = (N, A)$ şi x_0 nodul rădăcină al arborelui extins

$G'_a = (N', A'_a)$. Potenţialele nod se determină din condiţiile $b^p(x,y) = 0$ pentru toate arcele $(x,y) \in A'_a$. Iniţial se consideră $p(x_0) = \theta$ şi mai întâi determinăm pentru fiecare nod x, $x \in N'$, $x \neq x_0$, potenţialul ca o funcţie liniară de θ de forma $p(x,\theta) = \lambda_1(x) + \lambda_2(x)\,\theta$ cu $\lambda_1(x)$ şi $\lambda_2(x)$ constante. Valoarea lui θ se determină din condiţia $b^p(x',y') - p(x') + \mu(x',y')\,p(y') = 0$ şi rezultă $\theta = (b(x',y') - \lambda_1(x') + \mu(x',y')\lambda_1(y'))\,/(\lambda_2(x') - \mu(x',y')\lambda_2(y'))$. Înlocuind valoarea lui θ în $p(x,\theta)$ se obţine valoarea numerică a potenţialului $p(x)$ pentru fiecare nod $x \in N'$. Algoritmul PNAE pentru determinarea potenţialelor nodurilor unui arbore extins este prezentat mai jos.

```
(1) PROCEDURA PNAE;
(2) BEGIN
(3)     p(x_0) := θ;
(4)     y := ω(x_0);
(5)     WHILE y ≠ x_0 DO
(6)     BEGIN
(7)         x := π(y);
(8)         IF (x,y) ∈ A
(9)                 THEN p(y,θ) := (p(x,θ) − b(x,y)) /μ(x,y);
(10)        IF (y,x) ∈ A
(11)                THEN p(y,θ) := μ(y,x) p(x,θ) + b(y,x);
(12)        y := ω(x);
(13)    END;
(14)    se calculează valoarea lui θ;
(15)    FOR x ∈ N' DO
(16)            se calculează valoarea lui p(x);
(17) END;
```

Teorema 12.7. *Algoritmul PNAE calculează corect potenţialele nodurilor unui arbore extins.*

Demonstraţie. Corectitudinea algoritmului PNAE rezultă din faptul că vectorul p trebuie să satisfacă condiţia (12.5.a). Numitorul $\lambda_2(x') - \mu(x',y')\lambda_2(y')$ este diferit de zero dacă arborele extins G'_a este bun $(\mu(L'_a) \neq 1)$. ■

Teorema 12.8. *Algoritmul PNAE are complexitatea $O(n)$.*

Demonstraţie. Instrucţiunile WHILE şi FOR se execută fiecare de cel mult n ori. Fiecare iteraţie a acestor instrucţiuni are complexitatea $O(1)$. Deci algoritmul PNAE are complexitatea $O(n)$. ■

Exemplul 12.3. Se consideră arborele extins $G'_a = (N', A'_a)$, $a = (2,3)$ reprezentat în figura 12.3. În tabelul 12.1 sunt date valorile pentru $\omega(x), \pi(x), p(x)$ şi funcţiile $p(x,\theta)$ pentru nodurile x din N', $N' = \{1, 2, 3, 4, 5\}$, $x_0 = 1$.

Fig.12.3

x	$\omega(x)$	$\pi(x)$	$p(x,\theta)$	$p(x)$
1	2	0	θ	- 17
2	4	1	$-10+\theta$	- 27
3	5	1	$-2.5+0.5\,\theta$	- 11
4	3	2	$-22+3\,\theta$	- 73
5	1	3	$-6.25+0.25\,\theta$	- 10.5

Tabelul 12.1

În continuare se va prezenta o metodă pentru determinarea fluxului corespunzător unei structuri pădure extinsă. Pentru început presupunem că reţeaua este necapacitată şi în consecinţă $\overline{P}'_c = \emptyset$. Se va prezenta un algoritm pentru determinarea fluxului corespunzător unui arbore extins. Aplicând acest algoritm pentru fiecare arbore extins se determină fluxul corespunzător pădurii extinse. Se porneşte de la un nod terminal şi se face deplasarea către nodul rădăcină x_0.

Se definesc $e(x) = v(x),\ x \in N'$ şi $f(x,y) = 0,\ (x,y) \in A',\ f(x',y';r) = r$. Astfel $e(x';r) = e(x') - r$ şi $e(y';r) = e(y') + \mu(x',y')\,r$. Se determină fluxurile $f(x,y;r) = \lambda_1(x,y) + \lambda_2(x,y)\,r$. După determinarea valorii numerice a lui r se pot determina valorile numerice pentru $f(x,y),\ (x,y) \in A'_a$.

Fie y un nod terminal din N'. Un singur arc din A' este incident în y: fie arcul (y,x), fie arcul (x,y). Dacă $(y,x) \in A'$ atunci $f(y,x;r) = e(y;r),\ e(y;r) = 0,\ e(x;r) = e(x;r) + \mu(y,x)\,e(y;r)$. Dacă $(x,y) \in A'$ atunci $f(x,y;r) = -\,e(y;r)/\mu(x,y),\ e(y;r) = 0,\ e(x;r) = e(x;r) + e(y;r)/\mu(x,y)$. În continuare eliminăm nodul y şi arcul corespunzător din N' respectiv A' şi repetăm procedura pe noul arbore. La terminare vom obţine $e(x;r) = 0,\ x \in N',\ x \neq x_0,\ e(x_0;r) = \lambda_1(x_0) + \lambda_2(x_0)\,r$ şi $f(x,y;r) = \lambda_1(x,y) + \lambda_2(x,y)r,\ (x,y) \in A'_a$. Din condiţia $e(x;r) = 0$ se obţine $r = -\lambda_1(x_0)/\lambda_2(x_0)$ şi se determină valorile pentru $f(x,y),\ (x,y) \in A'_a$. Algoritmul FAE este prezentat mai jos.

```
(1) PROCEDURA FAE;
(2) BEGIN
(3)     FOR x ∈ N' DO e(x) := v(x);
(4)     FOR (x,y) ∈ N' DO f(x,y) := 0;
(5)     f(x',y') := r; e(x';r) := e(x') − r; e(y';r) = e(y') + μ(x',y') r;
(6)     N'_1 := N'; A'_1 := A';
(7)     WHILE N'_1 ≠ {x_0} DO
(8)     BEGIN
(9)        se selectează un nod terminal y din N'_1;
(10)       x := π(y);
(11)       IF (y,x) ∈ A'_1
(12)          THEN BEGIN
(13)               f(y,x;r) := e(y;r);
(14)               e(x;r) := e(x;r) + μ(y,x) e(y;r);
(15)               e(y;r) := 0; a' = (y,x);
(16)                    END;
(17)       IF (x,y) ∈ A'_1
(18)          THEN BEGIN
(19)               f(x,y;r) := − e(y;r)/μ(x,y);
(20)               e(x;r) := e(x;r) + e(y;r)/μ(x,y);
```

(21) $e(y;r) := 0;\ a' = (x,y);$
(22) END;
(23) $N_1' := N_1' - \{y\};\ A_1' := A_1' - \{a'\};$
(24) END;
(25) $r := -\lambda_1(x_0)/\lambda_2(x_0);$
(26) FOR $(x,y) \in A_a'$ DO
(27) se determină valoarea lui $f(x,y)$;
(28) END;

Teorema 12.9. *Algoritmul FAE determină un flux corespunzător arborelui extins* $G_a' = (N', A_a')$.

Demonstraţie. Algoritmul FAE determină $f(x,y)$ pentru $(x,y) \in A_a'$ care verifică constrângerile (12.1.*b*). Deci $f(x,y)$ este un flux. Mai trebuie să arătăm că $\lambda_2(x_0)$ nu este zero. Dacă $f(x',y';r) = r$ atunci $e(x';r) = -r$ şi $e(y';r) = \mu(x',y')\,r$. Fie $L_{x'}'$ lanţul arbore de la nodul x_0 la nodul x' şi $L_{y'}'$ lanţul de la nodul y' la nodul x_0. Pentru a obţine $e(x';r) = 0$ trebuie trimise $r/\mu\,(L_{x'}')$ unităţi de flux de la nodul x_0 la nodul x'. Similar, dacă se trimit $\mu(x',y')\,r$ unităţi de flux de la nodul y' la nodul x_0 pe lanţul $L_{y'}'$, atunci $\mu(x',y')\,r\,\mu(L_{y'}')$ unităţi de flux sosesc la nodul x_0. Aceste remarci arată că $\lambda_2(x_0) = \mu(x',y')\,\mu\left(L_{y'}'\right) - 1/\mu\,(L_{x'}')$. Deci $\lambda_2(x_0) = 0$ dacă şi numai dacă $\mu(x',y')\,\mu\,(L_{x'})\,\mu\,(L_{y'}) = 1$. Deoarece $\mu(\overset{\circ}{L}{}_a') = \mu(x',y')\,\mu\,(L_{x'}')\,\mu\left(L_{y'}'\right) \neq 1$ (arborele extins G_a' este bun) rezultă că $\lambda_2(x_0) \neq 0$. ■

Teorema 12.10. *Algoritmul FAE are complexitatea* $O(nm)$.

Demonstraţie. Instrucţiunea WHILE se execută de cel mult n ori. Complexitatea unei iteraţii a acestei instrucţiuni este $O(m)$. Deci algoritmul are complexitatea $O(nm)$. ■

Exemplul 12.4. Se consideră arborele extins $G_a' = (N', A_a')\,,\ a = (2,3),\ x_0 = 1$, reprezentat în figura 12.4.

Iniţializări. $\pi = (0,1,1,2,3),\ e(1) = 5,\ e(2) = 0,\ e(3) = 0,\ e(4) = 10,$
$e(5) = -20,\ f(1,2) = 0,\ f(1,3) = 0,\ f(3,5) = 0,\ f(4,2) = 0;$
$f(2,3) = r,\ e(2;r) = -r,\ e(3;r) = 0.5\,r;\ N_1' = N' = \{1,2,3,4,5\},$
$A_1' = A' = \{(1,2),\ (1,3),\ (3,5),\ (4,2)\}.$

Iteraţia 1: $y = 4,\ x = \pi(4) = 2,\ (4,2) \in A_1',\ f(4,2;r) = 10,\ e(2;r) = 30 - r,\ e(4;r) = 0,\ a' = (4,2),\ N_1' = \{1,\ 2,\ 3,\ 5\},\ A_1' = \{(1,2),\ (1,3),\ (3,5)\}.$

Iteraţia 2: $y = 5,\ x = \pi(5) = 3,\ (3,5) \in A_1',\ f(3,5;r) = 10,\ e(3;r) = -10 + 0.5\,r,\ e(5;r) = 0,\ a' = (3,5),\ N_1' = \{1,\ 2,\ 3\},\ A_1' = \{(1,2),\ (1,3)\}.$

Iteraţia 3: $y = 2,\ x = \pi(2) = 1,\ (1,2) \in A_1',\ f(1,2;r) = -30 + r,\ e(1;r) = 35 - r,\ e(2;r) = 0,\ a' = (1,2),\ N_1' = \{1,\ 3\},\ A_1' = \{(1,3)\}.$

Iteraţia 4: $y = 3,\ x = \pi(3) = 1,\ (1,3) \in A_1',\ f(1,3;r) = 10 - 0.5\,r,\ e(1;r) =$

$25 - 0.5\,r$, $e(3;r) = 0$, $a' = (1,3)$, $N'_1 = \{1\}$, $A'_1 = \emptyset$.

Se trece la instrucţiunea din linia (25) şi se obţine $r = 50$ şi valorile fluxului sunt $f(1,2) = 20$, $f(1,3) = -15$, $f(2,3) = 50$, $f(3,5) = 10$, $f(4,2) = 10$.

Fig.12.4

Dacă reţeaua este capacitată atunci iniţial $v(x) = b(x)$ pentru fiecare nod x din N' şi $f(x,y) = 0$ pentru fiecare arc (x,y) din A'. După aceea se execută instrucţiunile:

```
FOR (x,y) ∈ P'_c DO
BEGIN
    f(x,y) := c(x,y);
    e(x) := e(x) − c(x,y);
    e(y) := e(y) + μ(x,y) c(x,y);
END;
```

După aceste iniţializări se aplică algoritmul FAE.

În continuare se va stabili o legătură între pădurile extinse bune ale lui G şi bazele problemei fluxului cu multiplicatori de cost minim.

Fie problema de programare liniară

Minimizează $b\,f$
$\mathcal{A}\,f = v$
$0 \le f \le c$

În această formulare $\mathcal{A}$ este o matrice $p \times q$ cu liniile liniar independente (matricea $\mathcal{A}$ are rangul egal cu p). Fie $\mathcal{A}_i$, $i = 1, \ldots, q$, coloanele matricei $\mathcal{A}$, B o mulţime de indici a p variabile, $f_B = \{f_i \,|\, i \in B\}$, şi $\mathcal{B} = \{\mathcal{A}_i \,|\, i \in B\}$. Este cunoscut următorul rezultat:

Teorema 12.11. *Variabilele f_B definesc o bază a problemei de programare matematică dacă şi numai dacă sistemul de ecuaţii $\mathcal{B} f_B = v$ are o soluţie unică.*

■

Translatând această teoremă la problema fluxului cu multiplicatori de cost minim, se obţine faptul că o submulţime de arce B defineşte o bază a problemei fluxului cu multiplicatori de cost minim dacă şi numai dacă pentru fiecare vector ofertă/cerere v, arcele din B au un flux unic (fără a fi necesar să respecte constrângerile de mărginire) care verifică constrângerile de conservare.

Teorema 12.12. *O mulţime B de arce defineşte o bază a problemei fluxului cu multiplicatori de cost minim dacă şi numai dacă B este o pădure extinsă bună.*

Demonstraţie. Din cele precizate la prezentarea algoritmului FAE rezultă că dacă B este o pădure extinsă bună, atunci fluxul pe fiecare arc al acestei păduri este unic. Deci fiecare pădure extinsă bună defineşte o bază a problemei fluxului cu

multiplicatori de cost minim. În continuare se stabileşte rezultatul invers. Dacă B nu este o pădure extinsă bună, atunci nu poate fi o bază. Pentru a demonstra acestă afirmaţie se utilizează un rezultat bine cunoscut care stabileşte că fiecare bază a unei probleme de programare liniară conţine acelaşi număr de variabile. Deoarece fiecare pădure extinsă bună conţine n arce şi defineşte o bază, fiecare bază B a problemei fluxului cu multiplicatori de cost minim trebuie să conţină n arce. Se consideră o mulţime B de n arce care nu defineşte o pădure extinsă bună şi componentele conexe ale lui B care nu sunt arbori extinşi buni. Există următoarele două cazuri: fie o componentă este un arbore (cazul c_1), fie o componentă este un arbore extins cu $\mu(\overset{\circ}{L}'_a) = 1$ (cazul c_2). (O a treia posibilitate ar fi ca o componentă să aibă două sau mai multe extraarce; dar în acest caz o altă componentă trebuie să satisfacă cazul c_1; deci este suficient să considerăm cazul c_1 şi cazul c_2). Studiem aceste două cazuri separat.

c_1) Fie $\hat{B}$ o componentă a lui B care este un arbore. Desemnăm un nod arbitrar, să spunem nodul rădăcină x_0 al lui $\hat{B}$. Fie $v(x_0) = 1$ şi $v(x) = 0$ pentru fiecare nod $x \in N - \{x_0\}$. Este uşor de verificat că nu se poate niciodată consuma oferta nodului x_0 dacă se restricţionează fluxul numai la arcele arbore. Rezultă că nu se verifică proprietatea că pentru fiecare vector v, un flux unic verifică constrângerile de conservare.

c_2) Fie $\hat{B}$ o componentă a lui B care este un arbore extins cu $\mu(\overset{\circ}{L}'_a) = 1$. Dacă se trimite flux adiţional de-a lungul ciclului $\overset{\circ}{L}'_a$ se menţin constrângerile de conservare pentru toate nodurile ciclului deoarece $\mu(\overset{\circ}{L}'_a) = 1$. Deci, dacă $\hat{B}$ are un flux admisibil, atunci are o infinitate de fluxuri admisibile. Rezultă că nu se verifică proprietatea că pentru fiecare vector v, un flux unic verifică constrângerile de conservare. ■

Deoarece fiecare pădure extinsă bună constituie o bază a problemei fluxului cu multiplicatori de cost minim, fiecare structură pădure extinsă bună defineşte o structură bază a problemei fluxului cu multiplicatori de cost minim.

Este uşor de obţinut o structură pădure extinsă iniţială. Fie $\overline{b}' > \max\{b(x,y) \mid (x,y) \in A\}$ şi pentru fiecare nod x se introduce arcul artificial (x,x) cu $c(x,x) = \infty$, $b(x,x) = \overline{b}'$, $\mu(x,x) = 0.5$ dacă $v(x) > 0$ şi $\mu(x,x) = 2$ dacă $v(x) \leq 0$. Proprietatea 12.4 implică faptul că dacă $f(x,x) = e(x)/(1-\mu(x,x))$ atunci este satisfăcută oferta/cererea la nodul x. De asemenea, deoarece fiecare arc artificial (x,x) are $b(x,x) = \overline{b}'$, nici o soluţie cu un flux pozitiv pe oricare arc artificial nu va fi optimală în afară de cazul când problema fluxului cu multiplicatori de cost minim este inadmisibilă. Vectorul potenţial nod p iniţial se determină din condiţiile $b^p(x,x) = b(x,x) - p(x) + \mu(x,x)\,p(x) = 0$ pentru fiecare nod $x \in N$. Se obţine $p(x) = \overline{b}'/(1-\mu(x,x))$ pentru fiecare nod $x \in N$.

În continuare se prezintă algoritmul simplex pentru fluxuri cu multiplicatori

de cost minim (algoritmul SFMCM).

(1) PROGRAM SFMCM;
(2) BEGIN
(3) se determină o structură pădure extinsă admisibilă iniţială $\left(P', \overline{P}'_0, \overline{P}'_c\right)$;
(4) se determină fluxul iniţial f şi vectorul potenţial iniţial p;
(5) WHILE există arc $(x, y) \in \overline{P}'$ care nu verifică condiţiile de optimalitate
(6) DO BEGIN
(7) se selectează arcul de intrare (x_1, y_1) care nu verifică condiţiile de optimalitate;
(8) se adaugă arcul (x_1, y_1) la pădurea extinsă;
(9) se determină arcul de ieşire (x_2, y_2);
(10) se actualizează f, $\left(P', \overline{P}'_0, \overline{P}'_c\right)$ şi p;
(11) END;
(12) END.

Algoritmul SFMCM menţine o structură pădure extinsă bună la fiecare iteraţie şi prin executarea unei operaţii de pivotare se transformă această soluţie într-o structură pădure extinsă bună îmbunătăţită. Algoritmul repetă acest proces până când structura pădure extinsă bună verifică condiţiile de optimalitate (12.5).

În continuare se prezintă cum se selectează arcul de intrare (x_1, y_1), cum se determină arcul de ieşire (x_2, y_2) şi cum se actualizează fluxul f, vectorul potenţial p şi structura pădure extinsă bună $\left(P', \overline{P}'_0, \overline{P}'_c\right)$.

Există două tipuri de arce eligibile pentru a intra în pădurea extinsă:

1. orice arc $(x, y) \in \overline{P}'_0$ cu $b^p(x, y) < 0$

2. orice arc $(x, y) \in \overline{P}'_c$ cu $b^p(x, y) > 0$

Algoritmul SFMCM poate selecta orice arc eligibil ca arc de intrare. Totuşi, diferite reguli, cunoscute ca reguli pivot, pot fi aplicate pentru selectarea arcului de intrare (x_1, y_1). Acestea sunt:

1. regula pivot a lui Dantzig, care selectează arcul de intrare (x_1, y_1) cu $|b^p(x_1, y_1)| = \max\{|\, b^p(x, y)| \mid (x, y) \in \overline{P}'_0$ sau $(x, y) \in \overline{P}'_c\}$

2. regula pivot a primului arc eligibil, care selectează primul arc cu $|b^p(x, y)| > 0$ întâlnit în examinarea listei arcelor;

3. regula pivot a listei candidat, care menţine o listă candidat de arce (x, y) cu $|\, b^p(x, y)\,| > 0$ şi selectează arcul (x_1, y_1) cu $|b^p(x_1, y_1)| = \max\{|b^p(x, y)| \mid (x, y)$ din lista candidat $\}$.

Presupunem că arcul de intrare (x_1, y_1) este din $\overline{P}'_0$. Se determină mai întâi valoarea cu care fluxul pe orice arc (x, y) din P' se schimbă prin creşterea fluxului cu 1 unitate pe arcul de intrare (x_1, y_1). Fie $g(x, y)$ această valoare. Evident că fluxul pe arcele pădurii extinse se modifică liniar cu fluxul pe arcul de intrare (x_1, y_1) (dacă pe arcul de intrare $g\,(x_1, y_1) = h$ atunci pe arcul (x, y) fluxul este $h\,g(x, y)$). Se consideră $g\,(x_1, y_1) = 1$. Se pot calcula celelalte valori $g(x, y)$ din constrângerile de conservare. Aceste valori se pot calcula cu PROCEDURA FAE pentru arborii extinşi care conţin nodurile x_1 şi y_1, plecând iniţial cu $f = 0$ şi vectorul e:

$$e(x) = \begin{cases} -1 & \text{pentru} \quad x = x_1, \\ \mu(x_1, y_1) & \text{pentru} \quad x = y_1, \\ 0 & \text{pentru} \quad x \neq x_1 \text{ şi } x \neq y_1. \end{cases}$$

Dacă nodurile x_1 şi y_1 sunt din arbori extinşi diferiţi, atunci se execută PROCEDURA FAE de două ori; altfel, se execută PROCEDURA FAE o singură dată. Fluxurile arc obţinute sunt valorile $g(x, y)$. Având valorile $g(x, y)$ se poate calcula uşor fluxul adiţional maxim h pe arcul de intrare (x_1, y_1). Fie $f(x, y)$ fluxul corespunzător structurii pădure extinsă curentă $\left(P', \overline{P}'_0, \overline{P}'_c\right)$. Constrângerile de mărginire a fluxului sunt:

$$0 \leq f(x, y) + h\,g(x, y) \leq c(x, y), \;\; (x, y) \in P' \cup \{(x_1, y_1)\}.$$

Dacă pentru arcul (x, y), $g(x, y) > 0$, atunci fluxul pe arc creşte cu h şi eventual va fi egal cu capacitatea arcului. Similar, dacă $g(x, y) < 0$ fluxul pe arc descreşte cu h şi eventual va fi egal cu zero. Deci, dacă $h(x, y)$ sunt valorile din care se determină valoarea h, atunci:

$$h(x, y) = \begin{cases} \left(c(x, y) - f(x, y)\right)/g(x, y) & \text{dacă} \quad g(x, y) > 0, \\ f(x, y)/\left(-g(x, y)\right) & \text{dacă} \quad g(x, y) < 0, \\ \infty & \text{dacă} \quad g(x, y) = 0. \end{cases}$$

Din constrângerile de mărginire a fluxului se determină h astfel

$$h = \min\{\, h(x, y) \,|\, (x, y) \in P' \cup \{\, (x_1, y_1) \,\}\,\}.$$

Un arc (x, y) pentru care $h = h(x, y)$ se numeşte *arc de blocare.* Arcul de ieşire (x_2, y_2) este orice arc de blocare. Se remarcă faptul că pentru arcul de ieşire (x_2, y_2) avem $g(x_2, y_2) \neq 0$. Se spune că iteraţia este nedegenerată dacă $h = 0$. O iteraţie degenerată are loc numai dacă P' este o pădure extinsă degenerată.

Exemplul 12.5. Se consideră arborele extins prezentat în figura 12.5(a). În figura 12.5(b) sunt date capacităţile arcelor, fluxurile arc curente şi multiplicatorii arcelor. Presupunem că arcul de intrare este $(x_1, y_1) = (5, 6)$. Mai întâi

determinăm valorile $g(x,y)$ ale modificărilor flux. Pentru determinarea acestor valori se consideră $g(5,6)=1$ şi $g(2,3)=r$, unde $(2,3)$ este extraarc. Se aplică PROCEDURA FAE şi se obţin fluxurile $g(x,y;h)$ prezentate în figura 12.5(c). Din ecuaţia $e(1)=0.5-0.5\,r$ se obţine $r=1$. În figura 12.5(b) sunt prezentate valorile $g(x,y)$ şi $h(x,y)$. Utilizând valorile $h(x,y)$ se obţine $h=2$.

Evident că arcul de ieşire este $(x_2,y_2)=(1,3)$ şi arborele extins este prezentat în figura 12.5(d) cu extraarcul $(x',y')=(5,6)$.

Fig.12.5

Structura pădure extinsă bună $\left(P',\,\overline{P}'_0,\,\overline{P}'_c\right)$ se actualizează în modul următor. Dacă arcul de ieşire este acelaşi cu arcul de intrare, atunci arcul (x_1,y_1) trece din mulţimea $\overline{P}'_0$ în mulţimea $\overline{P}'_c$ sau vice versa. Dacă arcul de ieşire diferă de arcul de intrare, atunci se actualizează mulţimile P', $\overline{P}'_0$, $\overline{P}'_c$ aşa cum s-a prezentat în exemplul 12.5. Deoarece fiecare bază a problemei fluxului cu multiplicatori de cost minim este o pădure extinsă bună, algoritmul SFMCM trece o pădure extinsă bună într-o altă pădure extinsă bună.

După actualizarea fluxului f şi a structurii $\left(P',\,\overline{P}'_0,\,\overline{P}'_c\right)$ trebuie actualizat vectorul potenţial p. Evident că este necesar să se actualizeze numai potenţialele nod $p(x)$ pentru nodurile x aparţinând arborilor extinşi care sunt implicaţi în operaţia de pivotare. Recalcularea potenţialelor nod se poate face cu PROCEDURA PNAE. În continuare trebuie actualizate tablourile π şi ω.

Teorema 12.13. *Algoritmul SFMCM determină un flux cu multiplicatori de cost minim în reţeaua* $G=(N,A,c,b,\mu,v)$.

Demonstraţie. Corectitudinea algoritmului SFMCM rezultă din enunţul acestui algoritm şi explicitarea lui. ■

În cazul cel mai defavorabil nu se poate mărgini numărul de iteraţii ale algoritmului SFMCM printr-o funcţie polinomială de n şi m, deşi în practică această mărginire are loc. În cadrul unei iteraţii a algoritmului SFMCM sunt necesare $O(m)$ operaţii pentru identificarea arcului de intrare. Fiecare din celelalte operaţii, cum sunt calculul valorilor $g(x,y)$, actualizarea fluxului, potenţialelor şi tablourilor π,ω necesită $O(m)$ operaţii. Deci, în practică, timpul de execuţie a algoritmului SFMCM este mărginit de o funcţie polinomială de n şi m.

Exemplul 12.6. Se aplică algoritmul SFMCM pentru reţeaua reprezentată în figura 12.6.

Fig.12.6

Iniţializări: se introduc arcele artificiale $(1,1)$, $(2,2)$, $(3,3)$, $(4,4)$ cu $c(x,x)=\infty$, $x=1,2,3,4$ şi $\mu(1,1)=0.5$, $\mu(2,2)=2$, $\mu(3,3)=0.5$, $\mu(4,4)=2$. Se consideră $\overline{b}'=10$. Din $f(x,x)=e(x)/\left(1-\mu(x,x)\right)$ se obţin $f(1,1)=10$, $f(2,2)=$

2, $f(3,3) = 2$, $f(4,4) = 13$ şi din $p(x) = \overline{b}'/(1-\mu(x,x))$ se obţin $p(1) = 20$, $p(2) = -10$, $p(3) = 20$, $p(4) = -10$; $P' = \{(1,1), (2,2), (3,3), (4,4)\}$, $\overline{P}'_0 = \{(1,2), (1,3), (2,3), (2,4), (3,4)\}$, $\overline{P}'_c = \emptyset$.
Iteraţia 1: $b^p(1,2) = b(1,2) - p(1) + \mu(1,2)\, p(2) = 3 - 20 + 2(-10) = -37$ şi arcul de intrare este $(x_1, y_1) = (1,2)$ care se adaugă la P'; $e = (-1,2,0,0)$ şi cu PROCEDURA FAE se obţin valorile $g(1,2) = 1$, $g(1,1) = -2$, $g(2,2) = -2$, $g(3,3) = 0$, $g(4,4) = 0$, valorile $h(x,y)$ sunt: $h(1,2) = (7-0)/1 = 7$, $h(1,1) = 5$, $h(2,2) = 1$, $h(3,3) = \infty$, $h(4,4) = \infty$ şi rezultă că $h = \min\{7,5,1,\infty,\infty\} = 1$; arcul de ieşire este $(x_2, y_2) = (2,2)$; fluxul iniţial $f = (0,0,0,0,0)$ pe arcele efective devine $f = (1,0,0,0,0)$ unde $f(1,2)$ actualizat are valoarea $f(1,2) + hg(1,2) = 0 + 1 \cdot 1 = 1$; $P'\{(1,1),(1,2),(3,3),(4,4)\}$, $\overline{P}'_0 = \{(2,2),(1,3),(2,3),(2,4),(3,4)\}$, $\overline{P}'_c = \emptyset$; $p = (20, 17/2, 20, -10)$.
Iteraţia 2: $b^p(1,3) = -9$, $(x_1, y_1) = (1,3)$; $e = (-1, 0.5, 0, 0)$, $g(1,1) = -2$, $g(1,2) = 0$, $g(3,3) = 1$, $g(4,4) = 0$; $h(1,1) = 4, h(1,2) = \infty, h(1,3) = 4, h(3,3) = 2, h(4,4) = \infty, h = 4$; $(x_2, y_2) = (3,3)$; $f = (1,2,0,0,0)$; $P' = \{(1,1),(1,2),(1,3),(4,4)\}, \overline{P}'_0 = \{(2,2),(3,3),(2,3),(2,4),(3,4)\}, \overline{P}'_c = \emptyset$; $p = (20, 17/2, 38, -10)$.

Iteraţiile se continuă până când $b^p(x,y) \geq 0$, $(x,y) \in \overline{P}'_0 \cup \overline{P}'_c$.

12.2 Fluxuri cu mai multe produse

Fie $G = (N, A, c, c_1, \ldots, c_p, b_1, \ldots, b^p, v_1, \ldots, v_p)$ o reţea orientată cu N mulţimea nodurilor, A mulţimea arcelor, c vectorul capacitate, c_i vectorul capacitate pentru produsul i, b_i vectorul cost pentru produsul i şi v_i vectorul valoare pentru produsul i, $i = 1, \ldots, p$.

Formularea în programarea liniară pentru problema fluxului de cost minim cu p produse este următoarea:

$$\text{Minimizează } z = \sum_{i=1}^{p} b_i\, f_i, \tag{12.6.a}$$

$$\overline{M}\, f_i = v_i, \quad i = 1, \ldots, p \tag{12.6.b}$$

$$\sum_{i=1}^{p} f_i \leq c \tag{12.6.c}$$

$$0 \leq f_i \leq c_i, \quad i = 1, \ldots, p \tag{12.6.d}$$

unde f_i este vectorul flux al produsului i, $\quad i = 1, \ldots, p$ şi $\overline{M}$ este matricea de incidenţă nod arc cu n linii (numărul de noduri) şi m coloane (numărul de arce). Se reaminteşte că fiecare coloană a acestei matrice corespunde unei variabile

$f_i(x,y)$ şi coloana conţine un element +1 în linia corespunzătoare nodului x, un element -1 în linia corespunzătoare nodului y şi restul elementelor sunt zero. Această formulare pentru problema fluxului cu mai multe produse este numită *formularea nod-arc.*

Exemplul 12.7. Fie reţeaua $G = (N, A, c, c_1, c_2, c_3, v_1, v_2, v_3)$ cu $N = \{1,2,3\}$ în această ordine, $A = \{(1,2),(2,3),(3,1)\}$ în această ordine, $c(x,y) = 1,\ (x,y)$ din $A,\ c_i(x,y) = 1/2,\ (x,y) \in A,\ i = 1,2,3$ prezentată în figura 12.7. Nodurile sursă şi stoc pentru cele trei produse sunt următoarele: $s_1 = 1,\ t_1 = 3;\ s_2 = 2,\ t_2 = 1;\ s_3 = 3,\ t_3 = 2$. Care sunt fluxurile f_1, f_2, f_3 corespunzătoare celor trei produse astfel încât suma valorilor $v_{11} + v_{22} + v_{33}$ ale acestor fluxuri să fie maximă.

Fig.12.7

Determinarea fluxului maxim pentru problema cu trei produse din figura 12.7 este relativ simplă deoarece există numai un drum D_k de la nodul sursă s_i la nodul stoc t_i pentru fluxul f_i corespunzător produsului $i,\ i = 1,2,3$. Drumurile pentru fluxurile f_1, f_2, f_3 sunt următoarele: $D_1 = ((1,2),(2,3)),\ D_2 = ((2,3),(3,1)),\ D_3 = ((3,1),(1,2))$. Cele trei fluxuri sunt $f_1 = (1/2, 1/2, 0), f_2 = (0, 1/2, 1/2), f_3 = (1/2, 0, 1/2)$ cu valorile $v_{11} = 1/2, v_{22} = 1/2, v_{33} = 1/2$. Evident că $f_1(x,y) + f_2(x,y) + f_3(x,y) = 1 = c(x,y),\ (x,y) \in A$. Rezultă că acest flux cu trei produse este maxim cu valoarea $v_{11} + v_{22} + v_{33} = 3/2$. Din acest exemplu se remarcă faptul că problema fluxului cu mai multe produse nu furnizează în mod necesar fluxuri întregi.

În majoritatea problemelor practice constrângerile (12.6.d) nu sunt necesare. De aceea uneori se va renunţa la aceste restricţii. Inegalitatea vectorială (12.6.c) se poate transforma într-o egalitate prin adunarea vectorului ecart u la membrul stâng.

Considerând variabilele ecart ale vectorului ecart u şi ignorând constrângerile (12.6.d), problema fluxului de cost minim cu p produse are $m(p+1)$ variabile şi $n\,p + m$ constrângeri. Astfel, chiar pentru probleme cu dimensiuni moderate numărul de constrângeri va fi mare. De exemplu, presupunem că avem o problemă cu $n = 100$, $m = 250$ şi $p = 10$. Această problemă va avea 2750 variabile şi 1250 constrângeri.

Deoarece problema fluxului de cost minim cu mai multe produse are o structură bloc diagonală se poate aplica algoritmul descompunerii Dantzig-Wolf din programarea liniară.

Fie $F_i = \{f_i \,|\, \overline{M}\, f_i = v_i,\ 0 \le f_i \le c_i\}$. Presupunem că fiecare componentă a lui c_i este finită. Deci tronsoanele $F_i,\ 1 \le i \le p$, sunt mărginite şi F_i este poliedru convex. Deci oricare f_i poate fi exprimat ca o combinaţie convexă de

punctele extreme ale lui F_i sub forma:

$$f_i = \sum_{j=1}^{k_i} \lambda_{ij}\, f_{ij}$$

cu

$$\sum_{j=1}^{k_i} \lambda_{ij} = 1$$

$$\lambda_{ij} \geq 0,\ j = 1, \ldots, k_i$$

unde $f_{i1}, \ldots, f_{ik_i}$ sunt punctele extreme ale lui F_i. Substituind pe f_i în problema (12.6) şi introducând vectorul ecart u se obţine problema principală:

$$\text{Minimizează } \sum_{i=1}^{p} \sum_{j=1}^{k_i} (b_i f_{ij})\, \lambda_{ij} \tag{12.7.a}$$

$$\sum_{i=1}^{p} \sum_{j=1}^{k_i} f_i\ \lambda_{ij} + u = c \tag{12.7.b}$$

$$\sum_{j=1}^{k_i} \lambda_{ij} = 1,\ i = 1, \ldots, p \tag{12.7.c}$$

$$\lambda_{ij} \geq 0,\ j = 1, \ldots, k_i,\ i = 1, \ldots, p \tag{12.7.d}$$

$$u \geq 0. \tag{12.7.e}$$

Presupunem că avem o soluţie admisibilă de bază pentru problema (12.7) în termenii variabilelor λ_{ij} şi fie (w, α) vectorul variabilelor duale corespunzător soluţiei admisibile de bază (w are m componente şi α are p componente). Atunci admisibilitatea dual̆a este dată prin următoarele constrângeri:

$$w(x, y) \leq 0,\ (x, y) \in A \tag{12.8.a}$$

$$w\, f_{ij} + \alpha_i - b_i\, f_{ij} \leq 0,\ j = 1, \ldots, k_i,\ i = 1, \ldots, p. \tag{12.8.b}$$

Constrângerile (12.8.a) corespund variabilelor ecart $u(x, y)$ şi constrângerile (12.8.b) corespund variabilelor λ_{ij}. Dacă oricare din aceste constrângeri nu este verificată atunci variabila corespunzătoare $u(x, y)$ sau λ_{ij} este candidată pentru a intra în baza principală. Pentru un produs dat i, o variabilă nebazică λ_{ij}

poate intra în bază dacă valoarea funcţiei obiectiv a următoarei subprobleme este pozitivă:

$$\text{Maximizează } ((w - b_i)f_i + \alpha_i) \tag{12.9.a}$$

$$\overline{M}\, f_i = v_i \tag{12.9.b}$$

$$0 \leq f_i \leq c_i \tag{12.9.c}$$

Deoarece $\overline{M}$ este matricea de incidenţă nod-arc, aceasta este o problemă de flux cu un singur produs. Astfel, se poate rezolva prin una din tehnicile eficiente pentru problema fluxului cu un singur produs.

Matricea $\overline{M}$ are o linie pentru fiecare nod al reţelei şi o coloană pentru fiecare arc. Fiecare coloană conţine exact două elemente diferite de zero: un +1 şi un -1. Coloana asociată arcului (x, y) conţine un +1 în linia corespunzătoare nodului x şi un -1 în linia corespunzătoare nodului y şi restul elementelor sunt zero. Astfel fiecare coloană $\overline{m}^c_{xy}$ a matricei $\overline{M}$ se poate exprima în modul următor:

$$\overline{m}^c_{xy} = \overline{e}_x - \overline{e}_y$$

unde $\overline{e}_x$ şi $\overline{e}_y$ sunt vectori unitari din $\Re^n$, cu 1 în poziţia x şi respectiv în poziţia y. Este cunoscut faptul că rangul matricei $\overline{M}$ este $n - 1$. Pentru a putea aplica algoritmul simplex trebuie să adăugăm o variabilă artificială astfel încât rangul noii matrici să fie n. Introducerea variabilei artificiale corespunzătoare nodului z conduce la matricea $\overline{M}' = [\overline{M}, e_z]$. Orice soluţie bazică trebuie să conţină n coloane liniar independente. Deci variabila artificială trebuie să apară în fiecare soluţie bazică. Coloana corespunzătoare variabilei artificiale poate fi considerată că este asociată unui arc $(z, \ )$ care nu are nod terminal. Acest arc se numeşte *arc rădăcină* şi nodul z se numeşte *nod rădăcină.* Coloanele corespunzătoare unui arbore parţial al reţelei împreună cu coloana corespunzătoare variabilei artificiale formează o bază pentru matricea $\overline{M}$. Arborele parţial corespunzător bazei se numeşte *arbore parţial rădăcină.*

Exemplul 12.8. Fie digraful $G = (N, A)$ reprezentat în figura 12.8(a). Digraful $\tilde{G} = \left(\tilde{N}, \tilde{A}\right)$ cu $\tilde{N} = N \cup \{z\}$, $\tilde{A} = A \cup \{(z, \ )\}$, $z = 4$ este reprezentat în figura 12.8(b) şi arborele parţial rădăcină $\tilde{G}' = \left(\tilde{N}', \tilde{A}'\right)$ este reprezentat în figura 12.8(c).

Fig.12.8

Matricea de incidenţă nod-arc $\overline{M}$ pentru digraful $G = (N, A)$ este

$$\overline{M} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccccc} (1,2) & (1,3) & (2,3) & (2,4) & (3,2) & (3,4) \end{array} \\ \left[\begin{array}{cccccc} 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & -1 & 0 \\ 0 & -1 & -1 & 0 & 1 & 1 \\ 0 & 0 & 0 & -1 & 0 & -1 \end{array}\right] \end{array}$$

Matricea $\overline{M}' = \left(\overline{M}, e_4\right)$ asociată digrafului $\tilde{G} = \left(\tilde{N}, \tilde{A}\right)$ este formată din matricea $\overline{M}$ la care se adaugă vectorul coloană $e_4 = (0, 0, 0, 1)$. Baza B pentru $\overline{M}'$ la care corespunde arborele parţial rădăcină $\tilde{G}' = \left(\tilde{N}, \tilde{A}'\right)$ este

$$\overline{B} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} (1,3) & (2,3) & (3,4) & (4,\ ) \end{array} \\ \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{array}\right] \end{array}$$

Concluzionăm cu următoarea teoremă:

Teorema 12.14. *Fie o problemă de flux de cost minim definită pe o reţea conexă G având un arc rădăcină. Atunci B este o matrice bază pentru această problemă dacă şi numai dacă ea este matricea de incidenţă nod-arc a arborelui parţial rădăcină $\tilde{G}'$.* ■

Evident că acest rezultat este valabil pentru problema fluxului de cost minim cu un singur produs. Pentru problema fluxului de cost minim cu p produse este necesar să adăugăm o variabilă artificială pentru fiecare produs. În acest caz matricea problemei este

$$\left[\begin{array}{cccccccc} f_1 & f_2 & . & . & . & f_p & f & \\ I' & I' & . & . & . & I' & I & \\ \overline{M}' & \overline{M}'_0 & . & . & . & \overline{M}'_0 & \overline{M}_0 & \\ \overline{M}'_0 & \overline{M}' & . & . & . & \overline{M}'_0 & \overline{M}_0 & \\ . & . & . & . & . & . & . & \\ \overline{M}'_0 & \overline{M}'_0 & . & . & . & \overline{M}' & \overline{M}_0 & \end{array}\right] \begin{array}{ll} m & linii \\ n & linii \\ n & linii \\ \ldots & \\ n & linii \end{array}$$

unde $\overline{M}' = \left[\overline{M}, e_z\right]$, $I' = [I, e_0]$ cu I matricea unitate $m \times m$, e_0 un vector cu toate elementele nule, $\overline{M}'_0$ o matrice cu toate elementele nule. Selectând o submatrice bază din matricea anterioară se obţine

$$\left[\begin{array}{cccccc} E_1 & E_2 & . & . & . & E_p & E \\ \overline{M}'_1 & \overline{M}'_{02} & . & . & . & \overline{M}'_{0p} & \overline{M}_{00} \\ \overline{M}'_{01} & \overline{M}'_2 & . & . & . & \overline{M}'_{0p} & \overline{M}_{00} \\ . & . & . & . & . & . & . \\ \overline{M}'_{01} & \overline{M}'_{02} & . & . & . & \overline{M}'_p & \overline{M}_{00} \end{array}\right]$$

Aplicând algoritmul descompunerii din programarea liniară problemei fluxului de cost minim cu mai multe produse se obţine următorul algoritm:

(1) PROGRAM FCMMP;
(2) BEGIN
(3) se determină o soluţie admisibilă de bază pentru problema principală;
(4) se determină baza B, inversa B^{-1} şi vectorul (w, α);
(5) se calculează tabloul simplu revizuit;
(6) $k := 0$;
(7) REPEAT
(8) $k := k + 1$;
(9) IF există $w(x, y) > 0$
(10) THEN BEGIN
(11) se actualizează coloana corespunzătoare variabilei $u(x, y)$;
(12) se execută o pivotare;
(13) END
(14) ELSE FOR $i := 1$ TO p DO
(15) BEGIN
(16) se determină f_{ik} din problema (12.4);
(17) se calculează $z_{ik} - b_{ik}$;
(18) IF $z_{ik} - b_{ik} > 0$
(19) THEN BEGIN
(20) se actualizează coloana corespunzătoare variabilei λ_{ik};
(21) se execută o pivotare;
(22) END;
(23) END;
(24) UNTIL nu există variabilă $u(x, y)$ sau λ_{ik} care să intre în bază;
(25) se determină soluţia optimă $f_i^*,\ i = 1, \ldots, p$;
(26) END.

Fie $\hat{c} = \begin{bmatrix} c \\ \overline{e} \end{bmatrix}$, unde $\overline{e}$ este vectorul coloană cu p elemente toate egale cu 1.

Dacă B este matricea bază atunci $\hat{v} = B^{-1} \hat{c}$ şi $(w, \alpha) = \hat{b}_B B^{-1}$, unde $\hat{b}_{ij} = b_i f_{ij}$ pentru variabilele λ_{ij}. De asemenea avem $z_{ik} - b_{ik} = (w - b_i) f_{ik} + \alpha_i$. Actualizarea coloanei corespunzătoare variabilei $w(x, y)$ sau λ_{ik} constă în a calcula

$$B^{-1} \begin{pmatrix} e_{xy} \\ e_0 \end{pmatrix} \text{ respectiv } B^{-1} \begin{pmatrix} f_{ik} \\ e_i \end{pmatrix}$$

unde e_{xy} este vectorul unitar cu m componente care are 1 în linia corespunzătoare

arcului (x, y) şi restul elementelor sunt 0, e_0 este vectorul coloană cu p elemente cu toate elementele 0, f_{ik} este vectorul coloană cu m componente soluţie a problemei (12.9.b) şi (12.9.c) şi e_i este vectorul unitar cu p componente care are 1 pe poziţia i şi restul elementelor sunt zero.

Exemplul 12.9. Să considerăm problema fluxului de cost minim cu două produse reprezentată în figura 12.9.

Matricea constrângerilor şi membrul drept sunt prezentate în figura 12.10 (constrângerile $0 \leq f_1 \leq c_1$ şi $0 \leq f_2 \leq c_2$ nu sunt prezentate). Precizăm că $f_1(4, \ )$ şi $f_2(4, \ )$ sunt variabilele artificiale.

Observăm structura constrângerilor de mărginire şi structura diagonală bloc a constrângerilor de conservare a fluxului.

Pentru a evita metoda celor două faze începem cu următoarele soluţii admisibile:

$$f_{10} = \begin{bmatrix} f_1(1,2) \\ f_1(2,3) \\ f_1(3,4) \\ f_1(4,1) \\ f_1(4,2) \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 2 \\ 0 \\ 0 \end{bmatrix} \text{ şi } f_{20} = \begin{bmatrix} f_2(1,2) \\ f_2(2,3) \\ f_2(3,4) \\ f_2(4,1) \\ f_2(4,2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 3 \\ 0 \\ 3 \end{bmatrix}$$

Fig.12.9

	$f_1(1,2)$	$f_1(2,3)$	$f_1(3,4)$	$f_1(4,1)$	$f_1(4,2)$	$f_1(4,)$	$f_2(1,2)$	$f_2(2,3)$	$f_2(3,4)$	$f_2(4,1)$	$f_2(4,2)$	$f_2(4,)$	$u(1,2)$	$u(2,3)$	$u(3,4)$	$u(4,1)$	$u(4,2)$	MD
	-5	-1	0	- 1	-4	0	2	0	-2	1	-6	0	0	0	0	0	0	0
	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	4
Constrângerile	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	3
de	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	7
mărginire	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1
	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	5
Matricea de	1	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	2
incidenţă	-1	1	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	1
pentru	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
subproblema 1	0	0	-1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	-2
Matricea de	0	0	0	0	0	0	1	0	0	-1	0	0	0	0	0	0	0	0
incidenţă	0	0	0	0	0	0	-1	1	0	0	-1	0	0	0	0	0	0	-3
pentru	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	3
subproblema 2	0	0	0	0	0	0	0	0	-1	1	1	1	0	0	0	0	0	0

Figura 12.10

Remarcăm că baza principală corespunde celor cinci variabile ecart şi variabilelor λ_{10} şi λ_{20}. Baza şi inversa ei sunt:

$$B = \begin{array}{c} \begin{matrix} u(1,2) & u(2,3) & u(3,4) & u(4,1) & u(4,2) & \lambda_{10} & \lambda_{20} \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

$$B^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & -2 & 0 \\ 0 & 1 & 0 & 0 & 0 & -3 & 0 \\ 0 & 0 & 1 & 0 & 0 & -2 & -3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -3 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Avem $b_1 = (5, 1, 0, 1, 4)$ şi $b_2 = (-2, 0, 2, -1, 6)$, deci $b_1 f_{10} = 13$ şi $b_2 f_{20} = 24$.

Rezultă că $\hat{b}_B = (0,0,0,0,0,13,24)$, $(w,\infty) = \hat{b}_B\ B^{-1} = (0,0,0,0,0,13,24)$,

$$f_B = B^{-1}\ \hat{c} = B^{-1} \begin{bmatrix} 4 \\ 3 \\ 7 \\ 1 \\ 5 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 2 \\ 1 \\ 2 \\ 1 \\ 1 \end{bmatrix} \text{ şi } z = \hat{b}_B\ f_B = 37$$

Tabelul simplex revizuit este:

(w,α)	z
B^{-1}	f_B

care pentru problema noastră devine:

	$w(1,2)$	$w(2,3)$	$w(3,4)$	$w(4,1)$	$w(4,2)$	α_1	α_2	
z	0	0	0	0	0	13	24	37
$u(1,2)$	1	0	0	0	0	-2	0	2
$u(2,3)$	0	1	0	0	0	-3	0	0
$u(3,4)$	0	0	1	0	0	-2	-3	2
$u(4,1)$	0	0	0	1	0	0	0	1
$u(4,2)$	0	0	0	0	1	0	-3	2
λ_{10}	0	0	0	0	0	1	0	1
λ_{20}	0	0	0	0	0	0	1	1

Toate aceste calcule reprezintă iniţializările specificate în liniile (3) şi (4) ale algoritmului. În continuare se execută instruţiunea REPEAT.

Iteraţia $k = 1$.

Toate variabilele $w(x,y) \leq 0$ şi se trece la linia (14).

Subproblema $i = 1$**.**

$w - b_1 = (0,0,0,0,0) - (5,1,0,1,4) = (-5,-1,0,-1,-4)$

Subproblema 1 este problema fluxului cu un singur produs prezentată în figura 12.11.

Fig.12.11

O soluţie optimă (maximă) este $f_{11} = (2,3,2,0,0)^t$ şi $z_{11} - b_{11} = (w - b_1)f_{11} + \alpha_1 = -13 + 13 = 0$. Astfel, nu există variabilă de intrare pentru subproblema 1.

Subproblema $i = 2$**.**

$w - b_2 = (0,0,0,0,0) - (-2,0,2,-1,6) = (2,0,-2,1,-6)$

Subproblema 2 este problema fluxului cu un singur produs prezentată în figura 12.12.

O soluţie optimă (maximă) este $f_{21} = (3, 0, 3, 3, 0)^t$ şi $z_{21} - b_{21} = (w - b_2)f_{21} + \alpha_2 = 3 + 24 = 27$. Astfel, variabila λ_{21} intră în bază. Actualizarea coloanei corespunzătoare variabilei λ_{21} (exclusiv $z_{21} - b_{21}$) este

$$B^{-1} \begin{bmatrix} f_{21} \\ 0 \\ 1 \end{bmatrix} = (3, 0, 0, 3, -3, 0, 1)^t.$$

Fig.12.12

Procesul de pivotare este prezentat în continuare.

	$w(1,2)$	$w(2,3)$	$w(3,4)$	$w(4,1)$	$w(4,2)$	α_1	α_2	MD		λ_{21}
z	0	0	0	0	0	13	24	37		27
$u(1,2)$	1	0	0	0	0	-2	0	2		3
$u(2,3)$	0	1	0	0	0	-3	0	0		0
$u(3,4)$	0	0	1	0	0	-2	-3	2		0
$u(4,1)$	0	0	0	1	0	0	0	1		**3**
$u(4,2)$	0	0	0	0	1	0	-3	2		-3
λ_{10}	0	0	0	0	0	1	0	1		0
λ_{20}	0	0	0	0	0	0	1	1		1

	$w(1,2)$	$w(2,3)$	$w(3,4)$	$w(4,1)$	$w(4,2)$	α_1	α_2	MD
z	0	0	0	-9	0	13	24	28
$u(1,2)$	1	0	0	-1	0	-2	0	1
$u(2,3)$	0	1	0	0	0	-3	0	0
$u(3,4)$	0	0	1	0	0	-2	-3	2
λ_{21}	0	0	0	1/3	0	0	0	1/3
$u(4,2)$	0	0	0	1	1	0	-3	3
λ_{10}	0	0	0	0	0	1	0	1
λ_{20}	0	0	0	1/3	0	0	1	2/3

Iteraţia $k = 2$.

Toate variabilele $w(x, y) \leq 0$ şi se trece la linia (14).

Subproblema $i = 1$.

$w - b_1 = (0, 0, 0, -9, 0) - (5, 1, 0, 1, 4) = (-5, -1, 0, -10, -4)$

Subproblema 1 este problema fluxului cu un singur produs prezentată în figura 12.13.

Fig.12.13

O soluţie optimă (maximă) este $f_{12} = (2, 3, 2, 0, 0)^t$ şi $z_{12} - b_{12} = (w - b_1)f_{12} + \alpha_1 = -13 + 13 = 0$. Astfel, nu există variabilă de intrare pentru subproblema 1.

Subproblema $i = 2$**.**

$w - b_2 = (0, 0, 0, -9, 0) - (-2, 0, 2, -1, 6) = (2, 0, -2, -8, -6)$

Subproblema 2 este problema fluxului cu un singur produs prezentată în figura 12.14.

O soluţie optimă (maximă) este $f_{22} = (3, 0, 3, 3, 0)^t$ şi $z_{22} - b_{22} = (w - b_2)f_{22} + \alpha_2 = -24 + 24 = 0$. Astfel, nu există variabilă de intrare pentru subproblema 2.

Este îndeplinită condiţia din linia (24) şi nu se mai execută instrucţiunea REPEAT.

Fig.12.14

Soluţia optimă este următoarea: $f_1^* = \lambda_{10} f_{10} = 1 \cdot (2, 3, 2, 0, 0)^t = (2, 3, 2, 0, 0)^t$, $f_2^* = \lambda_{20} f_{20} + \lambda_{21} f_{21} = 2/3 \cdot (0, 0, 3, 0, 3)^t + 1/3 \cdot (3, 0, 3, 3, 0)^t$, $= (1, 0, 3, 1, 2)^t$ cu valoarea funcţiei obiectiv $z^* = 28$.

Alte abordări de rezolvare a problemei fluxului de cost minim cu mai multe produse sunt tratate în bibliografia indicată. De asemenea, în această bibliografie este tratată problema fluxului maxim cu mai multe produse.

12.2.1 Aplicaţii şi comentarii bibliografice

12.2.2 Încărcarea maşinilor

Problema încărcării maşinilor apare într-o varietate de domenii aplicative. Se cere planificarea producţiei a k produse pe k' maşini. Presupunem că maşina j' este disponibilă pentru $h(j')$ ore şi că oricare maşină poate să producă fiecare produs. Producând 1 unitate din produsul i pe maşina j' se consumă $h'(i, j')$ ore din timpul maşinii şi costă $\beta(i, j')$ unităţi de cost. Pentru a satisface cererile de produse, trebuie produse $\alpha(i)$ unităţi din produsul i. Problema încărcării maşinilor cere să se determine producerea, la cel mai mic cost de producţie posibil, a k produse pe k' maşini.

Această problemă se poate modele ca o problemă de flux de cost minim cu multiplicatori. Se construieşte reţeaua $G = (N, A, c, b, \mu, v)$ cu mulţimea nodurilor $N = \{1, \ldots, k, 1', \ldots, k'\}$, mulţimea arcelor $A = \{(i, j'), (j', j') \mid i = 1, \ldots, k,\ j' = 1', \ldots, k'\}$, $c(i, j') = \infty$, $b(i, j') = \beta(i, j')$, $\mu(i, j') = h'(i, j')$, $(i, j') \in A$, $c(j', j') = \infty$, $b(j', j') = 0$, $\mu(j', j') = 2$, $v(i) = \alpha(i)$, $i = 1, \ldots, k$, $v(j') = -h(j')$, $j' = 1', \ldots, k'$. Graful $G = (N, A)$ este reprezentat în figura 12.15.

Fig.12.15

Scopul arcelor (j', j') este să explicăm disponibilităţile de timp neîndeplinite ale maşinilor: aşa cum s-a precizat în paragraful 12.1 se poate trimite flux adiţional de-a lungul acestor arce pentru a genera flux suficient pentru a consuma exact timpul disponibil al maşinii.

12.2.3 Depozitarea produselor sezoniere

O companie manufacturieră execută produse sezoniere cu cererile variind săptămânal, lunar sau trimestrial. Pentru utilizarea forţei de muncă şi capitalul echipamentului eficient, compania doreşte să "uşureze" producţia, depozitând producţia pre-sezon pentru suplimentarea producţie plin-sezon. Compania are o depozitare cu capacitate fixată c care este utilizată să depoziteze toate produsele executate. Problema este să se identifice nivelele de producţie a tuturor produselor pentru fiecare săptămână, lună sau trimestru al anului care va permite să satisfacă cererile şi costurile să fie minime.

Această problemă se poate modela ca o problemă de flux cu mai multe produse. Pentru simplificare, se consideră că această companie execută 2 produse şi că este necesar să se planifice producţia pentru patru trimestre ale anului. Fie $v_1(j),\ v_2(j)$ cererea pentru produsul 1, respectiv 2 în trimestrul j. Presupunem că în trimestrul j capacităţile şi costurile de producţie sunt $c_1(j),\ c_2(j)$ şi respectiv $b_1(j),\ b_2(j)$ pentru cele două produse. Fie $b'_1(j),\ b'_2(j)$ costurile de stocare ale celor două produse de la trimestrul j la trimestrul $j+1$.

Se contruieşte reţeaua $G = (N, A, c, c_1, c_2, b_1, b_2, v_1, v_2)$. Mulţimea nodurilor este $N = N_1 \cup N_2 \cup N_3$ cu $N_1 = \{s_1, s_2\}$ mulţimile nodurilor sursă, $N_2 = \{1, 2, 3, 4\}$ mulţimea nodurilor trimestru, $N_3 = \{t_1, t_2\}$ mulţimea nodurilor stoc. Mulţimea arcelor este $A = A_1 \cup A_2 \cup A_3$ cu $A_1 = \{(s_i, j) \,|\, s_i \in N_1,\ j \in N_2\},\ A_2 = \{(j, j+1) \mid j \in N_2 - \{4\}\},\ A_3 = \{(j, t_k) \mid j \in N_2,\ t_k \in N_3\}$. Graful $G = (N, A)$ este reprezentat în figura 12.16. Capacităţile şi costurile arcelor sunt următoarele: $c_i(s_i, j) = c_i(j),\ b_i(s_i, j) = b_i(j),\ (s_i, j) \in A_1,\ i = 1, 2;\ c(j, j+1) = c$, c este capacitatea depozitului pentru cele două produse, $b_i(j, j+1) = b'_i(j),\ (j, j+1) \in A_2,\ i = 1, 2;\ c_k(j, t_k) = v_k(j),\ (j, t_k) \in A_3,\ k = 1, 2.$

Fig.12.16

Este uşor de văzut că fiecare flux admisibil cu mai multe produse f în reţeaua G specifică o producţie admisibilă. Prin optimizarea fluxului cu mai multe produse se determină o producţie optimă.

12.2.4 Comentarii bibliografice

Problema fluxului cu multiplicatori de cost minim, prezentată în acest capitol, este o adaptare a metodei simplex din programarea liniară. Prezentarea din acest capitol este o prelucrare a acestei probleme preluată din Ahuja R.K., Magnanti T.L. şi Orlin J.B.(1993).

Cercetătorii au studiat şi alte abordări pentru problema fluxului cu multiplicatori de cost minim. Aceste abordări sunt următoarele:

1. algoritmul primal-dual dezvoltat de Jewell;
2. algoritmul dual propus de Jensen şi Bhaumik;
3. algoritmul relaxării dezvoltat de Bertsekas şi Tseng;
4. metoda punctului interior propusă de Goldfarb şi Lin.

Cercetătorii au propus numeroase abordări pentru rezolvarea problemei fluxului cu mai multe produse de cost minim. Prezentarea din acest capitol este o prelucrare a acestei probleme preluată din Bazarra M.S., Jarvis J.J. şi Sherali H.D. (2005).

Ford şi Fulkerson au propus primii o procedură de generare a coloanei pentru problema fluxului cu mai multe produse. Aceasta a precedat procedura de descompunere a lui Dantzig-Wolfe pentru programarea liniară. Hartman J.K., Lasdon L.S. au propus o procedură bazată pe metoda simplex pentru rezolvarea problemei fluxului cu mai multe produse. O discuţie bună şi sintetică a acestei probleme este prezentată de Kennington J.F., Helgason R.V. şi Ahuja R.K., Magnanti T.L., Orlin J.B.

Bibliografie

Aarts, E. and Lenstra, J. K.(1997): *Local Search in Combinatorial Optimization*. Wiley, New York.

Abu-Sbeih, M. Z.(1990): On the number of spanning trees of K_n and $K_{m,n}$.*Discr. Math.* **84**, 205-207.

Aho, A. V., Hopcroft, J. E and Ullman, J. D. (1974): *The Design and Analysis of Computer Algorithms*. Addison Wesley, Reading, Mass.

Aho, A.V., Hopcroft, J. E. and Ullman, J. D. (1983): *Data Structures and Algorithms*. Addison Westley, Reading, Mass.

Ahuja, R. K., Goldberg, A. V., Orlin, J. B. and Tarjan, R. E. (1992): Finding minimum-cost flows by double scaling. *Math. Progr.* **53**, 243-266.

Ahuja, R. K., Kodialam, M., Mishra, A. K. and Orlin, J. B. (1992): Computational testing of maximum flow algorithms. Sloan working paper, Sloan School of Management, MIT.

Ahuja, R. K., Magnanti, T. L. and Orlin, J. B. (1989): Network flows. In:*Handbooks in Operations Research and Management Science, Vol 1: Optimization* (Eds. G. L. Nemhauser, A. H. G. Rinnooy Kan and M. J. Todd). North Holland, Amsterdam, pp. 211-369.

Ahuja R. K., Magnanti, T. L. and Orlin, J. B. (1991): Some recent advances in network flows. *SIAM Review* **33**, 175-219.

Ahuja, R. K., Magnanti, T. L. and Orlin, J. B. (1993): *Network Flows: Theory, Algorithms and Applications.* Prentice Hall, Englewood, Cliffs, N. J.

Ahuja, R. K., Mehlhorn, K., Orlin, J. B. and Tarjan, R. E. (1990): Faster algorithms for the shortest path problem. *J. ACM* **37**, 213-223.

Ahuja, R. K. and Orlin, J. B. (1989): A fast and simple algorithm for the maximum flow problem. *Oper. Res.* **37**, 748-759.

Ahuja, R. K. and Orlin, J. B. (1992): The scaling network simplex algorithm. *Oper. Res.* **40**, Suppl. 1. pp. S 5 - S 13.

Ahuja, R. K. and Orlin, J. B. (1995): A capacity scaling algorithm for the constrained maximum flow problem. *Networks* **25**, 89-98.

Ahuja, R. K., Orlin, J. B., Stein, C. and Tarjan, R. E. (1994): Improved algorithm for bipartite network flow. *SIAM J. Computing* **23**, 906-933.

Ahuja, R. K., Orlin, J. B. and Tarjan, R. E. (1989): Improved time bounds for the maximum flow problem. *SIAM J. Comp.* **18**, 939-954.

Aigner, M. (1984): *Graphentheorie. Eine Entwicklung aus dem 4-Farben-Problem.* Teubner, Stuttgart.

Aigner, M. (1997): *Combinatorial Theory.* Springer, New York.

Alon, N. (1990): Generating pseudo-random permutations and maximum flow algorithms. *Inform. Proc. Letters* **35**, 201-204.

Anderson, I. (1990): *Combinatorial Designs: Construction Methods.* Ellis Horwood Ltd., Chichester.

Anderson, S. S. and Harary, F. (1967): Trees and unicyclic graphs. *Math. Teacher* **60**, 345-348.

Applegate, D., Bixby, R., Chvátal, V. and Cook, W. (1995): Finding cuts in the TSP (A preliminary report). DIMACS Technical Report 95-05, March 1995.

Applegate, D. and Cook, W. (1993): Solving large-scale matching problems. In: *Network Flows and Matching* (Eds. D. S. Johnson and C. C. McGeoch). Amer. Math. Soc., Providence, pp. 557-576.

Arkin, E. M. and Papadimitriou, C.H. (1986): On the complexity of circulations. *J. Algor.* **7**, 134-145.

Aronson, J.E. (1989): A survey of dynamic networks flows. *Annals of Operation Research* **20**, 1 - 66.

Ausiello, G., Italiano, G.F., Marchetti Spaccamela, A. and Nanni, U. (1991): Incremental algorithms for minimal length paths. *J. Algor.* **12**, 615-638.

Bachmann, F. (1989): *Ebene Spiegelungsgeometrie.* B.I. Wissenschaftsverlag, Mannheim- Wien-Zürich.

Balas, E. and Fischetti, M. (1993): A lifting procedure for the asymmetric traveling salesman polytope and a large new class of facets. *Math. Progr.* **58**, 325-352.

Ball, M. O. (1985): Polynomial algorithms for matching problems with side constraints. Research report CORR 85-21, University of Waterloo.

Bandelt, H. J. (1990): Recognition of tree matrices. *SIAM J. Discr. Math.* **3**, 1-6.

Bang-Jensen, J. and Gutin, G. (2001):*Digraph: Theory, Algorithms and Applications.* Springer-Verlag, London.

Barahona, F. (1990): On some applications of the chinese postman problem. In: *Paths, flows and VLSI-Layout* (Eds. B. Korte, L. Lovász, H. J. Prömel and A. Schrijver). Springer, Berlin, pp. 1-16.

Barahona, F. and Pulleyblank, W. R. (1987): Exact arborescences, matchings and cycles. *Discr. Appl. Math.* **16**, 91-99.

Barahona, F. and Tardos, E. (1989): Note on Weintraub's minimum-cost circulation algorithm. *SIAM J. Comp.* **18**, 579-583.

Bauer, F. L. and Wössner, H. (1982): *Algorithmic Language and Program Development.* Springer, Berlin.

Bazaraa, M. S., Jarvis J. J. and Sherali, H. D. (1990): *Linear Programming and Network Flows (*2^{nd} *edition).* Wiley, New York.

Bazaraa, M, S., Sherali, H. D. and Shetty, C. M. (1993): *Nonlinear Programming: Theory and Algorithms (*2^{nd} *edition).* Wiley, New York.

Bellman, R. E. (1958): On a routing problem. *Quart. Appl. Math.* **16**, 87-90.

Bentley, J. L. (1990): Experiments on traveling salesman heuristics. *Proc. First SIAM Symp. on Discr. Algorithms*, pp. 91-99.

Berge, C. (1957): Two theorems in graph theory. *Proc. Nat. Acad. Sc. USA* **43**, 842-844.

Berge, C. (1958): Sur le couplage maximum d'un graphe. *C. R. Acad. Sci. Paris* **247**, 258-259.

Berge, C. (1973): *Graphs and Hypergraphs.* North Holland, Amsterdam.

Berge, C. and Chvátal, V. (1984): *Topics in Perfect Graphs.* North Holland, Amsterdam.

Berge, C. and Ghouila-Houri, A. (1962): *Programmes, Jeux et Réseaux de Transport.* Dunod, Paris.

Berman, P. and Ramaiyer, V. (1994): Improved approximations for the Steiner tree problem. *J. Algor.***17**, 381-408.

Bermond, J. C. (1978): Hamiltonian graphs. In: *Selected topics in graph theory* (Eds. L. W. Beineke and R. J. Wilson). Academic Press, New York, pp. 127-167.

Bermond, J. C., Ed. (1992): *Interconnection Networks.* North Holland, Amsterdam.

Bertsekas, D. P. (1991): *Linear Network Optimization: Algorithms and Codes.* The MIT Press Cambridge, Massachusetts.

Bertsekas, D. P. (1993): A simple and fast label correcting algorithm for shortest paths. *Networks* **23**, 703-709.

Beth, T., Jungnickel, D. and Lenz, H. (1998): *Design Theory (2nd ed.).* Cambridge University Press, Cambridge.

Bien, F. (1989): Constructions of telephone networks by group representations. *Notices AMS* **36**, 5-22.

Bienstock, D., Brickell, E. F. and Monma, C. N. (1990): On the structure of minimum- weight k-connected spanning networks. *SIAM J. Discr. Math.***3**, 320-329.

Biggs, N. L. (1993): *Algebraic Graph Theory (2^{nd} edition)*. Cambridge University Press, Cambridge.

Bland, R. G. and Shallcross, D. F. (1989): Large traveling salesman problems arising from experiments in x-ray crystallography: A preliminary report on computation. *Oper. Res. Letters* **8**, 125-128.

Bollobas, B. (1978): *Extremal Graph Theory*. Academic Press, New York.

Bondy, J. A. and Murty, U. S. R. (1976): *Graph Theory with Applications*. North Holland, Amsterdam.

Book, R. V. (1994): Relativizations of the $P =?NP$ and other problems: developments in structural complexity theory. *SIAM Review* **36**, 157-175.

Borgwardt, K. H. (1987): *The Simplex Method. A Probabilistic Analysis*. Springer, Berlin.

Boruvka, O. (1926a): O jistém problému minimálním. *Acta Societ. Scient. Natur. Moravicae* **3**, 37-58.

Boruvka, O. (1926b): Príspevek k resení otázky ekonomické stavby elektrovodních sítí. *Elektrotechnicky obzor* **15**, 153-154.

Bosák, J. (1990): *Decompositions of Graphs*. Kluwer Academic Publishers, Dordrecht.

Busacker, R. G. and Gowen, P. J. (1961): A procedure for determining a family of minimum cost flow networks. ORO Tehn. Report 15, John Hopkins University.

Busacker, R. G. and Saaty, T. L. (1965): *Finite Graphs and Networks.* McGraw-Hill, New York.

Cai, X., Sha, D. and Wong, C.K. (2001): Time-varying universal maximum flow problems. *Mathematical and Computers Modelling* **30**, 407 -430.

Camerini, P. M., Maffioli, F., Martello, S. and Toth, P. (1986): Most and least uniform spanning trees. *Discr. Appl. Math.* **15**, 181-197.

Cameron, P. J. (1976): *Parallelisms of Complete Designs*, Cambridge University Press, Cambridge.

Cameron P. J. and Van Lint, J. H. (1991): *Designs, Graphs, Codes and their Links*. Cambridge University Press, Cambridge.

Campbell, D. M. and Radford, D. (1991): Tree isomorphism algorithms: Speed versus clarity. *Math. Magazine* **64**, 252-261.

Carré, P. A. (1971): An algebra for network routing problems. *J. Inst. Math. Appl.* **7**, 273-294.

Carré, P. A. (1979): *Graphs and Networks.* Oxford University Press, Oxford.

Cayley, A. (1889): A theorem on trees. *Quart. J. Math.* **23**, 376-378.

Chandrasekaran, R., Aneja, Y. P. and Nair, K. P. K. (1981): Minimal cost reliability ratio spanning tree. *Ann. Discr. Math.* **11**, 53-60.

Chavátal, V. (1983): *Linear Programming.* Freeman, New York.

Chavátal, V. (1985): Hamiltonian cycles. In: *The travelling salesman problem* (Eds. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys). Wiley, New York, pp. 403-429.

Chavátal, V. and Erdös, P. (1972): A note on Hamiltonian circuits. *Discr. Math.* **2**, 111-113.

Chavátal, V. and Thomasson, C. (1978): Distances in orientations of graphs. *J. Comb. Th. (B)* **24**, 61-75.

Chen, W. K. (1990): *Theory of Nets: Flows in Networks.* Wiley, New York.

Cheng, C. K. and Hu, T. C. (1992): Maximum concurrent flows and minimum cuts. *Algorithmica* **8**, 233-249.

Cheriton, D. and Tarjan, R. E. (1976): Finding minimum spanning trees. *SIAM J. Comp.* **5**, 724-742.

Cheriyan, J. and Hagerup, T. (1989): A randomized maximum flow algorithm. *Proc.* 30^{th} *IEEE Conf. on Foundations of Computer Science,* pp. 118-123.

Cheriyan, J. and Hagerup, T. (1995): A randomized maximum flow algorithm. *SIAM J. Computing* **24**, 203-226.

Cheriyan, J., Hagerup, T. and Mehlhorn, K. (1996): A $o(n^3)$-time maximum flow algorithm. *SIAM J. Computing* **25**, 1144-1170.

Cheriyan, J. and Maheshwari, S. N. (1989): Analysis of preflow push algorithms for maximum network flow. *SIAM J. Comp.* **18**, 1057-1086.

Cherkassky, B. V. and Goldberg, A. V. (1995): On implementing Push-Relabel method for the maximum flow problem. In: *Integer programming and combinatorial optimization* (Eds. E. Balas and J. Clausen). Springer, Berlin, pp. 157-171.

Cheung, T. Y. (1980): Computational comparison of eight methods for the maximum network flow problem. *ACM Trans. Math. Software* **6**, 1-16.

Christofides, N. (1975): *Graph Theory: An Algorithmic Approach.* Academic Press, New York.

Chu, Y. J. and Liu, T. H. (1965): On the shortest arborescence of a directed graph. *Sci. Sinica* **14**, 1396-1400.

Chung, F. R. K., Garey, M. R. and Tarjan, R. E. (1985): Strongly connected orientations of mixed multigraphs. *Networks* **15**, 477-484.

Ciupală, L. (2004): About Universal Maximal Dynamic Flows. *Analele Universităţii din Bucureşti* **nr. 1**, 115-124.

Ciupală, L. (2005): A scaling out-of-kilter algorithm for minimum cost flow. *Control and Cybernetics*, vol. 34, **no. 4**, 1169-1174.

Ciurea, E. (1979): Deux remarques sur le flot dynamique maximal de coût minimal. *R.A.I.R.O.,* vol. 13, **no. 3**, 303-306.

Ciurea, E. (1982): Fiabilité d'un réseau dynamique. *Analele ştiinţifice ale Universităţii "Al. I.Cuza" din Iaşi*, tom. XXVIII, s. Ia, 85 - 88.

Ciurea, E. (1984): Les problemes des flots dynamiques. *Cahiers du C.E.R.O.,* vol. 26, **no. 1 -2**, 3-9.

Ciurea, E. (1985): Les problemes des flots dynamiques maximaux speciaux. *Analele ştiinţifice ale Universităţii "Al. I.Cuza" din Iaşi*, tom. XXXI, s. Ia, 297 - 300.

Ciurea, E. (1987): Les problemes des flots dynamiques avec les temps de parcours nonstationnaires. *Bulletin Mathématique de la Société des Sciences Mathématiques de la Roumanie,* tome 31(79), **no. 3**, 205-208.

Ciurea, E. (1995): Two remarks concerning the dynamic flow with nonstationary time function. *Cahiers du C.E.R.O.,* vol. 37, 55-63.

Ciurea, E. (1997a): Counterexemples in maximal dynamic flows.. *Libertas Mathematica,* tom. XVII, 77-87.

Ciurea, E. (1997b): Maximal dynamic flows in arc capacitated networks. *Computer Science Journal of Moldova,* vol. 5, **no. 3**(15), 298-308.

Ciurea, E. (1998): An algorithm for dynamic flows with demand at sink. *Technology Letters of Buckinghamshire University,* vol. 2, **no. 1**, 37-41.

Ciurea, E. (2000a): An algorithm for minimal dynamic flow. *Korean Journal of Computational and Applied Mathematics,* vol. 7, **no. 3**, 259-269.

Ciurea, E. (2000b): Dynamic flows with supply and demand. *Control and Cybernetics,* vol. 29, **no. 4**, 895-903.

Ciurea, E. (2001): *Algoritmi. Introducere în algoritmica grafurilor.* Editura Tehnică, Bucureşti.

Ciurea, E. (2002): Second best temporally repeated flows. *Korean Journal of Computational and Applied Mathematics,* vol. 9, **no. 1**, 77-86.

Ciurea, E. and Ciupală, L. (2001): Algorithms for minimum flows. *Computer Science Journal of Moldova,* vol. 9, **no. 3**(27), 275-290.

Ciurea, E. and Ciupală, L. (2004): Sequential and parallel algorithms for minimum flows. *Journal of Applied Mathematics and Computing,* vol. 15, **no. 1-2**, 53-75.

Ciurea, E., Luca, C. (2005): *Algoritmi şi programare Java.* Editura Albastră, Cluj-Napoca.

Ciurea, E., Tăbîrcă, S., Tăbîrcă, T.(1997): *Algoritmi. Metode de elaborare* Editura Universităţii *Transilvania* din Braşov.

Colbourn, C. J. (1987): *The Combinatorics of Network Reliability.* Oxford University Press, Oxford.

Colbourn, C. J., Day, R. P. J. and Nel, L. D. (1989): Unranking and ranking spanning trees of a graph. *J. Algor.* **10**, 271-286.

Conrad, A., Hindrichs, T., Morsy, H. and Wegener, I. (1994): Solution of the knight's Hamiltonian path problem on chessboards. *Discr. Appl. Math.* **50**, 125-134.

Corman, T. H., Leiserson, C. E. and Rivest, R. L. (1990): *Introduction To Algorithms.* MIT Press, Cambridge, Massachusetts.

Croitoru, C. (1992): *Tehnici de bază în optimizarea combinatorie.* Editura Universităţii 'Al. I. Cuza', Iaşi.

Cvetkovic, D. M., Doob, M., Gutman, I. and Torgasev, A. (1987): *Recent Results in the Theory of Graph Spectra.* North Holland, New York.

Cvetkovic, D. M., Doob, M. and Sachs, H. (1980): *Spectra of Graphs.* Academic Press, New York.

Dantzig, G. B., Ford, L. R. and Fulkerson, D. R. (1956): A primal-dual algorithm for linear programs. In: *Linear inequalities and related systems* (Eds. H. W. Kuhn and A. W. Tucker). Princeton University Press, Princeton, pp. 171-181.

Dantzig, G. B., Fulkerson, D. R. and Johnson, S. M. (1954): Solution of a large-scale traveling-salesman problem. *Oper. Res.* **2**, 393-410.

de Werra, D. (1980): Geography, games and graphs. *Discr. Appl. Math.* **2**, 327-337.

de Werra, D. (1981): Scheduling in sports. *Ann. Discr. Math.* **11**, 381-395.

de Werra, D. (1982): Minimizing irregularities in sports schedules using graph theory. *Discr. Appl. Math.* **4**, 217-226.

de Werra, D. (1988): Some models of graphs for scheduling sports competitions. *Discr. Appl. Math.* **21**, 47-65.

de Werra, D., Jacot-Descombes, L. and Masson, P. (1990): A constrained sports scheduling problem. *Discr. Appl. Math.* **26**, 41-49.

Deo, N., Prabhu, G. M. and Krishnamoorthy, M. S. (1982): Algorithms for generating fundamental cycles in a graph. *ACM Trans. Math. Software* **8**, 26-42.

Derigs, U. (1988): *Programming in Networks and Graphs.* Springer, Berlin.

Derigs, U. and Meier, W. (1989): Implementing Goldberg's max-flow algorithm - A computational investigation. *ZOR* **33**, 383-403.
Derigs, U. and Metz, A. (1991): Solving (large scale) matching problems combinatorially. *Math. Progr.* **50**, 113-121.
Dijkstra, E. W. (1959): A note on two problems in connexion with graphs. *Numer. Math.* **1**, 269-271.
Dinic, E. A. (1970): Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet. Math. Dokl.* **11**, 1277-1280.
Dixon, B., Rauch, M. and Tarjan, R. E. (1992): Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM J. Computing* **21**, 1184-1192.
Drăgan, I. (1976): *Tehnici de bază în programarea liniară.* Editura Tehnică, Bucureşti.
Edmonds, J. (1965a): Maximum matching and a polytope with 0,1-vertices. *J. Res. Nat. Bur. Stand. B* **69**, 125-130.
Edmonds, J. (1965b): Paths, Trees and flowers. *Canad. J. Math.* **17**, 449-467.
Edmonds, J. (1967a): An introduction to matching. Lecture Notes, Univ. of Michigan.
Edmonds, J. (1967b): Optimum branchings. *J. Res. Nat. Bur. Stand. B* **71**, 233-240.
Edmonds, J. and Johnson, E. L. (1973): Matching, Euler tours and the Chinese postman. *Math. Progr.* **5**, 88-124.
Edmonds, J. and Karp, R. M. (1972): Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM* **19**, 248-264.
Elias, P., Feinstein, A. and Shannon, C. E. (1956): Note on maximum flow through a network. *IRE Trans. Inform. Th.* **IT-12**, 117-119.
Engel, K. (1997): *Sperner Theory.* Cambridge University Press, Cambridge.
Eppstein, D. (1994): Offline algorithms for dynamic minimum spanning tree problems. *J. Algor.* **17**, 237-250.
Ervolina, T. R. and McCormick, S. T. (1993): Two strongly polynominal cut cancelling algorithms for minimum cost network flow. *Discr. Appl. Math.* **46**, 133-165.
Euler, L. (1736): Solutio problematis ad geometriam situs pertinentis. *Comment. Acad. Sci. Imper. Petropol.* **8**, 128-140.
Evans, J. R. and Minieka, E. (1992): *Optimization Algorithms for Networks and Graphs.* Marcel Dekker, Inc., New York.
Even, S. (1973): *Combinatorial Algorithms.* Macmillan, New York.

Even, S. (1979): *Graph Algorithms.* Computer Science Press, Rockville, Md.

Even, S., Garey, M. R. and Tarjan, R. E. (1977): A note on connectivity and circuits in directed graphs. Unpublished manuscript.

Even, S., Itai, A. and Shamir, A. (1976): On the complexity of timetable and multicommodity flow problems. *SIAM J. Comp.* **5**, 691-703.

Even, S. and Tarjan, R. E. (1975): Network flow and testing graph connectivity. *SIAM J. Comp.* **4**, 507-512.

Fiechter, C. N. (1994): A parallel tabu search algorithm for large traveling salesman problems. *Discr. Appl. Math.* **51**, 243-267.

Fleischner, H. (1990): *Eulerian Graphs and Related Topics, Part 1, Vol. 1.* North Holland, Amsterdam.

Fleischner, H. (1991): *Eulerian Graphs and Related Topics, Part 1, Vol. 2.* North Holland, Amsterdam.

Floyd, R. W. (1962): Algorithm 97, Shortest path. *Comm. ACM* **5**, 345.

Ford, L. R. (1956): *Network Flow Theory.* Rand Corp., Santa Monica, Cal.

Ford, L. R. and Fulkerson, D. R. (1956): Maximal flow through a network. *Canad. J. Math.* **8**, 399-404.

Ford, L. R. and Fulkerson, D. R. (1957): A simple algorithm for finding maximal network flows and an application to the Hitchcock problem. *Canad. J. Math.* **9**, 210-218.

Ford, L. R. and Fulkerson, D. R. (1958a): Constructing maximal dynamic flows from static flows. *Oper. Res.* **6**, 419-433.

Ford, L. R. and Fulkerson, D. R. (1958b): Network flow and systems of representatives. *Canad. J. Math.* **10**, 78-84.

Ford, L. R. and Fulkerson, D. R. (1958c): A suggested computation for maximum multi-commodity network flows. *Management Sc.* **5**, 97-101.

Ford, L. R. and Fulkerson, D. R. (1962): *Flows in Networks.* Princeton University Press, Princeton, N. J.

Frank, H. and Frish, I. I. (1971) *Communication, Transsmition, and Transportation Network.* Addison-Wesley, Reading, MA.

Frank, A. and Tardos, E. (1988): Generalized polymatroids and submodular flows. *Math. Progr.* **42**, 489-563.

Frederickson, G. N. (1985): Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comp.* **14**, 781-798.

Frederickson, G. N. (1987): Fast algorithms for shortest paths in planar graphs. *SIAM J. Comp.* **16**, 1004-1022.

Fredman, M. L. and Tarjan, R. E. (1987): Fibonacci heaps and their uses on improved network optimization algorithms. *J. ACM* **34**, 596-615.

Fredman, M. L. and Willard, D. E. (1994): Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comp. Syst. Sc.* **48**, 533-551.

Fujishige, S. (1986): An $O(m^3 \log n)$ capacity-rounding algorithm for the minimum- cost circulation problem: A dual framework of Tardos' algorithm. *Math. Progr.* **35**, 298-308.

Fujishige, S. (1991): *Submodular Functions and Optimization.* North Holland, Amsterdam.

Fulkerson, D. R. (1956): Note on Dilworth's decomposition theorem for partially ordered sets. *Proc. AMS* **7**, 701-702.

Fulkerson, D. R. (1959): Increasing the capacity of a network: The parametric budget problem. *Management Sc.* **5**, 472-483.

Fulkerson, D. R. (1961): An out-of-kilter method for minimal cost flow problems. *J. SIAM* **9**, 18-27.

Gabow, H. N. (1976): An efficient implementation of Edmonds' algorithm for maximum matchings on graphs. *J. ACM* **23**, 221-234.

Gabow, H. N. (1985): Scaling algorithms for network problems. *Journal of Computer and System Sciences* **31**, 148-168.

Gabow, H. N. (1990): Data structures for weighted matching and nearest common ancestors with linking. *Proc. First Annual ACM-SIAM Symposium on Discrete Algorithms*, Philadelphia, SIAM, pp. 434-443.

Gabow, H. N., Galil, Z., Spencer, T. and Tarjan, R. E. (1986): Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica* **6**, 109-122.

Gabow, H. N. and Kariv, O. (1982): Algorithms for edge coloring bipartite graphs and multigraphs. *SIAM J. Comp.* **11**, 117-129.

Gabow, H. N. and Tarjan, R. E. (1988): Algorithms for two bottleneck optimization problems. *J. Algor.* **9**, 411-417.

Gabow, H. N. and Tarjan, R. E. (1989): Faster scaling algorithms for network problem. *SIAM J. Comp.* **18**, 1013-1036.

Gabow, H. N. and Tarjan, R. E. (1991): Faster scaling algorithms for general graph- matching problems. *J. ACM* **38**, 815-853.

Gale, D. (1957): A theorem on flows in networks. *Pacific J. Math.* **7**, 1073-1082.

Gale, D. (1959): Transient flows in networks. *Michigan Mathematical Journal*, **6**, 59-63.

Galil, Z. (1980): Finding the vertex connectivity of graphs. *SIAM J. Comp.* **9**, 197-199.

Galil, Z. (1981): On the theoretical efficiency of various network flow algorithms. *Theor. Comp. Sc.* **14**, 103-111.

Galil, Z., Micali, S. and Gabow, H. (1986): An $O(EV \log V)$ algorithm for finding a maximal weighted matching in general graphs. *SIAM J. Comp.* **15**, 120-130.

Galil, Z. and Schieber, B. (1988): On finding most uniform spanning trees. *Discr. Appl. Math.* **20**, 173-175.

Galil, Z. and Tardos, E. (1988): An $O(n^2(m + n \log n) \log n)$ min-cost flow algorithm. *J. ACM* **35**, 374-386.

Gallo, G., Grigoriades, M. D. and Tarjan, R. E. (1989): A fast parametric maximum flow algorithm and applications. *SIAM J. Comp.* **18**, 30-55.

Gallo, G. and Pallottino, S. (1988): Shortest path algorithms. *Annals of Operations Research* **13**, 3-79.

Garey, M. R. and Johnson, D. S. (1979): *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York.

Garey, M. R., Johnson, D. S. and Stockmeyer, L. J. (1976): Some simplified NP- complete graph problems. *Theoret. Comp. Sc.* **1**, 237-267.

Garey, M. R., Johnson, D. S. and Tarjan, R. E. (1976): The planar Hamiltonian circuit problem is NP-complete. *SIAM J. Comp.* **5**, 704-714.

Gilbert, E. N. and Pollak, H. O. (1968): Steiner minimal trees. *SIAM J. Appl. Math.* **16**, 1-29.

Glover, F., Klingman, D. and Philips, N. V. (1992): *Network Models in Optimization and their Applications in Practice.* Wiley, New York.

Goldberg, A. V. (1985): A new max-flow algorithm. Technical Report MIT/LCS/
TM-291, Laboratory for Computer Science, MIT, Cambridge, MA.

Goldberg, A. V., Grigoriadis, M. D. and Tarjan, R. E. (1991): Use of dynamic trees in a network simplex algorithm for the maximum flow problem. *Math. Progr.* **50**, 277-290.

Goldberg, A. V., Plotkin, S. A. and Tardos, E. (1991): Combinatorial algorithms for the generalized circulation problem. *Math. Oper. Res.* **16**, 351-381.

Goldberg, A. V., Tardos, E. and Tarjan, R. E. (1990): Network flow algorithms. In: *Paths, flows and VLSI-layout* (Eds. B. Korte, L. Lovász, H. J. Prömel and A. Schrijver). Springer, Berlin, pp. 101-164.

Goldberg, A. V. and Tarjan, R. E. (1988): A new approach to the maximum flow problem. *J. ACM* **35**, 921-940.

Goldberg, A. V. and Tarjan, R. E. (1989): Finding minimum-cost circulations by canceling negative cycles. *J. ACM* **36**, 873-886.

Goldberg, A. V. and Tarjan, R. E. (1990): Solving minimum cost-flow problems by successive approximation. *Math. of Oper. Res.* **15**, 430-466.

Goldfarb, D. and Grigoridias, M. D. (1988): A computational comparison of the Dinic and network simplex methods for maximum flow. *Ann. Oper. Res.* **13**, 83-123.

Goldfarb, D. and Hao, J. (1990): A primal simplex algorithm that solves the maximum flow problem in at most nm pivots and $O(n^2m)$ time. *Math. Progr.* **47**, 353-365.

Goldfarb, D. and Hao, J. (1991): On strongly polynomial variants of the network simplex algorithm for the maximum flow problem. *Oper. Res. Letters* **10**, 383-387.

Golumbic, M. C. (1980): *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York.

Gomory, R. E. and Hu, T. C. (1961): Multi-terminal network flows. *J. SIAM* **9**, 551-570.

Gomory, R. E. and Hu, T. C. (1962): An application of generalized linear programming to network flows. *J. SIAM* **10**, 260-283.

Gomory, R. E. and Hu, T. C. (1964): Synthesis of a communication network. *J. SIAM* **12**, 348-369.

Gondran, M. and Minoux, N. (1984): *Graphs and Algorithms.* Wiley, New York.

Gonzaga, C. C. (1992): Path-following methods for linear programming. *SIAM Review* **34**, 167-227.

Goulden, I. P. and Jackson, D. M. (1983): *Combinatorial Enumeration.* Wiley, New York.

Graham, R. L. and Hell, P. (1985): On the history of the minimum spanning tree problem. *Ann. History of Comp.* **7**, 43-57.

Griggs, T. and Rosa, A. (1996): A tour of European soccer schedules, or testing the popularity of GK_{2n}. *Bull. ICA* **18**, 65-68.

Gross, J. and Yellen, J. (1999): *Graph Theory and its Applications*, CRC Press, New York.

Grötschel, M. (1985): *Operations Research I.* Vorlesungsskript, Universität Augsburg.

Grötschel, M. and Holland, O. (1991): Solution of large-scale symmetric travelling salesman problems. *Math. Progr.* **51**, 141-202.

Grötschel, M., Jünger, M. and Reinelt, G. (1991): Optimal control of plotting and drilling machines: a case study. *ZOR* **35**, 61-84. **Grötschel, M., Lovász, L. and Schrijver, A.** (1993): *Geometric Algorithms and Combinatorial Optimization (2^{nd} edition).* Springer, Berlin.

Gusfield, D. (1990): Very simple methods for all pairs network flow analysis. *SIAM J. Comp.* **19**, 143-155.

Gusfield, D. and Irving, R. W. (1989): *The Stable Marriage Problem. Structure and Algorithms.* The MIT Press, Cambridge, Mass.

Gusfield, D., Martel, C. and Fernandez-Baca, D. (1987): Fast algorithms for bipartite network flow. *SIAM Comp.* **16**, 237-251.

Gusfield, D. and Naor, D. (1991): Efficient algorithms for generalized cut- trees. *Networks* **21**, 505-520.

Hadlock, F. O. (1975): Finding a maximum cut of a planar graph in polynomial time. *SIAM J.Comp.* **4**, 221-225.

Hall, M. J. (1986): *Combinatorial Theory (2^{nd} edition).* Wiley, New York.

Halpern, J. (1979): Generalized dynamic flows.*Networks,* **9**, 133 -167.

Hamacher, H. W. and Ruhe, G. (1994): On spanning tree problems with multiple objectives. *Ann. Oper. Res.* **52**, 209-230.

Harary, F. (1969): *Graph Theory.* Addison Wesley, Reading, Mass.

Hassin. R. and Johnson, D. B. (1985): An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks. *SIAM J. Comp.* **14**, 612-624.

Held, M. and Karp, R. (1970): The travelling salesman problem and mimimum spanning trees. *Oper. Res.* **18**, 1138-1162.

Held, M. and Karp, R. (1971): The travelling salesman problem and minimum spanning trees II. *Math. Progr.* **1**, 6-25.

Hitchcock, F. L. (1941): The distribution of a product from several sources to numerous localities. *J. Math. Phys.* **20**, 224-230.

Ho, J.-M., Lee, D. T., Chang, C.-H. and Wong, C. K. (1991): Minimum diameter spanning trees and related problems. *SIAM J. Computing* **20**, 987-997.

Hoffman, A. J. (1974): A generalization of max flow-min cut. *Math. Progr.* **6**, 352-359.

Hoffman, A. J. and Markowitz, H. M. (1963): A note on shortest path, assignment and transportation problems. *Naval Research Logistics Quarterly* **10**, 375-379.

Hopcroft, J. and Ullman, J. D. (1979): *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, Reading, Mass.

Hoppe, B. and Tardos, E. (1994): Polynomial time algorithms for some eva-cuation problems. *Proc. 5th Annual ACM-SIAM Symp. Discrete Algorithms,* 433-441.

Hoppe, B. and Tardos, E. (1995): The quickest transshipment problem. *Proc. 6th Annual ACM-SIAM Symp. Discrete Algorithms,* 512-521.

Horton, J. D. (1987): A polynomial time algorithm to find the shortest cycle basis of a graph. *SIAM J. Comp.* **16**, 358-366.

Hu, T. C. (1961): The maximum capacity route problem. *Oper. Res.* **9**, 898-900.

Hu, T. C. (1969): *Integer Programming and Network Flows.* Addison-Wesley, Reading, MA.

Hu, T. C. (1974): Optimum communication spanning trees. *SIAM J. Computing* **3**, 188-195.

Hung, M. S. and Divoky, J. J. (1988): A computational study of efficient shortest path algorithms. *Computers and Operations Research* **15**, 567-576.

Hwang, F. K., Richards, D. S. and Winter, P. (1992): *The Steiner Tree Problem.* North Holland, Amsterdam.

Imai, H. (1983): On the practical efficiency of various maximum flow algorithms. *J. Oper. Res. Soc. of Japan* **26**, 61-82.

Iri, M. (1969): *Network Flow: Transportation and Scheduling.* Academic Press, New York.

Itai, A., Perl, Y. and Shiloach, Y. (1982): The complexity of finding maximum disjoint paths with length constraints. *Networks* **12**, 277-286.

Itai, A. and Rodeh, M. (1978): Finding a minimum circuit in a graph. *SIAM J. Comp.* **7**, 413-423.

Itai, A. and Shiloach, Y. (1979): Maximum flow in planar networks. *SIAM J. Comp.* **8**, 135-150.

Jarník, V. (1930): O jistém problému minimálním. *Acta. Societ. Scient. Natur. Moravicae* **6**, 57-63.

Jarrah, A. I. Z., Yu, G., Krishnamurthy, N. and Rakshit, A. (1993): A decision support framework for airline flight cancellations and delays. *Transportation Sc.* **27**, 266-280.

Jensen, T. R. and Toft, B. (1995): *Graph Coloring Problems.* Wiley, New York.

Jensen, K. and Wirth, N. (1985): *PASCAL User Manual and Report (3^{rd} edition).* Springer, New York.

Johnson, D. B. (1975): Priority queues with update and minimum spanning trees. *Inf. Proc. Letters* **4**, 53-57.

Johnson, D. B. (1977): Efficient shortest path algorithms. *J. ACM.* **24**, 1-13.

Johnson, D. S. and McGeoch, C. C., Eds. (1993): *Network Flows and Matching.* American Mathematical Society, Providence.

Johnson, D. S. and Venkatesan, S. M. (1982): Using divide and conquer to find flows in directed planar networks in $O(n^{\frac{3}{2}} \log n)$ time. *Proc. 20^{th} Allerton Conf. on Communication, Control and Computing*, Urbana, Univ. of Illinois, pp. 898-905.

Jungnickel, D. (1999): *Graphs, Networks and Algorithms.* Springer, Berlin.

Kalaba, R. (1960): On some communication network problems. In: *Combinatorial analysis* (Eds. R. E. Bellman and M. Hall). AMS, Providence, pp. 261-280.

Karger, D. R. (1997): Using random sampling to find maximum flows in uncapacitated undirected graphs. Preprint, MIT Laboratory for Computer Science.

Karp, R. M. (1978): A characterization of the minimum cycle mean in a digraph. *Discr. Math.* **23**, 309-311.

Karzanov, A. V. (1974): Determining the maximal flow in a network with the method of preflows. *Soviet Math. Dokl.* **15**, 434- 437.

Khachiyan, L. G. (1979): A polynomial algorithm in linear programming. *Soviet. Math. Dokl.* **20**, 191-194.

King, V., Rao, S. and Tarjan, R. (1994): A faster deterministic maximum flow algorithm. *J. Algorithms* **17**, 447-474.

Kirchhoff, G. (1847): Über die Auflösungen der Gleichungen, auf die man bei der Untersuchung der Verteilung galvanischer Ströme geführt wird. *Ann. Phys. Chem.* **72**, 497-508.

Kirkman, T. P. (1847): On a problem in combinatorics. Cambridge and Dublin Math. J. **2**, 191-204.

Kirkman, T. P. (1850): Query VI. *Lady's and gentleman's diary* **147**, 48.

Klein, M. (1967): A primal method for minimal cost flows, with applications to the assignment and transportation problems. *Management Sc.* **14**, 205-220.

Kleitman, D. J. and West, D. B. (1991): Spanning trees with many leaves. *SIAM J. Discr. Math.* **4**, 99-106.

Klingman, D. and Philipps, N. V. (1986): Network algorithms and applications. *Discr. Appl. Math.* **13**, 107-292.

Kocay, W. and Stone, D. (1993): Balanced network flows. *Bull. ICA* **7**, 17-32.

Kocay, W. and Stone, D. (1995): An algorithm for balanced flows. *J. Comb. Math. Comb. Comp.* **19**, 3-31.

König, D. (1916): Über Graphen und ihre Anwendungen auf Determinantentheorie und Mengenlehre. *Math. Ann.* 77, 453-465.

König, D. (1931): Graphen und Matrizen (Hungarian with a summary in German). *Mat. Fiz. Lapok* **38**, 116-119.

Korte, B., Lovász, L., Prömel, H. J. and Schrijver, A. (1990): *Paths, Flows and VLSY-Layout*. Springer, Berlin.

Korte, B., Lovász, L. and Schrader, R. (1991): *Greedoids*. Springer, Berlin.

Korte, B., Prömel, H. J. and Steger. A. (1990): Steiner trees in VLSI-Layout. In: *Paths, Flows and VLSI-layout* (Eds. B. Korte, L. Lovász, H. J. Prömel and A. Schrijver). Springer, Berlin, pp. 185-214.

Kruskal, J. B. (1956): On the shortest spanning subtree of a graph and the travelling salesman problem. *Proc. AMS* **7**, 48-50.

Kuhn, H. W. (1955): The hungarian method for the assignment problem. *Naval Res. Logistic Quart.* **2**, 83-97.

Kuhn, H. W. (1956): Variants of the hungarian method for the assignment problem. *Naval Res. Logistic Quart.* **3**, 253-258.

Lawler, E. L. (1976): *Combinatorial Optimization: Networks and Matriods.* Holt, Rinehart and Winston, New York.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G. and Shmoys, D. B. (1993): Sequencing and scheduling: Algorithms and complexity. In: *Logistics of production and inventory* (Eds. S. C. Graves, A. H. G. Rinnooy Kan and P. H. Zipkin). Elsevier, Amsterdam, pp. 445-522.

Lengauer, T. (1990): *Combinatorial Algorithms for Integrated Circuit Layout.* Wiley, New York.

Lomonosov, M. V. (1985): Combinatorial approaches to multiflow problems. *Discr. Appl. Math.* **11**, 1-93.

Lovász, L. and Plummer, M. D. (1986): *Matching Theory.* North Holland, Amsterdam.

Lovetskiy, S.Y. and Melamed, I.I. (1986): Dynamic flows in networks. *Upravlenie v Transportnîx sistemax ,* 7-29.

Magnanti, T. L. and Wong, R. T. (1984): Network design and transportation planning: models and algorithms. *Transportation Sci.* **18**, 1-55.

Malhotra, V. M., Kumar, M. P. and Mahaswari, S. N. (1978): An $O(|V|^3)$ algorithm for finding maximum flows in networks. *Inform. Proc. Letters.* **7**, 277-278.

Matsui, T. (1995): The minimum spanning tree problem on a planar graph. *Discr. Appl. Math.* **58**, 91-94.

Mehlhorn, K. (1984): *Data Structures and Algorithms.* Springer, Berlin.

Michalewicz, Z. (1992): *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, Berlin.

Minieka, E. (1973): Maximal, lexicographic and dynamic network flows. *Oper. Res.,* vol. 12, **no. 2,** 517-527.

Minty, G. J. (1960): Monotone networks. *Proc. Royal Soc. London (A)* **257**, 194-212.

Minty, G. J. (1966): On the axiomatic foundations of the theories of directed linear graphs, electrical networks and network programming. *J. Math. Mech.* **15**, 485-520.

Monien, B. (1983): The complexity of determining a shortest cycle of even length. *Computing* **31**, 355- 369.

Moore, E. F. (1959): The shortest path through a maze. *Proc. Int. Symp. on Theory of Switching Part II*, Cambridge, Mass., Harvard University Press, pp. 285-292.

Murty, K. G. (1992): *Network Programming.* Prentice Hall, Englewood Cliff, NJ.

Nemhauser, G. L. and Wolsey, L. A. (1988): *Integer and Combinatorial Optimization*. Wiley, New York.

Nishizeki, T. and Chiba, N. (1988): *Planar Graphs: Theory and Algorithms*. North Holland, Amsterdam.

Oellerman, O. R. (1996): Connectivity and edge-connectivity in graphs: a survey. *Congr. Numer.* **116**, 231-252.

Orlin, J. B. (1993): A faster strongly polynomial minimum cost flow algorithm. *Oper. Res.* **41**, 338-350.

Orlin. J. B. and Ahuja, R. K. (1992): New scaling algorithms for the assignment and minmum cycle mean problems. *Math. Progr.* **54**, 41-56.

Orlin, J. B., Plotkin, S. A. and Tardos, E. (1993): Polynomial dual network simplex algorithms. *Math. Progr.* **60**, 255-276.

Oxley, J. G. (1992): *Matroid Theory*. Oxford University Press, Oxford.

Padberg, M. W. and Rinaldi, G. (1991): A branch-and-cut algorithm for the resolution of large-scale travelling salesman problems. *SIAM Rev.* **33**, 60-100.

Papadimitriou, C. H. (1976): On the complexity of edge traversing. *J. ACM* **23**, 544- 554.

Papadimitriou, C. H. and Steiglitz, K. (1982): *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, N.J.

Papadimitriou, C. H. and Yannakakis, M. (1982): The complexity of restricted spanning tree problems.*J. ACM* **29**, 285-309.

Papadimitriou, C. H. and Yannakakis, M. (1993): The traveling salesman problem with distances 1 and 2. *Math. of Oper. Res.* **18**, 1-11.

Pape, U. and Conradt, D. (1980): Maximales Matching in Graphen. In: *Ausgewählte Operations Research Software in FORTRAN* (Ed. H. Späth). Oldenbourg, München, pp. 103-114.

Plummer, M. D. (1994): Extending matchings in graphs: a survey. *Discr. Math.* **127**, 277-292.

Plummer, M. D. (1996): Extending matchings in graphs: an update. *Congr. Numer.* **116**, 3-32.

Potts, R. B. and Oliver, R. M. (1972): *Flow in Transportation Networks.* Academic Press, New York.

Powell, W.B., Sheffi, Y. and Thiriez (1984): The dynamic vehicle allocation problem with uncertain demands . *9th Int. Symp. on Transportation and Traffic Theory.*

Prim, R. C. (1957): Shortest connection networks and some generalizations. *Bell Systems Techn. J.* **36**, 1389-1401.

Prisner, E. (1996): Line graphs and generalizations - a survey. *Congr. Numer.* **116**, 193-229.

Qi, L. (1988): Directed submodularity, ditroids and directed submodular flows. *Math. Progr.* **42**, 579-599.

Radzik, T. and Goldberg, A. V. (1991): Tight bounds on the number of minimum mean cycle cancellations and related results. *Proc.* 2^{nd} *ACM-SIAM Symp. on Discrete Algorithms*, pp. 110-119.

Ramachandra Rao, A. (1968): An extremal problem in graph theory. *Israel J. Math.* **6**, 261-266.

Recski, A. (1989): *Matroid Theory and its Applications.* Springer, Berlin.

Reinelt, G. (1994): *The traveling salesman.* Springer, Berlin.

Rühe, G. (1991): *Algorithmic Aspects of Flows in Networks.* Kluwer Academic Publishers, Dordrecht.

Schreuder, J. A. M. (1980): Constructing timetables for sport competitions. *Math. Progr. Studies* **13**, 58-67.

Schreuder, J. A. M. (1992): Combinatorial aspects of construction of competition Dutch professional football leagues. *Discr. Appl. Math.* **35**, 301-312.

Schrijver, A. (1986): *Theory of Integer and Linear Programming.* Wiley, New York.

Seymour, P. (1979): Sums of circuits. In: *Graph Theory and related topics.* (Eds. J. A. Bondy and U. S. R. Murty). Academic Press, New York, pp. 341-355.

Shapiro, J. F. (1979): A survey of Lagrangian techniques for discrete optimization. *Ann. Discr. Math.* **5**, 113-138.

Shiloach, Y. and Vishkin (1982): An $O(n^2 \log n)$ parallel max-flow algorithm. *Journal of Algorithms* **3**, 362-391.

Sleator, D. D. (1980): *An* $O(mn \log n)$ *algorithm for maximum network flow.* Ph. D. thesis, Standford University.

Sleator, D. D. and Tarjan, R. E. (1983): A data structure for dynamic trees.*J. Comput. System Sci.* **26**, 362-391.

Syslo, M. M., Deo, N. and Kowalik, J. S. (1983): *Discrete Optimization Algorithms*. Prentice Hall, Englewood Cliffs, N. J.

Takács, L. (1990a): On Cayley's formula for counting forests. *J. Comb. Th. (A)* **53**, 321-323.

Takács, L. (1990b): On the number of distinct forests. *SIAM J. Discr. Math.* **3**, 574-581.

Takaoka, T. (1992): A new upper bound on the complexity of all the pairs shortest path problem. *Inf. Process. Lett.* **43**, 195-199.

Tardos, E. (1985): A strongly polynomial minimum cost circulation algorithm. *Combinatorica* **5**, 247-255.

Tardos, E. (1986): A strongly polynomial algorithm to solve combinatorial linear programs. *Oper. Res.* **34**, 250- 256.

Tarjan, R. E. (1972): Depth first search and linear graph algorithms. *SIAM J. Comp.* **1**, 146-160.

Tarjan, R. E. (1977): Finding optimum branchings. *Networks* **7**, 25-35.

Tarjan, R. E. (1983): *Data Structures and Network Algorithms*. SIAM, Philadelphia.

Tarjan, R. E. (1984): A simple version of Karzanov's blocking flow diagram. *Oper. Res. Letters* **2**, 265-268.

Tarry, G. (1895): Le problème des labyrinthes. *Nouv. Ann. de Math.* **14**, 187.

Tassiulas, L. (1997): Worst case length of nearest neighbor tours for the euclidean traveling salesman problem. *SIAM J. Discr. Math.* **10**, 171-179.

Terlaky, T. (1996): *Interior Point Methods of Mathematical Programming*. Kluwer, Dordrecht.

Tomescu, I. (1981): *Probleme de combinatorică şi teoria grafurilor*. Editura Didactică şi Pedagogică, Bucureşti.

Tomescu, I. (1997): *Data structures*. Editura Universităţii din Bucureşti, Bucureşti.

Tutte, W. T. (1984): *Graph Theory*. Cambridge University Press, Cambridge.

Vazirani, V. V. (1994): A theory of alternating paths and blossoms for proving correctness of the $O(V^{\frac{1}{2}}E)$ general graph matching algorithm. *Combinatorica* **14**, 71- 109.

Vemuganti, R.R., Oblak, M. and Aggarval, A. (1989): Network models for fleet management. *Decision Sciences,* vol. 20, **no. 1,** 182-197.

Warshall, S. (1962): A theorem on Boolean matrices. *J. ACM.* **9**, 11-12.

Weintraub, A. (1974): A primal algorithm to solve network flow problems with convex costs. *Management Sc.* **21**, 87-97.

White, N., Ed. (1986): *Theory of Matroids*. Cambridge University Press, Cambridge.

White, N., Ed. (1992): *Matroid Applications*. Cambridge University Press, Cambridge.

Wilkinson, W.L. (1971): An algorithm for universal maximal dynamic flows in a network . *Oper. Res.,* vol. 19, **no. 7,** 1602-1612.

Wilkinson, W.L. (1973): Min/max bounds for dynamic networks flows. *Naval Research Logistic Quaterly,* vol. 20, **no. 3,** 505-516.

Wilson, R. J. (1986): An Eulerian trail through Königsberg. *J. Graph Th.* **10**, 265-275.

Wilson, R. J. (1989): A brief history of Hamiltonian graphs. *Ann. Discr. Math.* **41**, 487-496.

Wirth, N. (1976): *Algorithms + Data Structures = Programs*. Prentice Hall, Englewood Cliffs, N. J.

Yao, A. C. (1975): An $O(|E| \log\log |V|)$ algorithm for finding minimum spanning trees. *Inform. Proc. Letters* **4**, 21-23.

Yap, H. P. (1986): *Some Topics in Graph Theory*. Cambridge University Press, Cambridge.

Young, N. E., Tarjan, R. E. and Orlin, J. B. (1991): Faster parametric shortest path and minimum-balance algorithms. *Networks* **21**, 205-221.

Yuster, R. and Zwick, U. (1997): Finding even cycles even faster. *SIAM J. Discr. Math.* **10**, 209-222.

Zahorik, A., Thomas, L.J. and Trigeiro, W. (1984): Network programming models for production scheduling in multi-stage, multi-item, capacitated systems. *Management Science,* vol. 30, **no. 3,** 308-325.

Zadeh, N. (1972): Theoretical efficiency of the Edmonds-Karp algorithm for computing maximal flows.*J. ACM* **19**, 248-264.

Zadeh, N. (1973a): A bad network problem for the simplex method and other minimum cost flow algorithms. *Math. Progr.* **5**, 255-266.

Zadeh, N. (1973b): More pathological examples for network flow problems. *Math. Progr.* **5**, 217-224.

Zimmermann, U. (1981): *Linear and Combinatorial Optimization in Ordered Algebraic Structures.* North Holland, Amsterdam.

Zuckerman, D. (1996): On unapproximable versions of NP-complete problems. *SIAM J. Computing* **25**, 1293-1304.