

UNIVERSITATEA TRANSILVANIA DIN BRAŞOV  
FACULTATEA DE MATEMATICĂ INFORMATICĂ

**ALGORITMI ÎN OPTIMIZARE COMBINATORIE 1**  
**CURS LA MASTERAT**

TITULAR: PROF. DR. CIUREA ELEONOR



# Cuprins

<b>1 Fluxuri maxime în rețele</b>	<b>1</b>
1.1 Noțiuni introductive . . . . .	1
1.2 Ipoteze asupra problemei fluxului maxim . . . . .	7
1.3 Fluxuri și tăieturi . . . . .	9
1.4 Algoritmi pseudopolinomiali . . . . .	11
1.4.1 Algoritmul generic . . . . .	11
1.4.2 Algoritmul Ford-Fulkerson de etichetare . . . . .	15
1.5 Aplicații ale teoremei flux max și tăietură min . . . . .	19
1.5.1 Conexitatea rețelei . . . . .	19
1.5.2 Cuplaje și acoperiri . . . . .	20
1.6 Fluxuri cu margini inferioare pozitive . . . . .	22
1.7 Aplicații . . . . .	28
1.7.1 Probleme de transport . . . . .	28
1.7.2 Un caz particular al problemei de afectare . . . . .	28
1.7.3 Planificarea lucrărilor pe mașini paralele . . . . .	29
<b>2 Algoritmi polinomiali pentru fluxul maxim</b>	<b>31</b>
2.1 Algoritmi polinomiali cu drumuri de mărire a fluxului . . . . .	31
2.1.1 Etichete distanță . . . . .	31
2.1.2 Algoritmul Gabow al scalării bit a capacitații . . . . .	32
2.1.3 Algoritmul Ahuja-Orlin al scalării maxime a capacitații . . . . .	34
2.1.4 Algoritmul Edmonds - Karp al drumului celui mai scurt . . . . .	36
2.1.5 Algoritmul Ahuja - Orlin al drumului celui mai scurt . . . . .	37
2.1.6 Algoritmul Dinic al rețelelor stratificate . . . . .	43
2.1.7 Algoritmul Ahuja - Orlin al rețelelor stratificate . . . . .	46
2.2 Algoritmi polinomiali cu prefluxuri . . . . .	51
2.2.1 Algoritmul preflux generic . . . . .	51
2.2.2 Algoritmul preflux FIFO . . . . .	56
2.2.3 Algoritmul preflux cu eticheta cea mai mare . . . . .	59
2.2.4 Algoritmul de scalare a excesului . . . . .	62
2.3 Aplicații . . . . .	65
2.3.1 Problema reprezentanților . . . . .	65
2.3.2 Problema rotunjirii matricelor . . . . .	66

<b>3 Fluxuri minime în rețele</b>	<b>69</b>
3.1 Noțiuni introductive . . . . .	69
3.2 Algoritmi pseudopolinomiali pentru fluxul minim . . . . .	71
3.2.1 Algoritmul generic . . . . .	71
3.2.2 Varianta algoritmului Ford-Fulkerson . . . . .	74
3.3 Algoritmi polinomiali pentru fluxul minim . . . . .	77
3.3.1 Noțiuni introductive . . . . .	77
3.3.2 Algoritmi polinomiali cu drumuri de micsorare . . . . .	78
3.3.3 Algoritmi polinomiali cu prefluxuri . . . . .	82
3.4 Algoritmul minimax . . . . .	95
3.5 Aplicații . . . . .	97
3.5.1 Problema planificării lucrărilor . . . . .	97
<b>Bibliografie</b>	<b>99</b>

# Capitolul 1

## Fluxuri maxime în rețele

### 1.1 Notiuni introductive

Fie  $G = (N, A)$  un digraf definit prin mulțimea  $N$  cu  $n$  noduri și mulțimea  $A$  cu  $m$  arce. Se definesc funcțiile: capacitatea  $c : A \rightarrow \mathbb{R}^+$ , marginea inferioară  $l : A \rightarrow \mathbb{R}^+$  și funcția valoare  $v : N \rightarrow \mathbb{R}$ . 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 și numărul  $v(x)$  desemnează valoarea nodului  $x$ . Problema fluxului în rețea  $G = (N, A, l, c)$  constă în a determina funcția flux  $f : A \rightarrow \mathbb{R}^+$  care verifică constrângerile:

$$\sum_y f(x, y) - \sum_y f(y, x) = v(x), \quad x \in N \quad (1.1.a)$$

$$l(x, y) \leq f(x, y) \leq c(x, y), \quad (x, y) \in A, \quad (1.1.b)$$

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

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

$$\bar{M}f = v \quad (1.2.a)$$

$$l \leq f \leq c \quad (1.2.b)$$

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

Constrângerile (1.1.a) se numesc *constrângerile de conservare a fluxului*. Primul termen al constrângerii corespunzătoare nodului  $x$  reprezintă fluxul total de ieșire din nodul  $x$  și 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 (1.1.b) se numesc *constrângerile de mărginire a fluxului*.

În problema (1.1) înlocuim variabila  $f(x, y)$  prin  $f'(x, y) + l(x, y)$  și obținem problema:

$$\sum_y f'(x, y) - \sum_y f'(y, x) = v'(x), \quad \forall x \in N \quad (1.3.a)$$

$$0 \leq f'(x, y) \leq c'(x, y), \quad \forall (x, y) \in A \quad (1.3.b)$$

unde

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

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 restrângere generalitatea, altfel spus considerăm problema fluxului în rețeaua  $G = (N, A, c)$ . Cazul cu  $l(x, y) \geq 0$ ,  $(x, y) \in A$  este studiat în paragraful 6.

În mulțimea fluxurilor în rețeaua  $G = (N, A, c)$  se introduc relația de ordine parțială și operația de adunare (scădere). Presupunem că mulțimea arcelor  $A = \{a_1, \dots, a_m\}$  este ordonată după o anumită ordine.

**Definiția 1.1.** Fie  $f_1 = (f_1(a_1), \dots, f_1(a_m))$  și  $f_2 = (f_2(a_1), \dots, f_2(a_m))$  două fluxuri în rețeaua  $G = (N, A, c)$ . Se spune că fluxul  $f_1$  este *mai mic* decât fluxul  $f_2$  și scriem  $f_1 < f_2$  dacă  $f_1(a_i) \leq f_2(a_i)$  pentru  $i = 1, \dots, m$  și există cel puțin un arc  $a_k$  astfel încât  $f_1(a_k) < f_2(a_k)$ . Se spune că  $f_3 = (f_3(a_1), \dots, f_3(a_m))$  este *suma* fluxurilor  $f_1$  și  $f_2$  dacă  $f_3(a_i) = f_1(a_i) + f_2(a_i)$  pentru  $i = 1, \dots, m$ . Analog se definește  $f_4 = f_2 - f_1$ .

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

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

**Definiția 1.2.** Un flux  $f$  în rețeaua  $G = (N, A, c)$  se numește *flux elementar* dacă  $f = g(D)$  sau  $f = \overset{\circ}{g}(\overset{\circ}{D})$ .

Fie  $\mathcal{D}$  mulțimea drumurilor de la nodurile sursă la nodurile stoc și  $\overset{\circ}{\mathcal{D}}$  mulțimea circuitelor în rețeaua  $G = (N, A, c)$ . Un flux în rețeaua  $G = (N, A, c)$  poate fi definit în două moduri: putem defini fluxurile pe arce și putem defini fluxurile elementare pe drumuri și circuite. În formularea flux arc variabilele sunt fluxurile  $f(x, y)$  care verifică constrângerile (1.1). În formularea drum - circuit variabilele sunt fluxurile elementare  $g(D)$ ,  $D \in \mathcal{D}$  și  $\overset{\circ}{g}(\overset{\circ}{D})$ ,  $\overset{\circ}{D} \in \overset{\circ}{\mathcal{D}}$ .

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

**Teorema 1.1. (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 proprietăți:*

(a) fiecare drum cu flux nenul conectează un nod sursă la un nod stoc;

(b) cel mult  $n + m$  drumuri și circuite au flux nenul; dintre acestea cel mult  $m$  circuite au flux nenul.

**Demonstrație.** Pentru afirmația directă definim:

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

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

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

Astfel fiecare flux drum-circuit determină fluxul arc unic.

Pentru afirmația inversă dăm o demonstrație 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; f1 := f;
(4)       WHILE fk > 0 DO
(5)           BEGIN
(6)               IF există nod sursă
(7)                   THEN BEGIN
(8)                       se selectează un nod sursă x0; Dk := {x0}; i := 0;
(9)                   END
(10)              ELSE BEGIN
(11)                  se determină un arc (x0, x1) ∈ A cu fk(x0, x1) > 0;
(12)                  Dk := {x0, x1}; i := 1;
(13)              END;
(14)              WHILE xi nu este nod stoc DO
(15)                  BEGIN
(16)                      se determină un arc (xi, xi+1) ∈ A cu fk(xi, xi+1) > 0;
(17)                      i := i + 1;
(18)                      IF xi ∉ Dk
(19)                          THEN Dk := Dk ∪ {xi}
(20)                      ELSE EXIT
(21)                  END;
(22)                  IF xi este nod stoc
(23)                      THEN BEGIN
(24)                          r := min{v(x0), -v(xi), min{fk(x, y)|(x, y) ∈ Dk}};
(25)                          v(x0) := v(x0) - r; v(xi) := v(xi) + r;
(26)                          FOR (x, y) ∈ A DO
(27)                              IF (x, y) ∈ Dk
(28)                                  THEN gk(x, y) := r
(29)                                  ELSE gk(x, y) := 0;
(30)                          fk+1 := fk - g(Dk);

```

```

(29)          END;
(30)      ELSE BEGIN
(31)           $\overset{\circ}{D}_k := \{x_i, \dots, x_i\}; \overset{\circ}{r} := \min\{f_k(x, y) \mid (x, y) \in \overset{\circ}{D}_k\};$ 
(32)          FOR  $(x, y) \in A$  DO
(33)              IF  $(x, y) \in \overset{\circ}{D}_k$ 
(34)                  THEN  $\overset{\circ}{g}_k(x, y) := \overset{\circ}{r}$ 
(35)                  ELSE  $\overset{\circ}{g}_k(x, y) := 0;$ 
(36)               $f_{k+1} := f_k - g(\overset{\circ}{D}_k);$ 
(37)          END;
(38)           $k := k + 1;$ 
(39)      END;
(40)  END.

```

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

**Teorema 1.2. (Teorema descompunerii circulației).** *Oricare circulație  $f$  poate fi reprezentată ca flux circuit de-a lungul a cel mult  $m$  circuite cu flux nenul.*

**Demonstrație.** Dacă  $v(x) = 0$  pentru toate nodurile  $x$  din  $N$ , atunci algoritmul descompunerii fluxului determină la fiecare iterație un flux circuit și conform Teoremei 1.1, cel mult  $m$  circuite au flux nenul. ■

**Teorema 1.3. (Teorema de complexitate a algoritmului descompunerii fluxului).** *Algoritmul descompunerii fluxului are complexitatea  $O(m^2)$ .*

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

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

**Exemplul 1.1.** Să ilustrăm algoritmul descompunerii fluxului pentru fluxul din rețeaua reprezentată în figura 1.1.

Deci  $f = (f(1, 2), f(1, 3), f(2, 4), f(3, 2), f(3, 4), f(4, 5), f(4, 6), f(5, 3), f(5, 6)) = (2, 0, 5, 3, 4, 6, 3, 7, 2)$  și  $v = (v(1), v(2), v(3), v(4), v(5), v(6)) = (2, 0, 0, 0, 3, -5)$ . Inițial  $f_1 = f$ . Dacă algoritmul selectează nodul sursă 1 și determină drumul  $D_1 = (1, 2, 4, 5, 6)$  cu

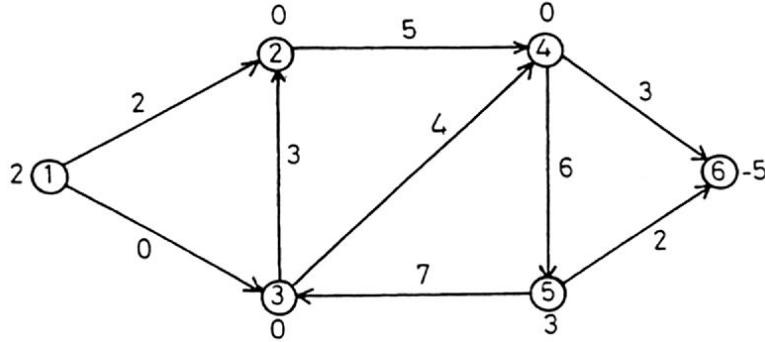


Fig.1.1

$g(D_1) = (2, 0, 2, 0, 0, 2, 0, 0, 2)$ , atunci  $f_2 = (0, 0, 3, 3, 4, 4, 3, 7, 0)$  și  $v(1) = 0, v(6) = -3$ . La următoarea iterare algoritmul selectează nodul sursă 5 și 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 iterare nu mai există noduri sursă ( $v(x) = 0$ ) și presupunem că algoritmul determină arcul  $(3,4)$  cu  $f_3(3,4) > 0$  și 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  $f_4 = (0, 0, 0, 0, 0, 0, 0, 0, 0)$  și algoritmul se termină. Această descompunere nu este unică.

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

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

Dacă mulțimea nodurilor sursă  $S$  are un singur nod îl notăm cu  $s$  și dacă mulțimea 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 rețeaua  $G = (N, A, c)$  constă în a determina funcția flux  $f$  care verifică constrângerile:

$$\text{Maximizează } v \quad (1.4.a)$$

$$\sum_y f(x, y) - \sum_y f(y, x) = \begin{cases} v(x), & x \in S, \\ 0, & x \in R \\ v(x) & x \in T, \end{cases} \quad (1.4.b)$$

$$0 \leq f(x, y) \leq c(x, y), \quad (x, y) \in A. \quad (1.4.c)$$

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

$$\text{Maximizează } v \quad (1.5.a)$$

$$\sum_y f(x, y) - \sum_y f(y, x) = \begin{cases} v, & x = s, \\ 0, & x \neq s, t \\ -v & x = t, \end{cases} \quad (1.5.b)$$

$$0 \leq f(x, y) \leq c(x, y), (x, y) \in A. \quad (1.5.c)$$

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

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

$$\begin{aligned} \tilde{N} &= \tilde{N}_1 \cup \tilde{N}_2 \cup \tilde{N}_3, \quad \tilde{N}_1 = \{\tilde{s}\}, \quad \tilde{N}_2 = N, \quad \tilde{N}_3 = \{\tilde{t}\}, \\ \tilde{A} &= \tilde{A}_1 \cup \tilde{A}_2 \cup \tilde{A}_3, \quad \tilde{A}_1 = \{(\tilde{s}, x) \mid x \in S\}, \quad \tilde{A}_2 = A, \quad \tilde{A}_3 = \{(x, \tilde{t}) \mid x \in T\}, \\ \tilde{c}(\tilde{s}, x) &= \infty, \quad (\tilde{s}, x) \in \tilde{A}_1, \quad \tilde{c}(x, y) = c(x, y), \quad (x, y) \in \tilde{A}_2, \quad \tilde{c}(x, \tilde{t}) = \infty, \quad (x, \tilde{t}) \in \tilde{A}_3. \end{aligned}$$

**Teorema 1.4. (Caracterizarea problemei fluxului intr-o rețea cu surse și stocuri multiple).** Problema fluxului maxim de la  $S$  la  $T$  în rețeaua  $G = (N, A, c, S, T)$  este echivalentă cu problema fluxului maxim de la  $\tilde{s}$  la  $\tilde{t}$  în rețeaua  $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{c}, \tilde{s}, \tilde{t})$ .

**Demonstrație.** Dacă  $f$  este un flux maxim cu valoarea  $v$  de la  $S$  la  $T$  în rețeaua  $G$ , atunci se poate defini aplicația  $\tilde{f} : \tilde{A} \rightarrow \mathbb{R}$  în modul următor:

$$\begin{aligned} \tilde{f}(\tilde{s}, x) &= v(x), \quad (\tilde{s}, x) \in \tilde{A}_1, \quad \tilde{f}(x, y) = f(x, y), \quad (x, y) \in \tilde{A}_2, \quad \tilde{f}(x, \tilde{t}) = -v(x), \\ (x, \tilde{t}) &\in \tilde{A}_3. \end{aligned}$$

Astfel  $\tilde{f}$  verifică constrângările

Maximizează  $\tilde{v}$

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

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

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

Reciproc, dacă  $\tilde{f}$  este un flux maxim de valoare  $\tilde{v}$  în rețeaua  $\tilde{G}$ , atunci restricția sa  $f : A \rightarrow \mathbb{R}$  este evident un flux maxim cu valoarea  $v = \tilde{v}$  în rețeaua  $G$ . ■

Problema determinării unui flux  $f$  care verifică (1.4.b) și (1.4.c) cu valorile  $v(x)$  precizate se numește *problema ofertei și cererii*. În acest caz un nod  $x \in S$  se numește *nod ofertă* și un nod  $x \in T$  se numește *nod cerere*.

În cazul problemei ofertei și cererii se construiește rețeaua extinsă  $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{c}, \tilde{s}, \tilde{t})$  plecând de la rețeaua  $G = (N, A, c, S, T)$ , cu  $S$  mulțimea nodurilor ofertă și  $T$  mulțimea nodurilor cerere, ca mai sus, cu deosebirea că funcția capacitate  $\tilde{c}$  se definește în modul următor:  $\tilde{c}(\tilde{s}, x) = v(x), \quad (\tilde{s}, x) \in \tilde{A}_1, \quad \tilde{c}(x, y) = c(x, y), \quad (x, y) \in \tilde{A}_2, \quad \tilde{c}(x, \tilde{t}) = -v(x), \quad (x, \tilde{t}) \in \tilde{A}_3$ .

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

**Teorema 1.5. (Caracterizarea problemei ofertă-cerere).** În rețeaua  $G = (N, A, c, S, T)$  există un flux care verifică (1.4 b) și (1.4 c) dacă și numai dacă în rețeaua  $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{c}, \tilde{s}, \tilde{t})$  există un flux maxim care saturează toate arcele din  $\tilde{A}_1$  și  $\tilde{A}_3$ .

**Demonstrație.** Se demonstrează analog ca Teorema 1.4. ■

**Exemplul 1.2.** Se consideră rețeaua  $G = (N, A, c, S, T)$  din figura 1.2, unde  $S = \{1, 2\}$ ,  $R = \{3, 4, 5\}$ ,  $T = \{6, 7\}$  și pe fiecare arc s-a trecut capacitatea.

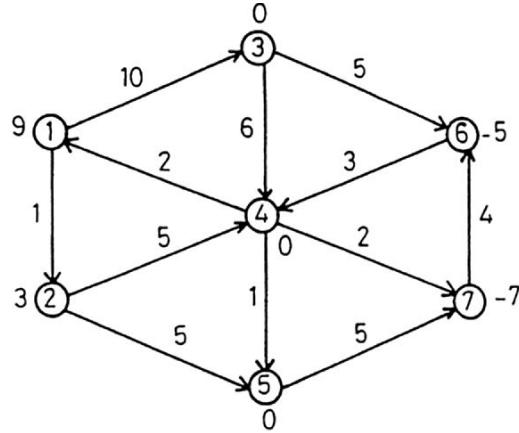


Fig.1.2

Rețeaua extinsă  $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{c}, \tilde{s}, \tilde{t})$  este reprezentată în figura 1.3, unde  $\tilde{s} = 0$ ,  $\tilde{t} = 8$ . Fluxul maxim, reprezentat de primul număr de pe fiecare arc al rețelei  $\tilde{G}$ , saturează arcele din  $\tilde{A}_1 = \{(0, 1), (0, 2)\}$  și  $\tilde{A}_3 = \{(6, 8), (7, 8)\}$ . Deci problema ofertă-cerere are soluție și este restricția funcției flux  $\tilde{f} : \tilde{A} \rightarrow \mathbb{R}$  la multimea  $A$ .

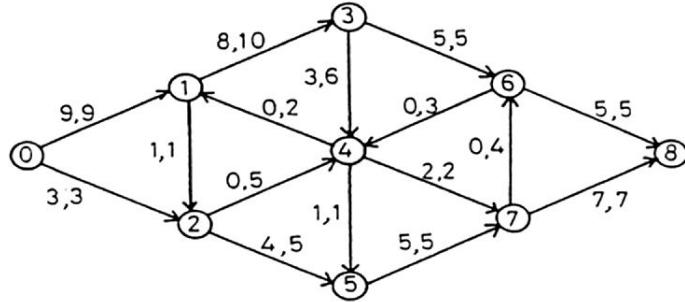


Fig.1.3

## 1.2 Ipoteze asupra problemei fluxului maxim

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

**Ipoteza 1.** Rețeaua este orientată.

Unele probleme de flux sunt formulate pe rețele neorientate sau mixte (care conțin arce și muchii). În acest caz, problema poate fi rezolvată pe o rețea orientată echivalentă în modul următor.

O muchie  $[x, y]$  cu capacitatea  $c[x, y]$  permite flux de la nodul  $x$  la nodul  $y$  și de asemenea de la nodul  $y$  la nodul  $x$ . Astfel muchia  $[x, y]$  are constrângerea de mărginire:  $f[x, y] + f[y, x] \leq c[x, y]$  și 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 înlocuiește fiecare muchie  $[x, y]$  prin două arce  $(x, y)$  și  $(y, x)$  cu  $c(x, y) = c(y, x) = c[x, y]$ . Fie  $f$  un flux în rețeaua originară (neorientată sau mixtă). Definim fluxul  $f'$  în rețeaua transformată (orientată) în modul următor:

- dacă fluxul este de la  $x$  la  $y$  atunci  $f'(x, y) = f[x, y]$  și  $f'(y, x) = 0$ ;
- dacă fluxul este de la  $y$  la  $x$  atunci  $f'(x, y) = 0$  și  $f'(y, x) = f[y, x]$ .

Invers, fie  $f'$  un flux în rețeaua transformată. Definim fluxul  $f$  în rețeaua originară în modul următor:

- dacă  $f'(x, y) - f'(y, x) > 0$ , atunci  $f[x, y] = f'(x, y) - f'(y, x)$  și  $f[y, x] = 0$ ;
- dacă  $f'(x, y) - f'(y, x) = 0$ , atunci  $f[x, y] = 0$  și  $f[y, x] = 0$ ;
- dacă  $f'(x, y) - f'(y, x) < 0$ , atunci  $f[x, y] = 0$  și  $f[y, x] = f'(y, x) - f'(x, y)$ .

Evident că fluxul  $f$  verifică constrângerile de mărginire pe muchii și condiția că fluxul este permis numai în unul din cele două sensuri.

**Exemplul 1.3.** Se consideră rețeaua mixtă  $G = (N, A, c)$  reprezentată în figura 1.4, unde  $s = 1$ ,  $t = 6$  și al doilea număr de pe fiecare arc sau muchie reprezintă capacitatea.

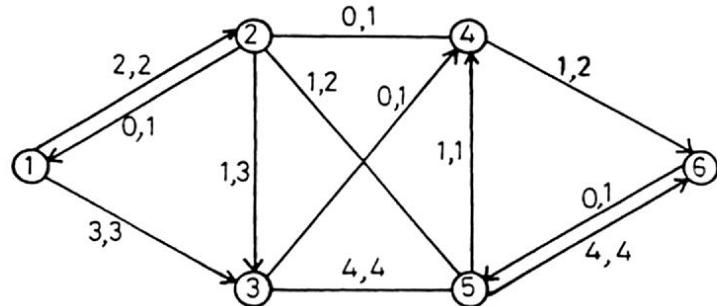


Fig.1.4

Presupunem că fluxul inițial în rețeaua mixtă  $G$  este fluxul nul și dorim să determinăm un flux maxim. Corespunzător fluxului nul și capacitaților arc din rețeaua mixtă  $G$  se obține rețeaua orientată  $G' = (N', A', c')$  cu fluxul nul și capacitațile pe fiecare arc sunt reprezentate de al doilea număr. Rețeaua  $G'$  este reprezentată în figura 1.5. Utilizând orice algoritm pentru determinarea unui flux maxim într-o rețea orientată se obține fluxul maxim  $f'$  reprezentat de primul număr de pe fiecare arc al rețelei  $G'$ . Fluxul maxim  $f$  din rețeaua mixtă  $G$  este următorul:  $f[2, 4] = 0$ ,  $f[4, 2] = 0$ ,  $f[2, 5] = 1$ ,  $f[5, 2] = 0$ ,  $f[3, 5] = 4$ ,  $f[5, 3] = 0$  și  $f(x, y) = f'(x, y)$  pentru toate arcele  $(x, y)$  din  $A$ . Acest flux este reprezentat de primul număr de pe fiecare arc sau muchie din rețeaua mixtă  $G$ .

**Ipoteza 2.** Toate capacitațile sunt întregi nenegativi.

Această ipoteză este cunoscută și sub numele de ipoteza integralității pentru capacitațile de arc. Ipoteza integralității este necesară numai pentru algoritmii a căror

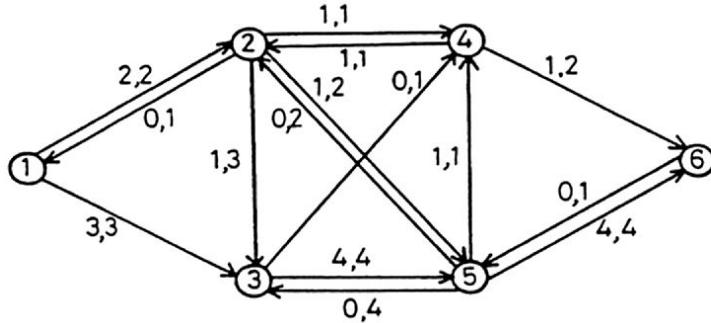


Fig.1.5

complexitate depinde de  $\bar{c}$ . În realitate, acestă ipoteză nu este restrictivă, deoarece toate calculatoarele memorează capacitatele ca numere raționale și putem întotdeauna transforma numerele raționale, prin multiplicarea lor printr-un număr corespunzător de mare, în numere întregi.

**Ipoteza 3.** Rețeaua nu conține nici un drum de la nodul sursă  $s$  la nodul stoc  $t$  alcătuit numai din arce cu capacitați 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 și deci valoarea fluxului maxim este nelimitată.

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

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

**Ipoteza 5.** Rețeaua nu conține arce paralele.

Această ipoteză este necesară pentru convenția notațională și nu micșorează generalitatea, deoarece arcele paralele  $a_1 = (x, y), \dots, a_k = (x, y)$  se pot înlocui cu un singur arc  $a = (x, y)$  cu  $c(a) = c(a_1) + \dots + c(a_k)$ .

### 1.3 Fluxuri și tăieturi

Conceptul de *rețea reziduală* joacă un rol central în dezvoltarea tuturor algoritmilor de flux maxim pe care îi vom prezenta. De aceea, în continuare vom defini noțiunea de rețea reziduală. Dat un flux  $f$ , capacitatea reziduală  $r(x, y)$  a oricărui arc  $(x, y) \in A$  este fluxul adițional maxim care poate fi trimis de la nodul  $x$  la nodul  $y$  utilizând arcele  $(x, y)$  și  $(y, x)$ . Reamintim ipoteza din paragraful 1.2: dacă arcul  $(x, y)$  aparține rețelei atunci și arcul  $(y, x)$  aparține rețelei. 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  $(y, x)$  și pe care îl putem micșora pentru a crește fluxul de la nodul  $x$  la nodul  $y$ . În consecință  $r(x, y) = c(x, y) - f(x, y) + f(y, x)$ . Rețeaua reziduală  $\tilde{G}(f)$  este formată din arcele cu capacitațiile reziduale pozitive, adică din arcele  $(x, y)$  cu  $r(x, y) > 0$ . În figura 1.6(b) se prezintă rețeaua reziduală  $\tilde{G}(f)$  corespunzătoare fluxului  $f$  din rețeaua prezentată în figura 1.6(a).

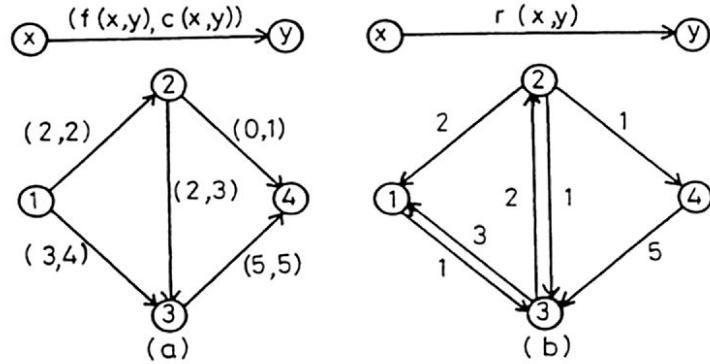


Fig.1.6

Dacă  $X$  și  $Y$  sunt două submulțimi ale mulțimii nodurilor  $N$ , nu în mod necesar disjuncte, definim mulțimea de arce:

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

Pentru orice funcție  $g : N \times N \rightarrow \mathbb{R}^+$  și orice funcție  $h : N \rightarrow \mathbb{R}^+$  definim

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

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

Cu aceste notații constrângerile de conservare (1.5 b) se scriu:

$$f(x, N) - f(N, x) = \begin{cases} v, & x = s, \\ 0, & x \neq s, t \\ -v & x = t \end{cases} \quad (1.5 \text{ } b')$$

În continuare, în funcție de context, vom utiliza constrângerile de conservare a fluxului sub forma (1.5 b) sau (1.5 b').

**Definiția 1.3.** Fie  $G = (N, A, c)$  o rețea și  $X \subset N$ ,  $\bar{X} = N - X$  cu  $X, \bar{X}$  nevide. O tăietură este mulțimea de arce  $[X, \bar{X}] = (X, \bar{X}) \cup (\bar{X}, X)$ , unde  $(X, \bar{X})$  reprezintă mulțimea arcelor directe și  $(\bar{X}, X)$  reprezintă mulțimea arcelor inverse ale acestei tăieturi.

Notăm că *ordinea* în care mulțimile  $X, \bar{X}$  sunt înregistrate în  $[X, \bar{X}]$  este importantă, schimbarea ordinei schimbă mulțimea arcelor directe și mulțimea arcelor inverse între ele. În rețea obținută, după eliminarea mulțimii arcelor directe  $(X, \bar{X})$  ale tăieturii  $[X, \bar{X}]$ , nu există drum de la orice nod  $x \in X$  la orice nod  $\bar{x} \in \bar{X}$ , și dacă ambele mulțimi de arce, directe  $(X, \bar{X})$  și inverse  $(\bar{X}, X)$ , sunt eliminate atunci nu există lanț de la orice nod  $x \in X$  la orice nod  $\bar{x} \in \bar{X}$ . Aceste proprietăți sunt cunoscute ca proprietatea de *blocare a drumului*, respectiv proprietatea de *blocare a lanțului*. Proprietatea de blocare a lanțului produce deconexarea rețelei.

**Definiția 1.4.** O tăietură  $[X, \bar{X}]$  cu  $s \in X$  și  $t \in \bar{X}$  se numește *tăietură*  $s - t$ . Capacitatea  $c[X, \bar{X}]$  a tăieturii  $s - t$   $[X, \bar{X}]$  este definită a fi  $c[X, \bar{X}] = c(X, \bar{X})$ . O tăietură  $s - t$  a cărei capacitate este minimă printre toate tăieturile  $s - t$  se numește *tăietură*  $s - t$  *minimă*. Capacitatea reziduală  $r[X, \bar{X}]$  a tăieturii  $s - t$   $[X, \bar{X}]$  este definită a fi  $r[X, \bar{X}] = r(X, \bar{X})$ ,  $r(X, \bar{X}) = c(X, \bar{X}) - f(X, \bar{X}) + f(\bar{X}, X)$ .

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

**Teorema 1.6. (Teorema valorii fluxului).** În rețeaua  $G = (N, A, c)$  dacă  $f$  este un flux de valoare  $v$  și  $[X, \bar{X}]$  o tăietură  $s - t$ , atunci  $f$  verifică relațiile

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

**Demonstrație.** Deoarece  $f$  este un flux în rețeaua  $G$  el satisfac constrângerile de conservare (1.5. b'). Sumând aceste ecuații pentru  $x \in X$  se obține  $v = f(X, N) - f(N, X)$ . Înlocuind  $N = X \cup \bar{X}$  în această relație și dezvoltând rezultă  $v = f(X, X \cup \bar{X}) - f(X \cup \bar{X}, X) = f(X, X) + f(X, \bar{X}) - f(X, X) - f(\bar{X}, X) = f(X, \bar{X}) - f(\bar{X}, X) = f[X, \bar{X}]$ . De asemenea  $f$  satisfac constrângerile de mărginire a fluxului (1.5 c) și se obține  $f(X, \bar{X}) \leq c(X, \bar{X})$ ,  $f(\bar{X}, X) \geq 0$ . Rezultă  $v = f[X, \bar{X}] = f(X, \bar{X}) - f(\bar{X}, X) \leq c(X, \bar{X}) = c[X, \bar{X}]$ . ■

**Teorema 1.7. (Teorema capacitații reziduale).** În rețeaua  $G = (N, A, c)$  dacă  $f$  este un flux de valoare  $v$ ,  $[X, \bar{X}]$  o tăietură  $s - t$  și  $f'$  este un flux de valoare  $v'$  cu  $v' \geq v$ , atunci

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

**Demonstrație.** Din Teorema 1.6 rezultă că  $v' \leq c(X, \bar{X})$ . Scăzând din această relație, relația  $v = f(X, \bar{X}) - f(\bar{X}, X)$  se obține  $v' - v \leq c(X, \bar{X}) - f(X, \bar{X}) + f(\bar{X}, X) = r(X, \bar{X}) = r[X, \bar{X}]$ . ■

Teorema 1.7 exprimă faptul că într-o rețea  $G = (N, A, c)$  dacă  $f$  este un flux admisibil de valoare  $v$ , atunci valoarea  $\Delta v = v' - v$  a fluxului adițional 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  $s - t$ .

## 1.4 Algoritmi pseudopolinomiali pentru fluxul maxim

### 1.4.1 Algoritmul generic

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

Un LMF  $L$  în raport cu fluxul  $f$  în rețeaua originară  $G$  corespunde la un *drum*  $\tilde{D}$  de mărire a fluxului (DMF) în rețeaua reziduală  $\tilde{G}(f)$  și 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*:  $f'(x, y) = f(x, y) + r(L)$ ,  $(x, y) \in L^+$ ,  $f'(x, y) = f(x, y) - r(L)$ ,  $(x, y) \in L^-$ ,  $f'(x, y) = f(x, y)$ ,  $(x, y) \notin L$  și fluxul  $f'$  are valoarea  $v' = v + r(L)$ .

Cât timp rețeaua reziduală  $\tilde{G}(f)$  conține 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 construiește  $\tilde{G}(f)$ ;
(5)   WHILE  $\tilde{G}(f)$  conține un DMF DO
(6)     BEGIN
(7)       se identifică un DMF  $\tilde{D}$ ;
(8)        $r(\tilde{D}) := \min\{r(x, y) \mid (x, y) \in \tilde{D}\}$ ;
(9)       se execută mărirea de flux;
(10)      se actualizează  $\tilde{G}(f)$ ;
(11)    END;
(12)  END.
```

În implementarea oricărei versiuni a algoritmului generic avem opțiunea să lucrăm fie pe rețeaua originară cu fluxurile  $f(x, y)$ , fie pe rețeaua reziduală cu capacitatele reziduale  $r(x, y)$  și când algoritmul se termină să calculăm fluxurile  $f(x, y)$ . Versiunea prezentată lucrează pe rețeaua reziduală. Să vedem cum putem determina fluxurile  $f(x, y)$  la terminarea algoritmului. În general, mai multe combinații ale lui  $f(x, y)$  și  $f(y, x)$  corespund pentru a verifica relația  $r(x, y) = c(x, y) - f(x, y) + f(y, x)$  și constrângerile de mărginire (1.5 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  $f(x, y) = c(x, y) - r(x, y)$  și  $f(y, x) = 0$ , 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 1.4.** Ilustrăm algoritmul generic pe rețelele din figura 1.7, în care  $s = 1$ ,  $t = 4$ .

Inițial  $f_0 = 0$ . Rețeaua reziduală este rețeaua originară din figura 1.7 (a). Presupunem că algoritmul generic selectează DMF  $\tilde{D} = (1, 3, 4)$  cu  $r(\tilde{D}) = \min\{r(1, 3), r(3, 4)\} = \min\{4, 5\} = 4$ . Mărirea de flux reduce  $r(1, 3) = 4$  la 0 și se elimină arcul  $(1, 3)$  din rețeaua reziduală, crește  $r(3, 1)$  de la 0 la 4 și se adaugă arcul  $(3, 1)$  la rețeaua reziduală. De asemenea, mărirea de flux descrește  $r(3, 4)$  de la 5 la 1, astfel în noua rețea reziduală  $r(3, 4) = 1$  și  $r(4, 3) = 4$ .

Figura 1.7 (b) arată rețeaua reziduală după prima iterare. În a doua iterare, presupunem că algoritmul selectează DMF  $\tilde{D} = (1, 2, 3, 4)$  cu  $r(\tilde{D}) = \min\{2, 3, 1\} = 1$ . Mărirea de flux conduce la rețeaua reziduală arătată în figura 1.7 (c). În a treia iterare, algoritmul generic selectează unicul DMF  $\tilde{D} = (1, 2, 4)$  cu  $r(\tilde{D}) = \min\{1, 1\} = 1$ . Mărirea de flux conduce la rețeaua reziduală arătată în figura 1.7 (d), care nu mai conține DMF.

În continuare se calculează fluxurile pe arce:

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

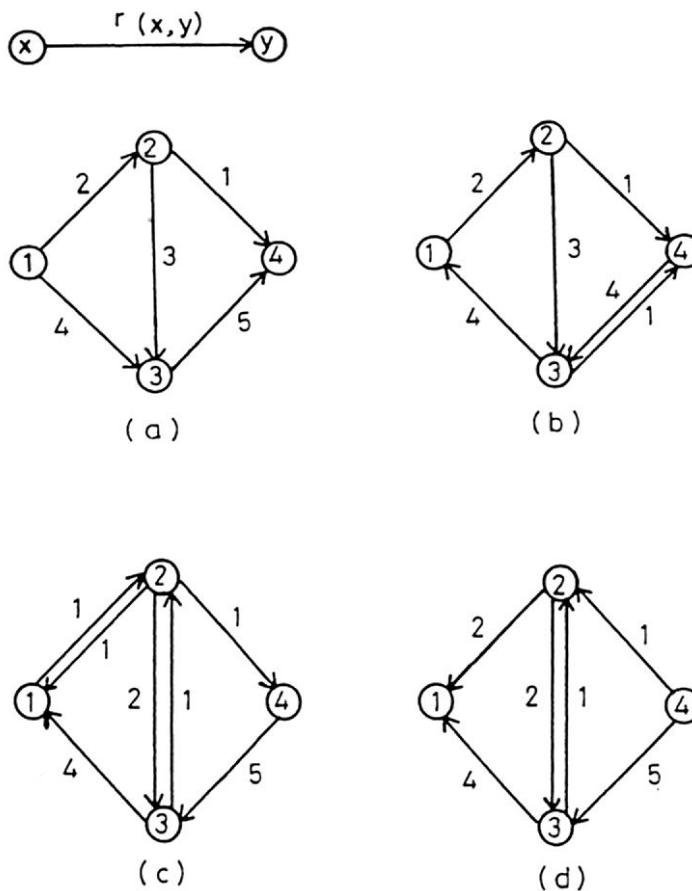


Fig.1.7

**Teorema 1.8. (Teorema valorilor întregi)** Dacă capacitatea  $c$  și fluxul inițial  $f_0$  sunt cu valori întregi atunci algoritm generic converge într-un număr finit de iterații și la terminarea execuției se obține un flux maxim cu valori întregi.

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

În cazul când capacitatea  $c$  și fluxul admisibil inițial  $f_0$  sunt cu valori raționale algoritm generic converge de asemenea într-un număr finit de iterații, deoarece în acest caz problema fluxului maxim poate fi transformată într-o problemă echivalentă în care capacitatea și fluxul admisibil inițial sunt cu valori întregi.

Algoritmul generic este cel mai simplu algoritm pentru rezolvarea problemei fluxului maxim. Empiric, algoritmul generic se prezintă rezonabil de bun. Totuși, în cazul cel mai defavorabil limita pe numărul de iterații nu este în totalitate satisfăcătoare. Edmonds și Karp (1972) au dat un exemplu patologic pentru algoritmul generic ilustrat pe rețea din figura 1.8.

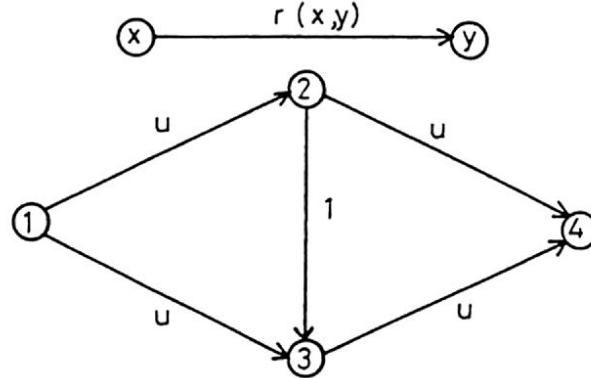


Fig.1.8

Inițial  $f_0 = (f_0(1, 2), f_0(1, 3), f_0(2, 3), f_0(2, 4), f_0(3, 4)) = (0, 0, 0, 0, 0)$ . Evident că fluxul admisibil maxim în această rețea este  $f = (u, u, 0, u, u)$  cu valoarea  $v = 2u$ . Definim secvența de fluxuri admisibile  $f_0, f_1, f_2, \dots, f_{2i-2}, f_{2i-1}, f_{2i}, \dots, f_{2u}$  cu  $f_{2i-1} = (i, i-1, 1, i-1, i)$ ,  $f_{2i} = (i, i, 0, i, i)$  pentru  $i = 1, \dots, u$ . În această secvență  $f_k$  are valoarea  $v_k = k$  pentru  $k = 0, \dots, 2u$ . Pentru  $i = 1$  la  $u$ ,  $f_{2i-1}$  este obținut prin mărirea lui  $f_{2i-2}$  utilizând DMF  $\tilde{D}_{2i-2} = (1, 2, 3, 4)$  cu  $r(\tilde{D}_{2i-2}) = 1$ , din rețeaua reziduală arătată în figura 1.9 (a). Pentru  $i = 1$  la  $u$ ,  $f_{2i}$  este obținut prin mărirea lui  $f_{2i-1}$  utilizând DMF  $\tilde{D}_{2i-1} = (1, 3, 2, 4)$  cu  $r(\tilde{D}_{2i-1}) = 1$  din rețeaua arătată în figura 1.9 (b).

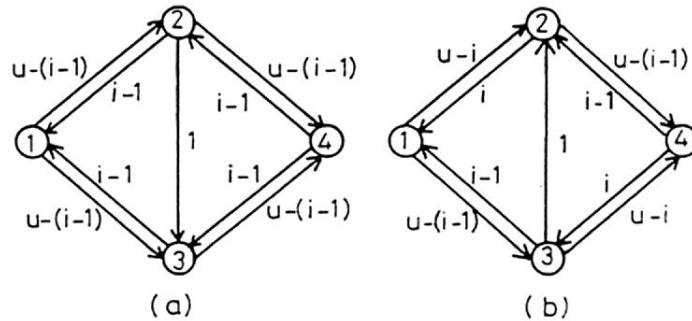


Fig. 1.9. (a) Rețea reziduală în raport cu fluxul  $f_{2i-2}$   
(b) Rețea reziduală în raport cu fluxul  $f_{2i-1}$

Această secvență a fluxurilor admisibile are  $2u+1$  fluxuri. Presupunând că  $u$  este un întreg pozitiv, dimensiunea problemei fluxului maxim pe rețeaua din figura 1.8 (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 secvență, este de  $2u$  iterații. Deci algoritmul generic are o complexitate exponențială în raport cu dimensiunea problemei.

Ford și Fulkerson (1962) au prezentat un exemplu pentru a ilustra faptul că dacă capacitatele sunt numere iraționale și fluxul admisibil initial este cu valori întregi, algoritmul generic poate să nu conveargă într-un număr finit de iterații și la limită poate să conveargă la un flux a cărui valoare este strict mai mică decât valoarea fluxului maxim.

### 1.4.2 Algoritmul Ford-Fulkerson de etichetare

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

```

(1)  PROGRAM FFE;
(2)  BEGIN
(3)     $f := f_0$ ;
(4)    se construiește  $\tilde{G}(f)$ ;
(5)     $\tilde{p}(t) := s$ ;
(6)    WHILE  $\tilde{p}(t) \neq 0$  DO
(7)      BEGIN
(8)        FOR  $y \in \tilde{N}$  DO  $\tilde{p}(y) := 0$ ;
(9)         $\tilde{V} := \{s\}$ ;  $\tilde{p}(s) := t$ ;
(10)       WHILE  $\tilde{V} \neq \emptyset$  și  $\tilde{p}(t) = 0$  DO
(11)         BEGIN
(12)           se extrage un nod  $x$  din  $\tilde{V}$ ;
(13)           FOR  $(x, y)$  din  $\tilde{A}$  DO
(14)             IF  $\tilde{p}(y) = 0$ 
(15)               THEN BEGIN  $\tilde{p}(y) := x$ ;  $\tilde{V} := \tilde{V} \cup \{y\}$ ; END;
(16)             END;
(17)             IF  $\tilde{p}(t) \neq 0$ 
(18)               THEN MĂRIRE
(19)           END;
(20)       END.

```

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

**Exemplul 1.5.** Să ilustrăm algoritmul de etichetare pe rețeaua din figura 1.10 (a). Fluxul inițial este  $f_0 = (f(1, 2), f(1, 3), f(2, 3), f(2, 4), f(3, 4)) = (1, 0, 1, 0, 1)$ . Rețeaua reziduală  $\tilde{G}(f_0)$  este rețeaua arătată în figura 1.10 (b). La prima iterare vectorul predecesor  $\tilde{p}$  poate fi  $\tilde{p} = (4, 1, 1, 2)$  sau  $\tilde{p} = (4, 1, 1, 3)$ . Presupunem că  $\tilde{p} = (4, 1, 1, 2)$ , atunci  $\tilde{D} = (1, 2, 4)$  cu  $r(\tilde{D}) = u - 1$ . Rețeaua reziduală actualizată după mărirea de flux este arătată în figura 1.10 (c). În a doua iterare se obține  $\tilde{p} = (4, 3, 1, 3)$ ,  $\tilde{D} = (1, 3, 4)$ ,  $r(\tilde{D}) = u - 1$ .

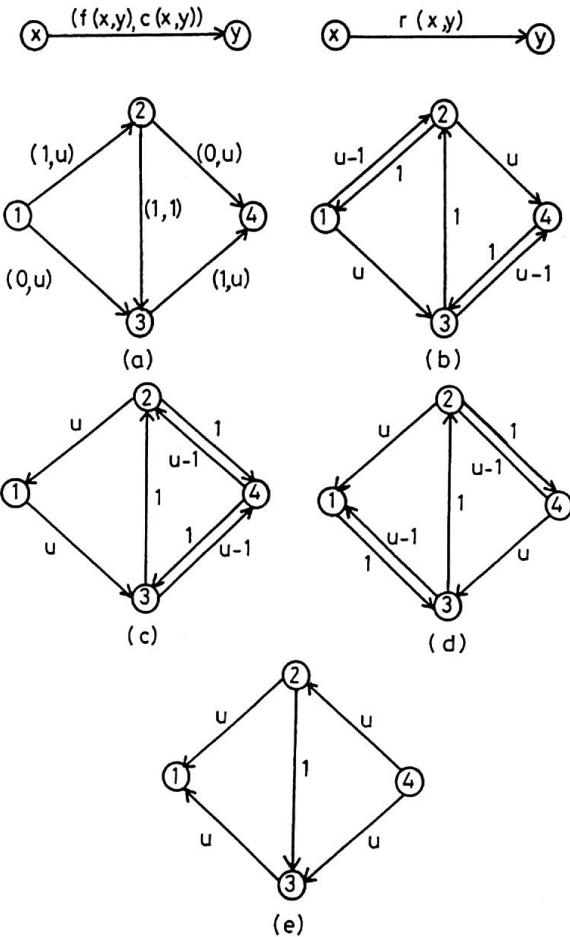


Fig.1.10 Ilustrarea algoritmului de etichetare

Rețeaua reziduală actualizată după mărirea de flux este arătată în figura 1.10 (d). În a treia iterare se obține  $\tilde{p} = (4, 3, 1, 2)$ ,  $\tilde{D} = (1, 3, 2, 4)$ ,  $r(\tilde{D}) = 1$ . După mărirea de flux se obține rețeaua reziduală din figura 1.10 (e). În această rețea inițial  $\tilde{V} = \{s\}$ , obținem  $\tilde{V} = \emptyset$ ,  $\tilde{p}(4) = 0$  și algoritmul se termină cu un flux maxim.

**Teorema 1.9. (Teorema fluxului maxim și a tăieturii minime).** *Într-o rețea  $G = (N, A, c)$  valoarea fluxului maxim de la nodul sursă  $s$  la nodul stoc  $t$  este egală cu capacitatea tăieturii  $s - t$  minime.*

**Demonstrație.** În rețeaua  $G = (N, A, c)$ , dacă  $f$  este un flux admisibil de valoare  $v$ ,  $[X, \bar{X}]$  o tăietură  $s - t$ , atunci conform Teoremei 1.6. avem  $v = f[X, \bar{X}] \leq c[X, \bar{X}]$ . Rezultă că, dacă  $f^*$  este un flux admisibil maxim de valoare  $v^*$  și  $[X^*, \bar{X}^*]$  este o tăietură  $s - t$  minimă atunci  $v^* = c[X^*, \bar{X}^*]$ . ■

**Teorema 1.10. (Teorema drumului de mărirea fluxului).** *În rețeaua  $G = (N, A, c)$  un flux  $f^*$  este un flux maxim dacă și numai dacă rețeaua reziduală  $\tilde{G}(f)$  nu conține drum de mărire a fluxului.*

**Demonstrație.** Dacă  $f^*$  este un flux maxim atunci  $\tilde{G}(f^*)$  nu conține DMF, altfel  $f^*$  nu ar fi flux maxim. Reciproc, dacă rețeaua reziduală  $\tilde{G}(f^*)$  nu conține DMF rezultă că nu poate fi etichetat nodul stoc  $t$  plecând de la nodul sursă  $s$ . Fie  $X^*$  mulțimea nodurilor etichetate și  $\bar{X}^* = N - X^*$  mulțimea nodurilor neetichetate. Evident că  $[X^*, \bar{X}^*]$  este o tăietură  $s - t$ . Deoarece algoritmul de etichetare nu a putut eticheta nodurile din  $\bar{X}^*$  de la nodurile din  $X^*$  rezultă că  $r^*(x, \bar{x}) = 0$  pentru oricare arc  $(x, \bar{x}) \in (X^*, \bar{X}^*)$ . Dar  $r^*(x, \bar{x}) = c(x, \bar{x}) - f^*(x, \bar{x}) + f^*(\bar{x}, x)$ ,  $c(x, \bar{x}) - f^*(x, \bar{x}) \geq 0$ ,  $f^*(\bar{x}, x) \geq 0$  și condițiile  $r^*(x, \bar{x}) = 0$  implică faptul că  $f^*(x, \bar{x}) = c(x, \bar{x})$  pentru oricare arc  $(x, \bar{x}) \in (X^*, \bar{X}^*)$  și  $f^*(\bar{x}, x) = 0$  pentru fiecare arc  $(\bar{x}, x) \in (\bar{X}^*, X^*)$ . Rezultă că  $v^* = f^*[X^*, \bar{X}^*] = f^*(X^*, \bar{X}^*) - f^*(\bar{X}^*, X^*) = f^*(X^*, \bar{X}^*) = c(X^*, \bar{X}^*) = c[X^*, \bar{X}^*]$ , adică fluxul  $f^*$  de valoare  $v^*$  obținut la terminarea algoritmului de etichetare este un flux maxim. ■

**Teorema 1.11. (Teorema de corectitudine a algoritmului de etichetare.)** *În rețeaua  $G = (N, A, c)$ , algoritmul de etichetare determină un flux maxim de la nodul sursă  $s$  la nodul stoc  $t$ .*

**Demonstrație.** Execuția algoritmului de etichetare se termină când nodul stoc  $t$  nu mai poate fi etichetat, prin operația de etichetare, pornind de la nodul sursă  $s$ . Deci în rețeaua reziduală  $\tilde{G}(f^*)$  nu există DMF și conform Teoremei 1.10  $f^*$  este un flux maxim. ■

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

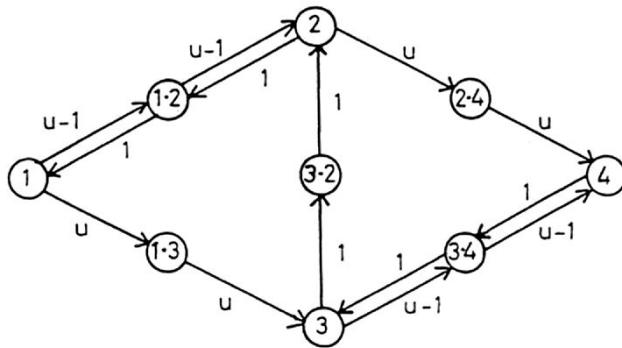
Pentru a studia complexitatea, în cazul cel mai defavorabil, a algoritmului de etichetare să analizăm dacă putem utiliza tehnica de construire a secvenței de fluxuri admisibile  $f_0, f_1, f_2, \dots, f_{2i-2}, f_{2i-1}, f_{2i}, \dots, f_{2u}$  de la algoritm generic. Evident, dacă se pleacă cu  $f_0 = 0$ , atunci  $\tilde{D} = (1, 2, 4)$  sau  $\tilde{D} = (1, 3, 4)$  și  $\tilde{D} = (1, 2, 3, 4)$  nu poate fi obținut în primele două iterări. Deci, cel puțin nu în toate cazurile, putem construi secvența de fluxuri admisibile de la algoritm generic. Totuși dacă se dă rețeaua originală  $G = (N, A, c)$ , atunci se poate construi rețeaua transformată  $G' = (N', A', c')$ , astfel încât comportarea algoritmului generic pe rețeaua reziduală  $\tilde{G}(f)$  este aceeași cu comportarea algoritmului de etichetare pe rețeaua reziduală  $\tilde{G}'(f)$ . Rețeaua  $G'$  se construiește în modul următor:

$$\begin{aligned} N' &= N \cup V, V = \{x.y \mid x.y \text{ se află pe arcul}(x,y) \in A\}, |N'| = n + m, \\ A' &= \{(x,x.y), (x.y,y) \mid (x,y) \in A\}, |A'| = 2m, \\ c' : A' &\rightarrow \mathbb{R}_+, c'(x,x.y) = c'(x.y,y) = c(x,y), (x,y) \in A. \end{aligned}$$

În aplicarea algoritmului de etichetare pe rețeaua  $\tilde{G}'(f)$  următoarele afirmații sunt adevărate:

- (a<sub>1</sub>) Dacă un nod  $x \in N$  este analizat, atunci numai noduri  $x.y \in V$  pot fi etichetate.
- (a<sub>2</sub>) Dacă un nod  $x.y \in V$  este analizat, atunci numai nodul  $y$  poate fi etichetat.

Din aceste afirmații rezultă că dacă algoritmul de etichetare se aplică pe  $\tilde{G}'(f)$  și  $\tilde{D}'$  este un DMF de la  $s$  la  $t$ , atunci este posibilă analizarea nodurilor din  $\tilde{D}'$  în ordinea în care ele apar în  $\tilde{D}'$ . Dacă  $\tilde{D}'$  este dat, atunci operația de etichetare se va termina prin determinarea DMF  $\tilde{D}'$ . Rețelei reziduale  $\tilde{G}(f)$  din figura 1.10 (b) îi corespunde rețeaua reziduală  $\tilde{G}'(f)$  arătată în figura 1.11.



**Fig.1.11** Rețeaua reziduală  $\tilde{G}'(f)$

Fie DMF  $\tilde{D} = (1, 3, 2, 4)$  în rețeaua  $\tilde{G}(f)$  din figura 1.10 (b). Am văzut că acest DMF nu poate fi obținut la prima iterație a algoritmului de etichetare. Drumul corespunzător în rețeaua  $\tilde{G}'(f)$  din figura 1.11 este  $\tilde{D}' = (1, 1.3, 3, 3.2, 2, 2.4, 4)$ . Se poate verifica că nodurile din  $\tilde{D}'$  se pot analiza în ordinea în care apar în  $\tilde{D}'$  și algoritmul de etichetare determină acest drum.

Rezultă următoarele concluzii:

- (c<sub>1</sub>) Dacă capacitatea și fluxul admisibil inițial sunt cu valori întregi, atunci aplicarea algoritmului de etichetare se termină cu un flux admisibil maxim după un număr finit de iterări și fluxul maxim este cu valori întregi.
- (c<sub>2</sub>) Dacă capacitatele sunt iraționale, atunci aplicarea algoritmului de etichetare poate produce o secvență infinită de fluxuri admisibile ale căror valori converg la o valoare strict mai mică decât valoarea fluxului maxim.
- (c<sub>3</sub>) Complexitatea algoritmului de etichetare este aceeași cu a algoritmului generic, adică exponentională în raport cu dimensiunea problemei.

**Teorema 1.12. (Teorema de complexitate a algoritmului de etichetare).**  
Dacă capacitatea și fluxul admisibil inițial sunt cu valori întregi atunci algoritmul de etichetare are complexitatea  $O(mn\bar{c})$ ,  $\bar{c} = \max\{c(x,y) \mid (x,y) \in A\}$ .

**Demonstrație.** Pentru determinarea unui DMF sunt necesare cel mult  $m$  analizări de arce. Dacă capacitatea și fluxul admisibil inițial 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  $\bar{c}$  este o margine superioară a capacitaților arcelor atunci  $v^* \leq (n - 1)\bar{c}$  deoarece  $(n - 1)\bar{c}$  este o margine superioară a capacitații tăieturii  $s - t$   $[s, \bar{X}]$ . Deci algoritmul de etichetare are complexitatea  $O(mn\bar{c})$ . ■

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

## 1.5 Aplicații ale teoremei fluxului maxim și tăieturii minime

### 1.5.1 Conexitatea rețelei

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

**Teorema 1.13. (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 rețea deconexează toate drumurile de la nodul  $s$  la nodul  $t$   $((s, t) \notin A)$ .*

**Demonstrație.** Definim capacitatea  $c : A \rightarrow \mathbb{R}_+$  prin  $c(x, y) = 1$  pentru fiecare arc  $(x, y) \in A$ . Fie  $f^*$  fluxul maxim de valoare  $v^*$ . Teorema descompunerii fluxului (Teorema 1.1) implică faptul că putem descompune fluxul  $f^*$  de-a lungul drumurilor de la  $s$  la  $t$  și 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 fiecarui arc este 1, aceste drumuri care transportă flux sunt disjuncte arc și fiecare transportă o unitate de flux. Deci numărul maxim de drumuri disjuncte arc de la  $s$  la  $t$  este  $v^*$ . Fie  $[X, \bar{X}]$  tăietura  $s - t$  minimă. Conform definiției capacitații oricărei tăieturi avem  $c[X, \bar{X}] = c(X, \bar{X})$ . Deoarece capacitatea fiecarui arc este 1 rezultă  $c(X, \bar{X}) = |(X, \bar{X})|$ . Fiecare drum de la nodul sursă  $s$  la nodul stoc  $t$  conține un arc din  $(X, \bar{X})$  și eliminarea acestor arce deconexează toate drumurile de la  $s$  la  $t$ . Tăietura  $s - t$   $[X, \bar{X}]$  fiind minimă și  $c[X, \bar{X}] = c(X, \bar{X}) = |(X, \bar{X})|$  rezultă că numărul arcelor care deconexează toate drumurile de la  $s$  la  $t$  este minim. Conform Teoremei fluxului maxim și tăieturii minime (Teorema 1.9) avem  $v^* = c[X, \bar{X}] = c(X, \bar{X}) = |(X, \bar{X})|$  și rezultă afirmația din teoremă. ■

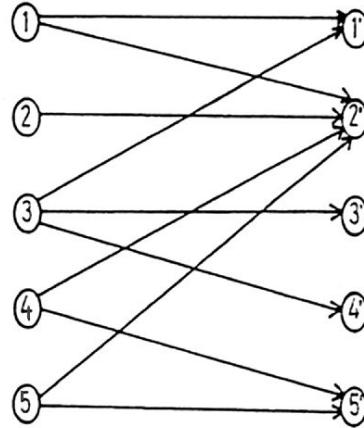
**Teorema 1.14. (Teorema deconexării drumurilor  $s - t$  prin noduri)** *Numărul maxim al drumurilor disjuncte nod de la nodul sursă  $s$  la nodul stoc  $t$  este egal cu numărul minim al nodurilor care eliminate din rețea deconexează toate drumurile de la nodul  $s$  la nodul  $t$   $((s, t) \notin A)$ .*

**Demonstrație.** Definim rețeaua  $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{c})$  cu  $N_0 = \{s, t\}$ ,  $\tilde{N} = \{x', x'' \mid x \in N - N_0\} \cup \tilde{N}_0$ ,  $\tilde{N}_0 = \{s'', t'\}$ ,  $s'' = s$ ,  $t' = t$ ,  $\tilde{A} = \tilde{A}_1 \cup \tilde{A}_2$ ,  $\tilde{A}_1 = \{(x'', y') \mid (x, y) \in A\}$ ,  $\tilde{A}_2 = \{(x', x'') \mid x \in N - N_0\}$ ,  $\tilde{c} : \tilde{A} \rightarrow \mathbb{R}_+$ ,  $\tilde{c}(x'', y') = \infty$ ,  $(x'', y') \in \tilde{A}_1$ ,  $\tilde{c}(x', x'') = 1$ ,  $(x', x'') \in \tilde{A}_2$ . Evident că există o corespondență biunivocă între mulțimea drumurilor

disjuncte nod din rețeaua originară  $G$  și mulțimea drumurilor disjuncte arc din rețeaua transformată  $\tilde{G}$ . Deoarece  $\tilde{c}(x'', y') = \infty$ ,  $(x'', y') \in \tilde{A}_1$ , rezultă că în  $\tilde{G}$  tăietura  $s'' - t'$  minimă are ca arce directe numai arce  $(x', x'')$  din  $\tilde{A}_2$ . Aplicând Teorema 1.13 și utilizând observațiile de mai sus rezultă afirmația din teoremă. ■

### 1.5.2 Cuplaje și acoperiri

Fie  $G = (N, A, c)$ ,  $N = N_1 \cup N_2$  o rețea bipartită. O submulțime  $A' \subset A$  se numește *cuplaj* dacă oricare două arce din  $A'$  nu sunt adiacente. O submulțime  $N' \subset N$  se numește *acoperire nod* dacă fiecare arc din  $A$  este incident la unul din nodurile din  $N'$ . **Exemplul 1.6.** Pentru ilustrarea acestor definiții se consideră rețeaua bipartită arătată în figura 1.12. Mulțimea  $A' = \{(1, 1'), (3, 3'), (4, 5'), (5, 2')\}$  este un cuplaj, dar  $A'' = \{(1, 2'), (3, 1'), (3, 4')\}$  nu este deoarece arcele  $(3, 1')$  și  $(3, 4')$  sunt adiacente fiind incidente la același nod 3. Mulțimea  $N' = \{1, 2', 3, 5'\}$  este o acoperire nod, dar mulțimea  $N'' = \{2', 3', 4, 5\}$  nu este o acoperire nod, deoarece arcele  $(1, 1')$ ,  $(3, 1')$  și  $(3, 4')$  nu sunt incidente la nici un nod din mulțimea  $N''$ .



**Fig.1.12** Rețeaua bipartită

**Teorema 1.15. (Teorema cuplajului maxim și a acoperirii minime.)** Într-o rețea 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  $G$ .

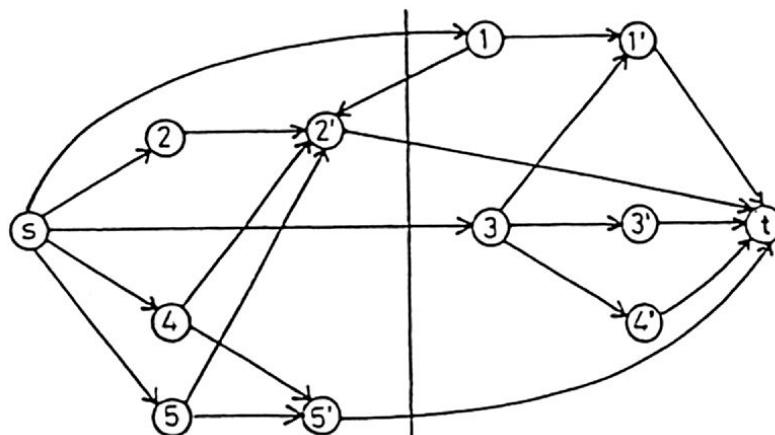
**Demonstrație.** Definim rețeaua  $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{c})$ , unde  $\tilde{N} = \tilde{N}_1 \cup \tilde{N}_2 \cup \tilde{N}_3$ ,  $\tilde{N}_1 = N_1$ ,  $\tilde{N}_2 = N_2$ ,  $\tilde{N}_3 = \{s, t\}$ ,  $\tilde{A} = \tilde{A}_1 \cup \tilde{A}_2$ ,  $\tilde{A}_1 = A$ ,  $\tilde{A}_2 = \{(s, x), (y, t) \mid x \in \tilde{N}_1, y \in \tilde{N}_2\}$ ,  $\tilde{c} : \tilde{A} \rightarrow \mathbb{R}_+$ ,  $\tilde{c}(x, y) = \infty$ ,  $(x, y) \in \tilde{A}_1$ ,  $\tilde{c}(x, y) = 1$ ,  $(x, y) \in \tilde{A}_2$ . Fie  $f^*$  fluxul maxim de valoare  $\tilde{v}^*$  în rețeaua modificată  $\tilde{G}$ . Putem descompune fluxul  $\tilde{f}^*$  în fluxuri de-a lungul a  $\tilde{v}^*$  drumuri de forma  $\tilde{D} = (s, x, y, t)$ , fiecare drum transportând o unitate de flux. Astfel  $\tilde{v}^*$  arce  $(x, y) \in \tilde{A}_1 = A$  au  $\tilde{f}^*(x, y) = 1$ . Deoarece aceste drumuri sunt disjuncte nod, mulțimea arcelor  $(x, y) \in \tilde{A}_1 = A$  cu  $\tilde{f}^*(x, y) = 1$  constituie un cuplaj în  $G$ . Din faptul că fluxul  $\tilde{f}^*$  este maxim rezultă că, cuplajul definit de acest flux este de cardinalitate

maximă egală cu  $\tilde{v}^*$ . Invers, un cuplaj de cardinalitate maximă  $\tilde{v}^*$  definește un flux maxim  $\tilde{f}^*$  în rețeaua  $\tilde{G}$ .

Fie  $[\tilde{X}^*, \tilde{\bar{X}}^*]$  o tăietură  $s - t$  minimă din rețeaua  $\tilde{G}$ . Deoarece  $\tilde{c}(x, y) = \infty$  pentru  $(x, y) \in \tilde{A}_1 = A$ , rezultă că mulțimea arcelor directe  $(\tilde{X}^*, \tilde{\bar{X}}^*)$  conține numai arce  $(x, y) \in \tilde{A}_2$  cu  $\tilde{c}(x, y) = 1$ . Deci  $c[\tilde{X}^*, \tilde{\bar{X}}^*] = c(\tilde{X}^*, \tilde{\bar{X}}^*) = |(\tilde{X}^*, \tilde{\bar{X}}^*)|$ . Din mulțimea arcelor directe  $(\tilde{X}^*, \tilde{\bar{X}}^*)$  construim mulțimea de noduri  $Y^*$  în modul următor. Fiecare arc  $(x, y) \in \tilde{A}_1 = A$  definește un drum  $\tilde{D} = (s, x, y, t)$ . Deoarece  $[\tilde{X}^*, \tilde{\bar{X}}^*]$  este o tăietură  $s - t$ , fie  $(s, x) \in (\tilde{X}^*, \tilde{\bar{X}}^*)$ , fie  $(y, t) \in (\tilde{X}^*, \tilde{\bar{X}}^*)$ , 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  $|Y^*| = |(\tilde{X}^*, \tilde{\bar{X}}^*)|$  și deoarece  $[\tilde{X}^*, \tilde{\bar{X}}^*]$  este o tăietură  $s - t$  minimă acoperirea nod  $Y^*$  este de cardinalitate minimă. Reciproc, dacă  $Y^*$  este o acoperire nod din rețeaua  $G$  se poate defini o tăietură  $[\tilde{X}^*, \tilde{\bar{X}}^*]$  în rețeaua  $\tilde{G}$  cu capacitatea  $|Y^*|$  în modul următor. Fie  $y \in Y^*$ . Dacă  $y \in N_1$ , atunci adăugăm arcul  $(s, y)$  la  $(\tilde{X}^*, \tilde{\bar{X}}^*)$ , iar dacă  $y \in N_2$  atunci adăugăm arcul  $(y, t)$  la  $(\tilde{X}^*, \tilde{\bar{X}}^*)$  și evident că  $c[\tilde{X}^*, \tilde{\bar{X}}^*] = c(\tilde{X}^*, \tilde{\bar{X}}^*) = |(\tilde{X}^*, \tilde{\bar{X}}^*)| = |Y^*|$ .

Conform teoremei fluxului maxim și tăieturii minime avem  $\tilde{v}^* = c[\tilde{X}^*, \tilde{\bar{X}}^*] = |(\tilde{X}^*, \tilde{\bar{X}}^*)| = |Y^*|$  și utilizând concluziile de mai sus rezultă afirmația teoremei. ■

**Exemplul 1.7.** Ilustrăm Teorema 1.15 în exemplul prezentat în figura 1.13.



**Fig.1.13** Tăietura minimă pentru problema fluxului maxim corespunzătoare figurii 1.12

Problema de cuplaj din figura 1.12 am transformat-o într-o problemă de flux maxim în rețeaua din figura 1.13. Mulțimea arcelor directe ale tăieturii minime este  $(\tilde{X}^*, \tilde{\bar{X}}^*) = \{(s, 1), (s, 3), (2', t), (5', t)\}$ . Corespunzător, mulțimea  $Y^* = \{1, 3, 2', 5'\}$  este o acoperire nod de cardinalitate minimă și un cuplaj de cardinalitate maximă este  $\{(1, 1'), (2, 2'), (3, 3'), (5, 5')\}$ .

## 1.6 Fluxuri cu margini inferioare pozitive

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

$$\text{Maximizează } v \quad (1.6.a)$$

$$f(x, N) - f(N, x) = \begin{cases} v, & x = s, \\ 0, & x \neq s, t, \\ -v, & x = t, \end{cases} \quad (1.6.b)$$

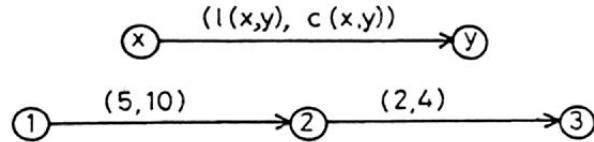
$$l(x, y) \leq f(x, y) \leq c(x, y), \quad (x, y) \in A. \quad (1.6.c)$$

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

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

Evident, dacă  $l(x, y) = 0$  pentru  $(x, y) \in A$  atunci  $c[X, \bar{X}] = c(X, \bar{X})$  și se obține relația prin care s-a definit capacitatea unei căieturi în cazul când marginile inferioare sunt zero.

Problema fluxului maxim cu margini inferioare zero are întotdeauna o soluție admisibilă și anume fluxul zero. Problema fluxului maxim cu margini inferioare pozitive poate fi inadmisibilă. De exemplu, considerăm problema fluxului maxim în rețeaua reprezentată în figura 1.14. Această problemă nu are soluție admisibilă deoarece pe arcul  $(1,2)$  trebuie să transportăm cel puțin 5 unități de flux și pe arcul  $(2,3)$  putem să transportăm cel mult 4 unități de flux și pentru nodul 2 nu se verifică constrângerea din (1.6 b).



**Fig.1.14** Problema fluxului maxim cu soluție inadmisibilă

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 rețeaua  $G = (N, A, l, c)$ . 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 creșterea maximă de flux pe arcul  $(x, y)$  utilizând capacitatea rămasă și termenul  $f(y, x) - l(y, x)$  reprezintă creșterea maximă de flux pe arcul  $(x, y)$  prin eliminarea fluxului excedent pe arcul  $(y, x)$ . Deoarece

algoritmii care au fost și vor fi prezentăți lucrează numai cu capacitate reziduale, putem utiliza oricare dintre acești algoritmi pentru a determina un flux maxim cu margini inferioare pozitive. Oricare din acești algoritmi se termină cu capacitate reziduale optime. Din aceste capacitate 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 obține  $r'(x, y) = c'(x, y) - f'(x, y) + f'(y, x)$ . Aceste capacitate reziduale sunt similare cu capacitatele reziduale pentru cazul când marginile inferioare sunt zero. Calculăm fluxul  $f'$  în funcție de  $r'$  și  $c'$  ca în cazul când marginile inferioare sunt zero și obținem  $f'(x, y) = \max\{0, c'(x, y) - r'(x, y)\}$  și  $f'(y, x) = \max\{0, c'(y, x) - r'(y, x)\}$ . Revenind la variabilele originare obținem  $f(x, y) = l(x, y) + \max\{0, c(x, y) - r(x, y) - l(x, y)\}$  și  $f(y, x) = l(y, x) + \max\{0, c(y, x) - r(y, x) - l(y, x)\}$ .

**Teorema 1.16. (Teorema generalizată a fluxului maxim și tăieturii minime).**

*Intr-o rețea  $G = (N, A, l, c)$ , valoarea fluxului maxim este egală cu capacitatea tăieturii  $s - t$  minime.*

**Demonstrație.** Fie  $f$  un flux admisibil în rețeaua  $G = (N, A, l, c)$  și  $[X, \bar{X}] = (X, \bar{X}) \cup (\bar{X}, X)$  o tăietură  $s - t$ . Fluxul  $f$  verifică constrângerile de conservare (1.6 b). Sumând aceste ecuații pentru  $x \in X$  se obține  $v = f(X, N) - f(N, X)$ . Înlocuind  $N = X \cup \bar{X}$  și dezvoltând rezultă  $v = f(X, \bar{X}) - f(\bar{X}, X) = f[X, \bar{X}]$ . De asemenea  $f$  verifică constrângerile de mărginire a fluxului (1.6 c) și se obține  $f(X, \bar{X}) \leq c(X, \bar{X})$ ,  $f(\bar{X}, X) \leq f(\bar{X}, X)$ . Rezultă  $v = f[X, \bar{X}] = f(X, \bar{X}) - f(\bar{X}, X) \leq c(X, \bar{X}) - f(\bar{X}, X) = c[X, \bar{X}]$ . Deci, dacă  $f^*$  este un flux maxim de valoare  $v^*$  și  $[X^*, \bar{X}^*]$  este o tăietură  $s - t$  minimă atunci  $v^* = f^*[X^*, \bar{X}^*] = c[X^*, \bar{X}^*]$ . ■

În continuare prezentăm problema determinării unui flux admisibil dacă el există. Transformăm problema fluxului admisibil  $f$  din rețeaua  $G = (N, A, l, c)$  în problema circulației admisibile  $\tilde{f}$  în rețeaua transformată  $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{l}, \tilde{c})$  unde  $\tilde{N} = N$ ,  $\tilde{A} = A \cup \{(t, s)\}$ ,  $\tilde{l}(x, y) = l(x, y)$ ,  $\tilde{c}(x, y) = c(x, y)$ ,  $(x, y) \in A$ ,  $\tilde{l}(t, s) = 0$ ,  $\tilde{c}(t, s) = \infty$ . Problema circulației admisibile constă în a determina un flux  $\tilde{f}$ , dacă există, satisfăcând constrângerile:

$$\tilde{f}(x, \tilde{N}) - \tilde{f}(\tilde{N}, x) = 0, \quad x \in \tilde{N} \quad (1.7.a)$$

$$\tilde{l}(x, y) \leq \tilde{f}(x, y) \leq \tilde{c}(x, y), \quad (x, y) \in \tilde{A} \quad (1.7.b)$$

Înlocuim  $\tilde{f}(x, y)$  prin  $\tilde{f}'(x, y) + \tilde{l}(x, y)$  și se obține

$$\tilde{f}'(x, \tilde{N}) - \tilde{f}'(\tilde{N}, x) = \tilde{v}'(x), \quad x \in \tilde{N} \quad (1.8.a)$$

$$0 \leq \tilde{f}'(x, y) \leq \tilde{c}'(x, y), \quad (x, y) \in \tilde{A} \quad (1.8.b)$$

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

$$\tilde{v}'(x) = \tilde{l}(\tilde{N}, x) - \tilde{l}(x, \tilde{N}), \quad x \in N \quad (1.8.c)$$

Se observă că  $\tilde{v}'(\tilde{N}) = 0$ . Astfel problema circulației admisibile este echivalentă cu a determina dacă problema transformată are o soluție  $\tilde{f}'$  care verifică (1.8). În problema ofertei și cererii prezentată în paragraful 1.1 am arătat că prin rezolvarea unei probleme de flux maxim, fie determinăm o soluție care satisface (1.8), fie arătăm că soluția nu satisface (1.8). Dacă  $\tilde{f}'$  este o soluție admisibilă pentru constrângările (1.8) atunci  $\tilde{f}$  cu  $\tilde{f}(x, y) = \tilde{f}'(x, y) + \tilde{l}(x, y)$  este o soluție admisibilă pentru constrângările (1.7).

**Exemplul 1.8.** Fie rețeaua  $G = (N, A, l, c)$  reprezentată în figura 1.15 unde pe fiecare arc  $(x, y)$  sunt trecute  $l(x, y)$  și  $c(x, y)$  în această ordine. Dacă există un flux admisibil, atunci să se determine un flux maxim.

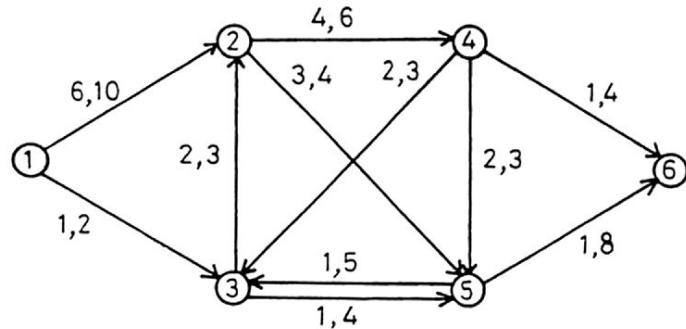


Fig.1.15

Pentru a rezolva problema ofertă-cerere (1.8) se construiește rețeaua  $\tilde{G}'_0 = (\tilde{N}'_0, \tilde{A}'_0, \tilde{c}'_0)$  unde:  $\tilde{N}'_0 = \tilde{N}'_1 \cup \tilde{N}'_2 \cup \tilde{N}'_3$ ,  $\tilde{N}'_1 = \{\tilde{s}\}$ ,  $\tilde{N}'_2 = \tilde{N}$ ,  $\tilde{N}'_3 = \{\tilde{t}\}$ ;  $\tilde{A}'_0 = \tilde{A}'_1 \cup \tilde{A}'_2 \cup \tilde{A}'_3$ ,  $\tilde{A}'_1 = \{(\tilde{s}, x) \mid \tilde{v}'(x) > 0\}$ ,  $\tilde{A}'_2 = \tilde{A}$ ,  $\tilde{A}'_3 = \{(x, \tilde{t}) \mid \tilde{v}'(x) < 0\}$ ;  $\tilde{c}'_0(\tilde{s}, x) = \tilde{v}'(x)$ ,  $(\tilde{s}, x) \in \tilde{A}'_1$ ,  $\tilde{c}'_0(x, y) = \tilde{c}'(x, y)$ ,  $(x, y) \in \tilde{A}'_2$ ,  $\tilde{c}'_0(x, \tilde{t}) = -\tilde{v}'(x)$ ,  $(x, \tilde{t}) \in \tilde{A}'_3$ . Rețeaua  $\tilde{G}'_0$  este reprezentată în figura 1.16, unde capacitatea  $\tilde{c}'_0$  este reprezentată de al doilea număr asociat arcului  $(x, y)$ .

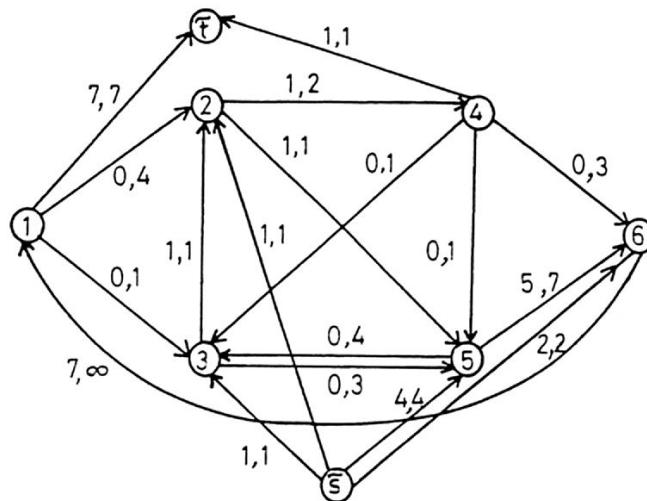


Fig.1.16

Primul număr asociat arcului  $(x, y)$  din rețeaua  $\tilde{G}'_0$  reprezintă fluxul maxim  $\tilde{f}'_0$ . Deoarece  $\tilde{f}'_0$  saturează toate arcele din  $\tilde{A}'_1$  și  $\tilde{A}'_3$  rezultă că  $\tilde{f}'$ , restricția lui  $\tilde{f}'_0$  la  $\tilde{A}'_2 = \tilde{A}$ , este o soluție pentru problema ofertă - cerere (1.8). Marginea inferioară  $l(x, y)$ , fluxul admisibil  $f(x, y) = l(x, y) + \tilde{f}'(x, y)$  și capacitatea  $c(x, y)$  sunt asociate fiecărui arc  $(x, y)$ , în această ordine, din rețeaua  $G = (N, A, l, c)$  reprezentată în figura 1.17.

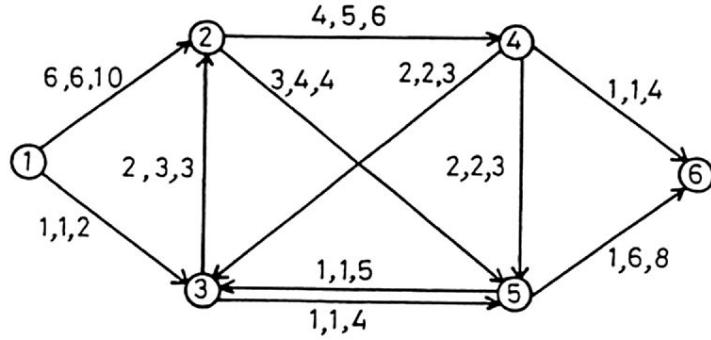


Fig.1.17

Rețeaua reziduală  $\tilde{G}(f') = (\tilde{N}, \tilde{A}, r')$  cu  $f'(x, y) = f(x, y) - l(x, y)$ ,  $c'(x, y) = c(x, y) - l(x, y)$ ,  $r'(x, y) = c'(x, y) - f'(x, y) + f'(y, x)$  este reprezentată în figura 1.18. Evident că  $\tilde{G}(f') \equiv \tilde{G}(f)$ .

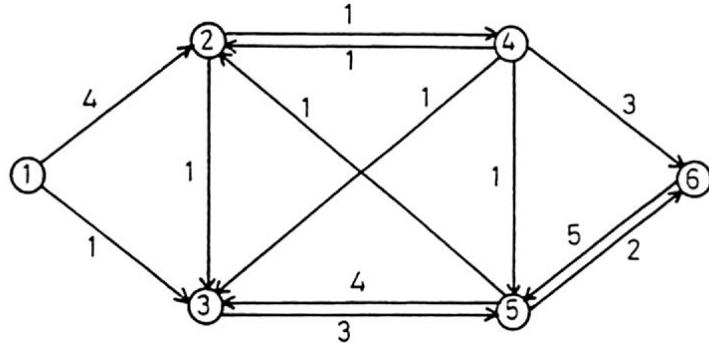


Fig.1.18

Aplicând algoritmul de etichetare Ford-Fulkerson se obține în final rețeaua reziduală din figura 1.19.

Rețeaua  $G = (N, A, l, c)$  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 1.20.

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

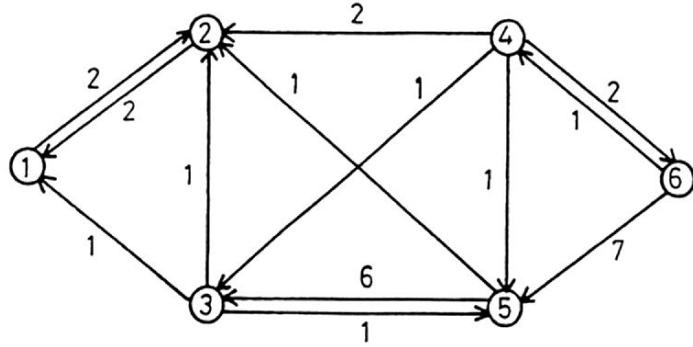


Fig.1.19

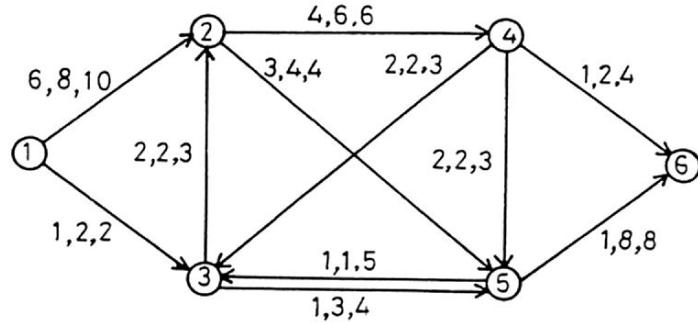


Fig.1.20

**Teorema 1.17. (Teorema admisibilității circulației).** Într-o rețea  $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{l}, \tilde{c})$  există o circulație admisibilă dacă și numai dacă pentru orice mulțime  $\tilde{X} \subset \tilde{N}$  este verificată condiția

$$\tilde{l}(\overline{\tilde{X}}, \tilde{X}) \leq \tilde{c}(\tilde{X}, \overline{\tilde{X}}) \quad (1.9)$$

**Demonstratie.** Necesitatea. Presupunem că circulația  $\tilde{f}$  este admisibilă în rețeaua  $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{l}, \tilde{c})$ . Aceasta înseamnă că circulația  $\tilde{f}$  verifică condițiile (1.7 a) și (1.7 b.)

Fie o mulțime oarecare  $\tilde{X} \subset \tilde{N}$  în rețeaua  $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{l}, \tilde{c})$ . Sumând condițiile (1.7 a) pentru  $x \in \tilde{X}$  se obține

$$\tilde{f}(\tilde{X}, \overline{\tilde{X}}) - \tilde{f}(\overline{\tilde{X}}, \tilde{X}) = 0. \quad (1.10)$$

Din condițiile (1.7 b) rezultă că  $\tilde{f}(\tilde{X}, \overline{\tilde{X}}) \leq \tilde{c}(\tilde{X}, \overline{\tilde{X}})$ ,  $\tilde{l}(\overline{\tilde{X}}, \tilde{X}) \leq \tilde{f}(\overline{\tilde{X}}, \tilde{X})$ . Utilizând aceste inegalități în (1.10) se obține

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

**Suficiența.** Presupunem că este verificată condiția (1.9) pentru orice mulțime  $\tilde{X} \subset \tilde{N}$ . Pentru suficiență se dă o demonstrație algoritmică. Algoritmul pornește cu o circulație

$\tilde{f}$  care verifică condițiile (1.7 a) și condițiile pentru capacitate din (1.7 b), dar nu verifică condițiile pentru marginile inferioare din (1.7 b). De exemplu  $\tilde{f} = 0$ . Rețeaua reziduală  $\tilde{G}'(\tilde{f})$  se definește conform celor precizate în acest paragraf cu deosebirea că pentru un arc  $(x, y)$  cu  $\tilde{f}(x, y) < \tilde{l}(x, y)$  definim  $\tilde{r}'(x, y) = \tilde{c}(x, y) - \tilde{f}(x, y)$ . Algoritmul este următorul:

```

(1)   PROGRAM CIRCULATIE;
(2)   BEGIN
(3)        $\tilde{f} := \tilde{f}_0$ ;
(4)       WHILE există arc  $(x_0, y_0) \in \tilde{A}'$  cu  $\tilde{f}(x_0, y_0) < \tilde{l}(x_0, y_0)$  DO
(5)           BEGIN
(6)               IF există DMF de la  $y_0$  la  $x_0$  în  $\tilde{G}'(\tilde{f})$ 
(7)               THEN BEGIN
(8)                   se determină un DMF  $\tilde{D}'$  de la  $y_0$  la  $x_0$  în  $\tilde{G}'(\tilde{f})$ ;
(9)                   se face mărirea de flux pe circuitul  $\tilde{D}' \cup \{(x_0, y_0)\}$ ;
(10)                  se actualizează  $\tilde{G}'(\tilde{f})$ ;
(11)              END;
(12)          ELSE EXIT;
(13)      END;
(14)  END.

```

Dacă algoritmul se termină fără execuția liniei (12), atunci se obține o circulație  $\tilde{f}$  admisibilă. În caz contrar arătăm că circulația  $\tilde{f}$  este inadmisibilă. Presupunem că pentru determinarea unui DMF în  $\tilde{G}'(\tilde{f})$  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  $\tilde{G}'(\tilde{f})$ . Fie  $\tilde{X}'$  mulțimea nodurilor etichetate. Este evident că  $y_0 \in \tilde{X}'$ ,  $x_0 \in \overline{\tilde{X}'}$  și  $\tilde{r}'(\tilde{X}', \overline{\tilde{X}'}) = 0$ . Pentru  $\tilde{N}' = \tilde{N}$  rezultă că  $\tilde{X}' = \tilde{X}$  și din  $\tilde{r}'(\tilde{X}, \overline{\tilde{X}}) = 0$  rezultă că  $\tilde{f}(\tilde{X}, \overline{\tilde{X}}) = \tilde{c}(\tilde{X}, \overline{\tilde{X}})$  și  $\tilde{f}(\overline{\tilde{X}}, \tilde{X}) \leq \tilde{l}(\overline{\tilde{X}}, \tilde{X})$ . Pentru  $(x_0, y_0) \in (\overline{\tilde{X}}, \tilde{X})$  avem  $\tilde{f}(x_0, y_0) < \tilde{l}(x_0, y_0)$ . Substituind aceste inegalități în (1.10) rezultă că

$$\tilde{l}(\overline{\tilde{X}}, \tilde{X}) > \tilde{c}(\tilde{X}, \overline{\tilde{X}}) \quad (1.11)$$

Dar condiția (1.11) contrazice condiția (1.9). Deci dacă este verificată condiția (1.9) pentru orice  $\tilde{X} \subset \tilde{N}$  atunci algoritmul se termină fără a executa linia (12) și determină o circulație admisibilă  $\tilde{f}$ . ■

**Teorema 1.18. (Teorema admisibilității fluxului).** *Într-o rețea  $G = (N, A, l, c)$  există un flux admisibil dacă și numai dacă pentru orice mulțime  $X \subset N$  cu  $s, t \in X$  sau  $s, t \in \overline{X}$  este verificată condiția*

$$l(\overline{X}, X) \leq c(X, \overline{X}) \quad (1.12)$$

**Demonstrație.** Se construiește rețeaua  $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{l}, \tilde{c})$ , unde  $\tilde{N} = N$ ,  $\tilde{A} = A \cup \{(s, t), (t, s)\}$ ,  $\tilde{l}(x, y) = l(x, y)$ ,  $\tilde{c}(x, y) = c(x, y)$ ,  $(x, y) \in A$ ,  $\tilde{l}(s, t) = \tilde{l}(t, s) = 0$ ,  $\tilde{c}(s, t) = \tilde{c}(t, s) = \infty$ . Rezultă că există un flux admisibil în rețeaua  $G$  dacă și numai dacă există o circulație admisibilă  $\tilde{f}$  în rețeaua  $\tilde{G}$ . Conform Teoremei 1.17 există o circulație admisibilă  $\tilde{f}$  în rețeaua  $\tilde{G}$  dacă și numai dacă pentru orice mulțime  $\tilde{X} \subset \tilde{N}$  este verificată condiția (1.9). Dacă  $s \in \tilde{X}$  și  $t \in \overline{\tilde{X}}$  sau dacă  $s \in \overline{\tilde{X}}$ ,  $t \in \tilde{X}$ , atunci

$c(\tilde{X}, \overline{\tilde{X}}) = \infty$  și condiția (1.9) este verificată. Deci (1.9) este verificată pentru orice multime  $\tilde{X} \subset \tilde{N}$  dacă și numai dacă (1.12) este verificată pentru orice multime  $X \subset N$  cu  $s, t \in X$  sau  $s, t \in \overline{X} (\tilde{N} = N, \tilde{X} = X)$ . ■

**Teorema 1.19.** (Teorema de admisibilitate a problemei ofertei și cererii). Într-o rețea  $G = (N, A, c)$ , problema ofertei și cererii este admisibilă dacă și numai dacă pentru orice multime  $X \subset N$  este verificată condiția

$$v(X) \leq c[X, \overline{X}] \quad (1.13)$$

**Demonstrație.** Se construiește rețeaua  $\tilde{G} = (\tilde{N}, \tilde{A}, \tilde{l}, \tilde{c})$ , unde  $\tilde{N} = \tilde{N}_1 \cup \tilde{N}_2 \cup \tilde{N}_3$ ,  $\tilde{N}_1 = \{\tilde{s}\}$ ,  $\tilde{N}_2 = N$ ,  $\tilde{N}_3 = \{\tilde{t}\}$ ;  $\tilde{A} = \tilde{A}_1 \cup \tilde{A}_2 \cup \tilde{A}_3$ ,  $\tilde{A}_1 = \{(\tilde{s}, x) \mid \tilde{v}(x) > 0\}$ ,  $\tilde{A}_2 = A$ ,  $\tilde{A}_3 = \{(x, \tilde{t}) \mid v(x) < 0\}$ ;  $\tilde{l}(\tilde{s}, x) = \tilde{c}(\tilde{s}, x) = v(x)$ ,  $(\tilde{s}, x) \in \tilde{A}_1$ ,  $\tilde{l}(x, \tilde{t}) = \tilde{c}(x, \tilde{t}) = -v(x)$ ,  $(x, \tilde{t}) \in \tilde{A}_3$ . Este evident că, există un flux admisibil în  $G$  dacă și numai dacă există un flux admisibil în  $\tilde{G}$ . Aplicând Teorema 1.18 la rețeaua  $\tilde{G}$  rezultă Teorema 1.19 pentru rețeaua  $G$ . ■

## 1.7 Aplicații

### 1.7.1 Probleme de transport

În centrele  $s_1, \dots, s_p$  se găsesc cantitățile  $u_1, \dots, u_p$  de produse care urmează a fi transportate în centrele  $t_1, \dots, t_q$ , fiecare centru având nevoie de cantitățile  $v_1, \dots, v_q$ . 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 cantitățile  $f(s_i, t_j)$  astfel încât să fie verificate condițiile:

$$\sum_{j=1}^q f(s_i, t_j) = u_i, \quad i = 1, \dots, p;$$

$$\sum_{i=1}^p f(s_i, t_j) = v_j, \quad j = 1, \dots, q;$$

$$0 \leq f(s_i, t_j) \leq c(s_i, t_j), \quad i = 1, \dots, p; \quad j = 1, \dots, q;$$

$$\sum_{i=1}^p u_i = \sum_{j=1}^q v_j.$$

Această problemă de transport constituie un caz particular (multimea nodurilor de tranzit  $R = \emptyset$ ) al problemei ofertă - cerere prezentată în paragraful 1.1.

### 1.7.2 Un caz particular al problemei de afectare

Problema particulară de afectare constituie un caz particular al problemei de transport. Problema constă în a determina cantitățile  $f(s_i, t_j)$  astfel încât să fie verificate condițiile:

$$\sum_{j=1}^q f(s_i, t_j) = 1, \quad i = 1, \dots, p;$$

$$\sum_{i=1}^p f(s_i, t_j) = 1, \quad j = 1, \dots, p;$$

$$0 \leq f(s_i, t_j) \leq \infty, \quad i = 1, \dots, p; \quad j = 1, \dots, p.$$

Este evident că o problemă de afectare este o problemă a cuplajului maxim prezentată în paragraful 1.5.

Un exemplu care ilustrează acest tip de problemă este acela al repartiției solicitanților pe posturi. Un număr de solicitanți, cu aptitudini multiple, urmează să fie repartizați la posturi. Fiecare solicitant este calificat pentru mai multe posturi. Se cere ca solicitanții să ocupe numai posturi pentru care sunt calificați și nici un post să nu rămână neocupat.

### 1.7.3 Planificarea lucrărilor pe mașini paralele

Considerăm că avem  $J$  lucrări și  $M$  mașini paralele. Fiecare lucrare  $j$ ,  $j = 1, \dots, J$  are un timp necesar prelucrării notat cu  $p_j$  (care reprezintă numărul de zile necesare unei mașini pentru a efectua lucrarea), o dată de apariție  $a_j$  (reprezentând începutul zilei în care lucrarea  $j$  a devenit disponibilă pentru prelucrare) și un termen limită  $t_j$  (reprezentând începutul zilei în care lucrarea  $j$  trebuie să fie gata). O mașină poate efectua o singură lucrare la un moment dat și o lucrare poate fi procesată de către o singură mașină la un moment dat, dar este permisă intreruperea procesării oricărei lucrări, care poate fi reluată într-o altă zi, de către o altă mașină. Problema planificării lucrărilor pe mașini 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 mașini paralele se poate modela ca o problemă de flux maxim în rețeaua  $G = (N, A, c)$  construită în modul următor. Mai întâi sortăm crescător elementele mulțimii  $\{a_j, t_j \mid j = 1, \dots, J\}$  și stabilim  $P \leq 2J - 1$  intervale disjuncte determinante de elemente consecutive. Notăm cu  $T_{k\ell}$  intervalul de la începutul zilei  $k$  la începutul zilei  $\ell + 1$ . Observăm că în intervalul  $T_{k\ell}$  mulțimea lucrărilor care pot fi prelucrate nu se schimbă. În intervalul  $T_{k\ell}$  putem procesa toate lucrările  $j$  cu proprietatea că  $a_j \leq k$  și  $t_j \geq \ell + 1$ . Rețeaua  $G = (N, A, c)$  se determină astfel:  $N = \{s\} \cup N_1 \cup N_2 \cup \{t\}$ ,  $N_1 = \{j \mid j = 1, \dots, J\}$ ,  $N_2 = \{T_{k\ell}^{(i)} \mid i = 1, \dots, P\}$ ,  $A = A_1 \cup A_2 \cup A_3$ ,  $A_1 = \{(s, j) \mid j \in N_1\}$ ,  $A_2 = \{(j, T_{k\ell}) \mid j \in N_1, T_{k\ell} \in N_2, a_j \leq k, t_j \geq \ell + 1\}$ ,  $A_3 = \{(T_{k\ell}, t) \mid T_{k\ell} \in N_2\}$ ,  $c(s, j) = p_j$ ,  $(s, j) \in A_1$ ,  $c(j, T_{k,\ell}) = \ell - k + 1$ ,  $(j, T_{k\ell}) \in A_2$ ,  $c(T_{k\ell}, t) = (\ell - k + 1)M$ . Apoi se determină un flux maxim în rețeaua  $G$ . Este ușor de arătat că există o corespondență biunivocă între planificările admisibile și 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 mașini paralele are soluții, altfel nu.

**Exemplul 1.9.** 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 apariției ( $a_j$ )	3	1	3	5
Termenul limită ( $t_j$ )	5	4	7	9

Mulțimea datelor  $\{a_j, t_j \mid j = 1, \dots, 4\}$  sortată crescător este  $\{1, 3, 4, 5, 7, 9\}$ . Deci, intervalele disjuncte vor fi  $T_{12}, T_{33}, T_{44}, T_{56}$  și  $T_{78}$ .

Rețeaua  $G = (N, A, c)$  este reprezentată în figura 1.21.

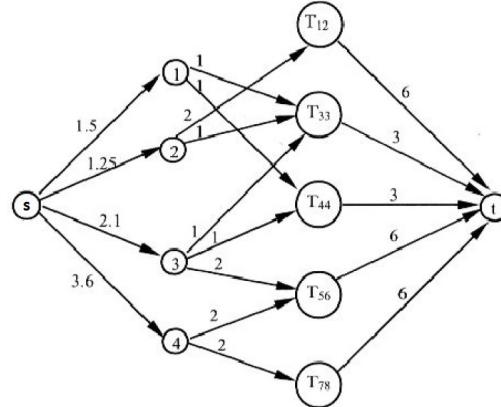


Fig.1.21

Un flux maxim este reprezentat în rețeaua din figura 1.22.

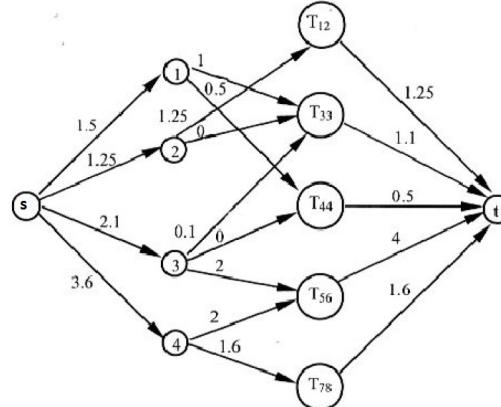


Fig.1.22

Deoarece valoarea fluxului maxim, 8.45, este egală cu  $\sum_{j=1}^4 p_j$ , rezultă că există o planificare admisibilă a lucrarilor. O planificare corespunzătoare fluxului din figura 1.22 este următoarea: prima mașină procesează începând din ziua 1 lucrarea 2 până la terminarea ei, apoi începând din ziua 3 lucrarea 1 până la terminarea ei și apoi începând din ziua 5 lucrarea 4 până la terminarea ei. A doua mașină procesează o parte (0.1) din lucrarea 3 începând din ziua 3, iar a treia mașină va începe procesarea lucrării 3 în ziua 5 și o va termina în ziua 6.

## Capitolul 2

# Algoritmi polinomiali pentru fluxul maxim

### 2.1 Algoritmi polinomiali cu drumuri de mărire a fluxului

#### 2.1.1 Etichete distanță

**Definiția 2.1.** Într-o rețea reziduală  $\tilde{G}(f) = (\tilde{N}, \tilde{A}, r)$  o funcție  $d : \tilde{N} \rightarrow \mathcal{N}$  se numește *funcția distanță*. Funcția distanță  $d$  se numește *validă* dacă satisfac următoarele două condiții:

$$d(t) = 0, \quad (2.1)$$

$$d(x) \leq d(y) + 1, \quad (x, y) \in \tilde{A}. \quad (2.2)$$

Valoarea  $d(x)$  se numește *eticheta distanță* a nodului  $x$  și condițiile (2.1), (2.2) se numesc *condiții de validitate*.

**Proprietatea 2.1.** Dacă etichetele distanță sunt valide, atunci eticheta distanță  $d(x)$  este o margine inferioară pentru lungimea drumului cel mai scurt de la nodul  $x$  la nodul  $t$  în rețeaua reziduală  $\tilde{G}(f)$ .

**Demonstrație.** Fie  $D = (x_1, x_2, \dots, x_k, x_{k+1})$  cu  $x_1 = x, x_{k+1} = t$  un drum oarecare de lungime  $k$  de la nodul  $x$  la nodul  $t$  din rețeaua reziduală  $\tilde{G}(f)$ . Condițiile 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 afirmația proprietății. ■

**Proprietatea 2.2.** Dacă  $d(s) \geq n$ , atunci rețeaua reziduală  $\tilde{G}(f)$  nu conține drum de la nodul sursă  $s$  la nodul stoc  $t$ .

**Demonstrație.** Eticheta distanță  $d(s)$  este o margine inferioară pentru lungimea drumului cel mai scurt de la  $s$  la  $t$  în rețeaua reziduală  $\tilde{G}(f)$ . Orice drum elementar de la  $s$  la  $t$  are lungimea cel mult  $n - 1$ . Deci dacă  $d(s) \geq n$  rețeaua reziduală  $\tilde{G}(f)$  nu

conține drum de la nodul  $s$  la  $t$ . ■

**Definiția 2.2.** Etichetele distanță 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 rețeaua reziduală  $\tilde{G}(f)$ .

**Exemplul 2.1.** Se consideră rețeaua din figura 2.1 în care  $s = 1$  și  $t = 4$ .

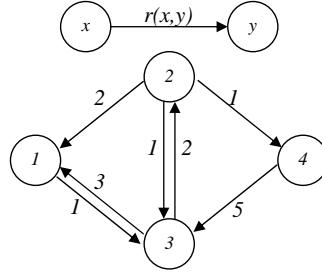


Fig.2.1

Un vector valid de etichete distanță este  $d = (0, 0, 0, 0)$  și vectorul de etichete distanță exacte este  $d = (3, 1, 2, 0)$ . Etichetele distanță exacte se pot determina cu ajutorul algoritmului parcurgerii inverse BF aplicat rețelei reziduale  $\tilde{G}(f)$  plecând de la nodul stoc  $t$ . Acest algoritm are complexitatea  $O(m)$ .

**Definiția 2.3.** Un arc  $(x, y)$  din rețeaua reziduală  $\tilde{G}(f)$  se numește *admisibil* dacă el satisface condiția  $d(x) = d(y) + 1$ , altfel arcul  $(x, y)$  se numește *inadmisibil*. Un drum de la nodul sursă  $s$  la nodul stoc  $t$  din rețeaua reziduală  $\tilde{G}(f)$  se numește *drum admisibil* dacă conține numai arce admisibile, altfel se numește *inadmisibil*.

**Proprietatea 2.3.** În rețeaua reziduală  $\tilde{G}(f)$  un drum admisibil de la nodul sursă  $s$  la nodul stoc  $t$  este un cel mai scurt drum de mărire a fluxului.

**Demonstrație.** Deoarece fiecare arc  $(x, y)$  dintr-un drum admisibil  $D$  este admisibil, capacitatea reziduală  $r(x, y)$  și etichetele distanță  $d(x), d(y)$  verifică condițiile: (1)  $r(x, y) > 0$ , (2)  $d(x) = d(y) + 1$ . Condiția (1) implică faptul că  $D$  este un drum de mărire a fluxului. Condiția (2) implică faptul că dacă  $D$  conține  $k$  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 rețeaua reziduală  $\tilde{G}(f)$ , drumul  $D$  trebuie să fie un cel mai scurt drum de mărire a fluxului. ■

### 2.1.2 Algoritmul Gabow al scalării bit a capacitatii

**Teorema 2.4.** Dacă  $f_1^*$  este un flux maxim de valoare  $v_1^*$  pentru capacitatele 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 rețeaua  $G = (N, A, c)$ ;
- (ii) în rețeaua reziduală  $\tilde{G}(2f_1^*)$  există cel mult  $m$  drumuri de mărire a fluxului și capacitatele reziduale ale lor sunt egale cu 1.

**Demonstrație.** (i) Fie  $[X^*, \bar{X}^*]$  o tăietură minimă  $s - t$  în rețeaua  $G = (N, A, c)$ . Deoarece  $c(x, y) \leq 2c_1(x, y) + 1$  rezultă  $v^* = c(X^*, \bar{X}^*) \leq 2c_1(X^*, \bar{X}^*) + m = 2v_1^* + m$  și deci  $v^* - 2v_1^* \leq m$ ;

(ii) Dacă  $f_1^*$  este un flux maxim pentru capacitatea  $c_1$  atunci în rețeaua reziduală

$\tilde{G}(2f_1^*)$  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  $c_0(x, y) = c(x, y)$ ,  $c_k(x, y) = \lfloor c_{k-1}(x, y)/2 \rfloor$ ,  $(x, y) \in A$ ,  $k = 1, \dots, \lfloor \log \bar{c} \rfloor$ , atunci notăm cu  $G_k$  rețeaua care are capacitatea  $c_k$ ,  $G_k = (N, A, c_k)$  și cu  $f_k$  un flux oarecare,  $f_k^*$  un flux maxim în  $G_k$ . Algoritmul Gabow al scalării bit a capacitații este următorul:

```

(1)  PROGRAM SCALARE BIT;
(2)  BEGIN
(3)    k := 0; c0 = c;
(4)    WHILE max{ck(x, y) | (x, y) ∈ A} > 1 DO
(5)      BEGIN
(6)        FOR (x, y) ∈ A DO
(7)          ck+1(x, y) := ⌊ ck(x, y)/2 ⌋;
(8)          k := k + 1;
(9)      END;
(10)     fk+1 := 0;
(11)     WHILE k ≥ 0 DO
(12)       BEGIN
(13)         fluxul inițial este 2fk+1^*;
(14)         se determină fk^* în Gk;
(15)         k := k - 1;
(16)       END;
(17)     END.

```

Conform definiției de mai sus avem  $G_0 = G$ . Afirmațiile Teoremei 2.4 sunt demonstreate pentru  $G_0$  și  $G_1$ . Evident că aceste afirmații rămân adevărate pentru  $G_{k-1}$  și  $G_k$ ,  $k = 1, \dots, \lfloor \log \bar{c} \rfloor$ .

**Teorema 2.5. (Teorema de corectitudine a algoritmului)** *Algoritmul Gabow al scalării bit a capacitații determină un flux maxim  $f^*$  în rețeaua  $G = (N, A, c)$ .*

**Demonstrație.** Inițial algoritmul determină un flux maxim  $f_k^*$  în rețeaua  $G_k = (N, A, c_k)$  pornind cu  $2f_{k+1}^* = 0$ ,  $k = \lfloor \log \bar{c} \rfloor$ . Algoritmul se termină când  $k < 0$ . La ultima iterație se calculează un flux maxim  $f_0^*$  în  $G_0 = G$ . Deci  $f^* = f_0^*$  este un flux maxim în  $G$ . ■

**Teorema 2.6. (Teorema de complexitate a algoritmului).** *Algoritmul Gabow al scalării bit a capacitații are complexitatea  $O(m^2 \log \bar{c})$ .*

**Demonstrație.** La fiecare iterație  $k$  algoritmul rezolvă o problemă  $P_k$  de flux maxim în rețeaua  $G_k$ ,  $k = \lfloor \log \bar{c} \rfloor, \dots, 0$ . Conform Teoremei 2.4, rețeaua reziduală  $\tilde{G}(2f_{k+1}^*)$  conține cel mult  $m$  drumuri  $D_k$  de mărire a fluxului cu capacitațile 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  $O(m^2)$ . Deoarece  $k = \lfloor \log \bar{c} \rfloor, \dots, 0$  se obține că algoritmul are complexitatea  $O(m^2 \log \bar{c})$ . ■

Algoritmul scalării bit a capacitații se datorează lui Gabow (1985) și de aceea se numește *algoritmul Gabow al scalării bit a capacitații*.

**Exemplul 2.2.** Să considerăm rețeaua din figura 2.2.

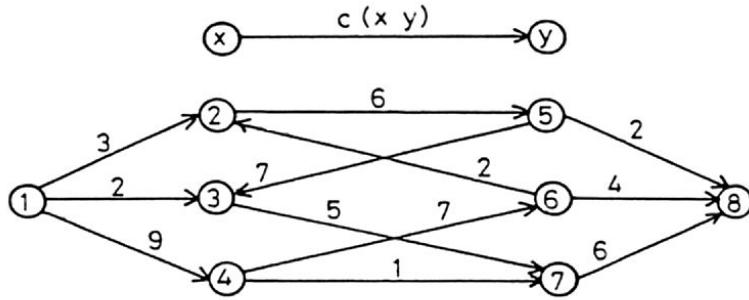


Fig. 2.2

Tabelul 2.1. Capacitățile  $c_k$  și fluxurile  $f_k^*$ ;  $f^* = f_0^*$  este fluxul maxim

$(x, y)$	(1, 2)	(1, 3)	(1, 4)	(2, 5)	(3, 7)	(4, 6)	(4, 7)	(5, 3)	(5, 8)	(6, 2)	(6, 8)	(7, 8)
$C_0$	3	2	9	6	5	7	1	7	2	2	4	6
$C_1$	1	1	4	3	2	3	0	3	1	1	2	3
$C_2$	0	0	2	1	1	1	0	1	0	0	1	1
$C_3$	0	0	1	0	0	0	0	0	0	0	0	0
$f_3^*$	0	0	0	0	0	0	0	0	0	0	0	0
$f_2^*$	0	0	1	0	0	1	0	0	0	0	1	0
$f_1^*$	1	1	3	2	2	3	0	1	1	1	2	2
$f_0^*$	3	2	7	5	5	6	1	3	2	2	4	6

### 2.1.3 Algoritmul Ahuja-Orlin al scalării maxime a capacitatei

Ideea esențială ce fundamentează algoritmul Ahuja-Orlin al scalării maxime a capacitatei 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 obține un drum de mărire a fluxului cu capacitatea reziduală suficient de mare mult mai ușor decât un drum de mărire a fluxului cu capacitatea reziduală maximă.

**Definiția 2.4.** Fie  $f$  un flux oarecare de valoare  $v$  în rețeaua  $G = (N, A, c)$ . O rețea reziduală  $\tilde{G}(f, \bar{r}) = (N, \tilde{A}(\bar{r}))$  cu  $\tilde{A}(\bar{r}) = \{(x, y) \mid (x, y) \in \tilde{A}, r(x, y) \geq \bar{r}\}$  se numește *rețea reziduală*  $\bar{r}$ .

Remarcăm faptul că  $\tilde{G}(f, 1) = \tilde{G}(f)$ .

**Exemplul 2.3.** Figura 2.3 (a) arată rețeaua reziduală  $\tilde{G}(f)$  și figura 2.3 (b) arată rețeaua reziduală  $\tilde{G}(f, \bar{r})$  cu  $\bar{r} = 8$ .

**Definiția 2.5.** O fază a algoritmului pe parcursul căreia  $\bar{r}$  rămâne constant se numește *fază de scalare* și o fază de scalare cu valoarea specificată a lui  $\bar{r}$  se numește *fază de scalare*  $\bar{r}$ .

Într-o fază de scalare  $\bar{r}$  fiecare drum  $D$  de mărire a fluxului are capacitatea reziduală  $r(D) \geq \bar{r}$ .

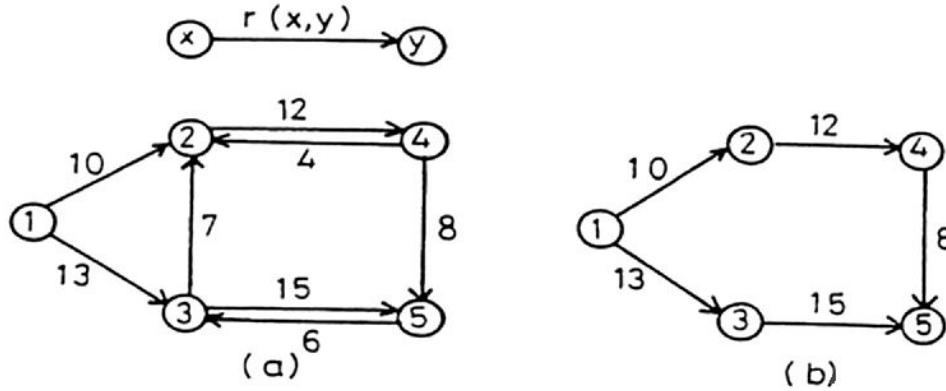


Fig.2.3

```

(1) PROGRAM SCALARE MAXIMĂ;
(2) BEGIN
(3)    $f := 0;$ 
(4)    $\bar{r} := 2^{\lfloor \log \bar{c} \rfloor};$ 
(5)   WHILE  $\bar{r} \geq 1$  DO
(6)     BEGIN
(7)       WHILE  $\tilde{G}(f, \bar{r})$  conține drum de la  $s$  la  $t$  DO
(8)         BEGIN
(9)           se identifică un drum  $\tilde{D}$  de la  $s$  la  $t$  în  $\tilde{G}(f, \bar{r})$ ;
(10)           $r(\tilde{D}) := \min\{r(x, y) \mid (x, y) \in \tilde{D}\};$ 
(11)          se execută mărirea de flux;
(12)          se actualizează  $\tilde{G}(f, \bar{r})$ ;
(13)        END;
(14)         $\bar{r} := \bar{r}/2;$ 
(15)      END;
(16)    END.

```

**Teorema 2.7. (Teorema de corectitudine a algoritmului)** *Algoritmul Ahuja-Orlin al scalării maxime a capacitatei determină un flux maxim  $f^*$  în rețeaua  $G = (N, A, c)$ .*

**Demonstrație.** Algoritmul pornește cu  $\bar{r} := 2^{\lfloor \log \bar{c} \rfloor}$  și înjumătățește valorile lui  $\bar{r}$  până când  $\bar{r} = 1$ . În consecință algoritmul execută  $1 + \lfloor \log \bar{c} \rfloor = O(\log \bar{c})$  faze de scalare. În ultima fază de scalare,  $\bar{r} = 1$ , astfel  $\tilde{G}(f, \bar{r}) = \tilde{G}(f)$ . Acest rezultat arată că algoritmul se termină cu un flux maxim  $f^*$  în rețeaua  $G$ . ■

**Teorema 2.8. (Teorema de complexitate a algoritmului)** *Algoritmul Ahuja - Orlin al scalării maxime a capacitatei are complexitatea  $O(m^2 \log \bar{c})$ .*

**Demonstrație.** Fie faza de scalare  $\bar{r}_k$  cu fluxul  $f_k^*$  de valoare  $v_k^*$  la sfârșitul acestei faze. Fie  $X_k^*$  mulțimea nodurilor atinse din  $s$  în  $\tilde{G}(f_k^*, \bar{r}_k)$ . Deoarece  $\tilde{G}(f_k^*, \bar{r}_k)$  nu conține drum de mărire a fluxului de la  $s$  la  $t$ , rezultă că  $t \in \overline{X}_k^*$ . Deci  $[X_k^*, \overline{X}_k^*]$  este o tăietură  $s - t$  în rețeaua  $G = (N, A, c)$  cu  $r(X_k^*, \overline{X}_k^*) \leq m\bar{r}_k$ , deoarece  $r(x, y) <$

$\bar{r}_k, (x, y) \in (X_k^*, \bar{X}_k^*)$ . Dacă  $f^*$  de valoare  $v^*$  este un flux maxim în rețeaua  $G$ , atunci conform Teoremei 1.7. avem  $v^* - v_k^* \leq m\bar{r}_k$ . În faza de scalare  $\bar{r}_{k+1}$  fiecare drum  $D_{k+1}$  de mărire a fluxului are capacitatea reziduală  $r(D_{k+1}) \geq \bar{r}_k/2$ . 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  $\bar{r}_k$  are complexitatea  $O(m^2)$  și deci algoritmul are complexitatea  $O(m^2 \log \bar{c})$ . ■

Remarcăm faptul că este posibil de a reduce complexitatea algoritmilor scalării capacitatii la  $O(mn \log \bar{c})$  dacă se determină un drum de mărire a fluxului cu algoritmul descris în secțiunea următoare. Deși algoritmii scalării au aceeași complexitate, algoritmul scalării maxime a capacitatii este, din punct de vedere practic, mai performant decât algoritmul scalării bit a capacitatii.

Algoritmul scalării maxime a capacitatii se datorează lui Ahuja și Orlin (1991) și de aceea se numește *algoritmul Ahuja - Orlin al scalării maxime a capacitatii*.

#### 2.1.4 Algoritmul Edmonds - Karp al drumului celui mai scurt

Algoritmul de etichetare Ford - Fulkerson are o complexitate  $O(mn\bar{c})$ , deci o complexitate timp pseudopolynomială. Edmonds și Karp (1972) au obținut o variantă îmbunătățită a acestui algoritm cu o complexitate timp polinomială. În algoritmul de etichetare Ford - Fulkerson nodurile etichetate și neanalyzate din lista  $\tilde{V}$  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 rețelei reziduale mai întâi în lățime 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 numește algoritmul Edmonds - Karp al drumului celui mai scurt. Acest algoritm se obține din algoritmul de etichetare Ford - Fulkerson printr-o minoră modificare: lista nodurilor etichetate și neanalyzate  $\tilde{V}$  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  $O(m^2n)$ .

**Exemplul 2.4.** Să ilustrăm algoritmul Edmonds - Karp al drumului celui mai scurt pe rețeaua reprezentată în figura 2.4 (a). Fluxul inițial este  $f_0 = (f(1, 2), f(1, 3), f(2, 3), f(2, 4), f(3, 4)) = (1, 0, 1, 0, 1)$ . Rețeaua reziduală  $\tilde{G}(f_0)$  este reprezentată în figura 2.4 (b).

Se etichetează nodul  $s = 1$  și se stabilește  $\tilde{V} = \{1\}$ . Se scoate nodul 1 din  $\tilde{V}$  pentru analizare. Se etichetează nodurile 2, 3 și se obține lista  $\tilde{V} = \{2, 3\}$ , în această ordine. Vectorul predecesor este  $\tilde{p} = (4, 1, 1, 0)$ . Se scoate nodul 2 din  $\tilde{V}$  pentru analizare. Se etichetează nodul  $t = 4$ , lista  $\tilde{V}$  devine  $\tilde{V} = \{3, 4\}$  și vectorul predecesor devine  $\tilde{p} = (4, 1, 1, 2)$ . Deci  $\tilde{D} = (1, 2, 4)$  cu  $r(\tilde{D}) = u - 1$ . Se execută mărirea de flux și se continuă până când se determină un flux maxim.

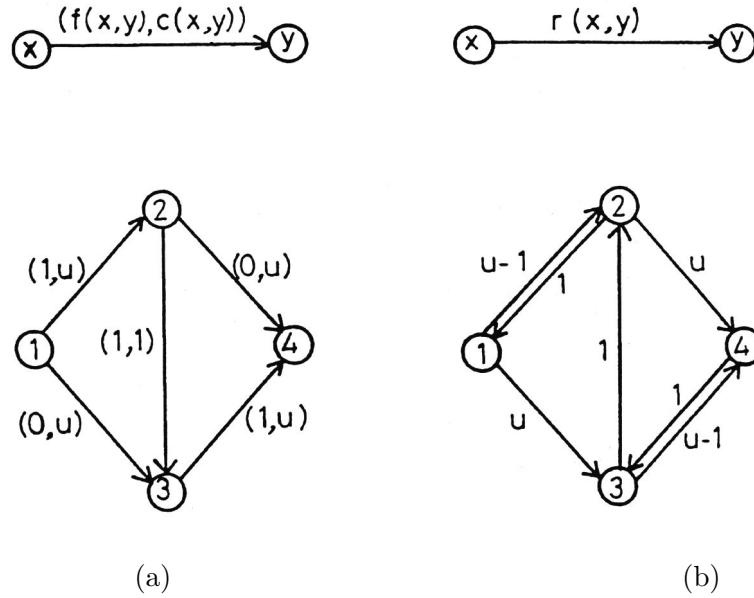


Fig.2.4

### 2.1.5 Algoritmul Ahuja - Orlin al drumului celui mai scurt

Algoritmul Ahuja - Orlin al drumului celui mai scurt este o variantă îmbunătățită a algoritmului Edmonds - Karp a drumului celui mai scurt. Algoritmul se datorează lui Ahuja și Orlin (1991) și se bazează pe etichetele distanță. Algoritmul Ahuja - Orlin al drumului celui mai scurt (algoritmul AODS) mărește fluxul de-a lungul drumurilor admisibile.

**Definiția 2.6.** În rețeaua reziduală, un drum de la nodul sursă  $s$  la un nod oarecare  $x$  constând numai din arce admisibile se numește *drum parțial admisibil*. Nodul  $x$  se numește *nod curent*.

Reamintim ipoteza 4 din paragraful 1.2 potrivit căreia, dacă arcul  $(x, y) \in A$  și  $(y, x) \notin A$ , 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 distanță 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)     $\tilde{p}(y) := x;$ 
(4)     $x := y;$ 
(5)  END;

(1)  PROCEDURA INAPOIERE( $x$ );
(2)  BEGIN
(3)     $d(x) := \min\{d(y) + 1 \mid (x, y) \in E^+(x), r(x, y) > 0\}$ 
(4)    IF  $x \neq s$ 
(5)      THEN  $x := \tilde{p}(x);$ 
(6)  END;

(1)  PROCEDURA MĂRIRE;
(2)  BEGIN
(3)    se determină DMF  $\tilde{D}$  utilizând vectorul predecesor  $\tilde{p}$ ;
(4)    se determină  $r(\tilde{D}) := \min\{r(x, y) \mid (x, y) \in \tilde{D}\};$ 
(5)    se execută mărirea de flux;
(6)    se actualizează rețeaua reziduală  $\tilde{G}(f);$ 
(7)  END;

```

Algoritmul menține un drum parțial admisibil și iterativ execută operații de înaintare sau înapoiere de la nodul curent  $x$ . Dacă nodul curent  $x$  este extremitatea inițială a unui arc admisibil  $(x, y)$  se execută o operație de înaintare și se adaugă arcul  $(x, y)$  la drumul parțial admisibil, altfel se execută o operație de înapoiere. Repetăm aceste operații până când drumul parțial admisibil atinge nodul stoc  $t$  la care timp executăm o mărire de flux. Continuăm acest proces până când fluxul devine maxim. Operația  $d(x) := \min\{d(y) + 1 \mid (x, y) \in E^+(x), r(x, y) > 0\}$  din procedura ÎNAPOIERE se numește *operația de reetichetare*.

**Exemplul 2.5.** Ilustrăm algoritmul drumului celui mai scurt Ahuja - Orlin pe exemplu dat în figura 2.5. Mai întâi calculăm vectorul etichetelor distanță exacte prin executarea parcurgerii inverse mai întâi în lățime pornind de la nodul stoc  $t = 12$  din rețeaua reziduală inițială dată în figura 2.5(a). Se obține  $d = (3, 2, 2, 2, 2, 1, 1, 1, 1, 1, 0)$ . Pornim de la nodul sursă  $s = 1$  cu un drum admisibil parțial vid. Executăm o operație de înaintare și adăugăm arcul  $(1,2)$  la drumul admisibil parțial. Memorăm acest drum utilizând vectorul predecesor  $\tilde{p}$ , astfel  $\tilde{p}(2) := 1$ . Acum nodul 2 este nodul curent și algoritmul execută o operație de înaintare de la nodul 2. Astfel, adăugăm arcul

$(2,7)$  la drumul parțial admisibil, care acum devine  $1-2-7$ . De asemenea  $\tilde{p}(7) := 2$ . În continuare algoritmul adaugă arcul  $(7,12)$  la drumul admisibil parțial obținând  $\tilde{D}_1 = (1, 2, 7, 12)$ , care este un drum admisibil la nodul stoc. Executăm o mărire de flux cu  $r(\tilde{D}_1) = \min\{r(1, 2), r(2, 7), r(7, 12)\} = \min\{2, 1, 2\} = 1$ . Rețeaua reziduală după prima mărire de flux este arătată în figura 2.5.(b).

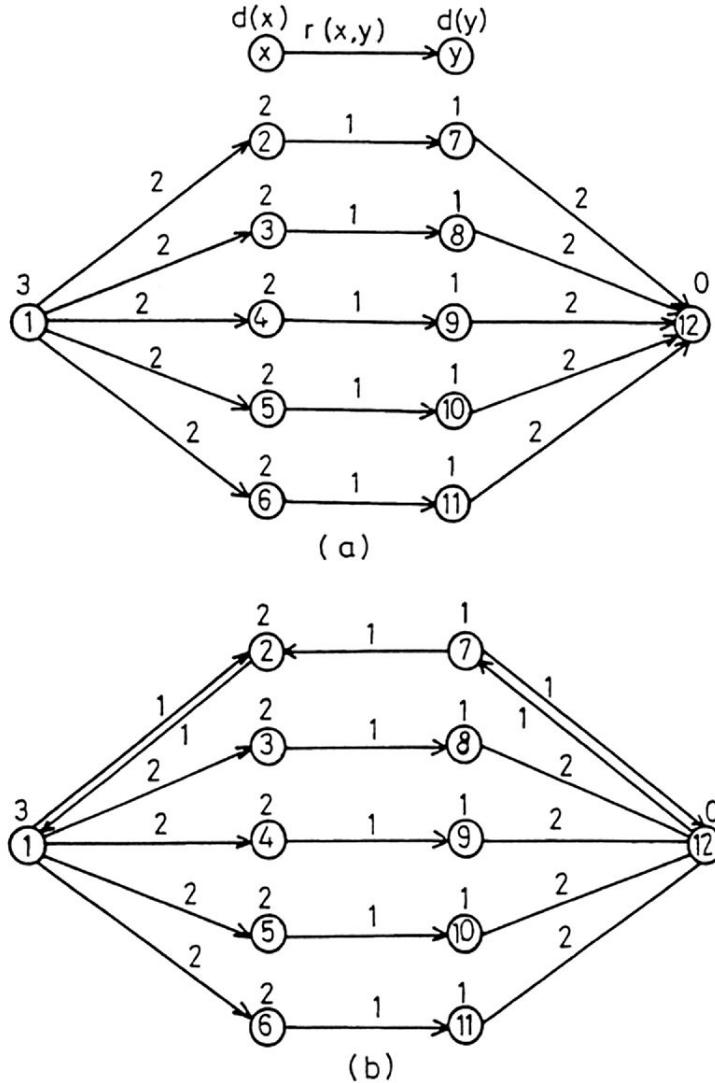


Fig.2.5

Din nou pornim de la nodul sursă  $s = 1$  cu un drum admisibil parțial vid. Algoritmul adaugă arcul  $(1, 2)$ ,  $\tilde{p}(2) := 1$  și nodul 2 devine nodul curent. Nodul 2 nu este extremitatea inițială a nici unui arc admisibil. Efectuăm o operație de înapoiere în cadrul căreia efectuăm operația de reetichetare  $d(2) := \min\{d(y) + 1 \mid (2, y) \in E^+(2), r(2, y) > 0\} = \min\{d(1) + 1\} = 4$ ,  $x := \tilde{p}(2) = 1$ , astfel arcul  $(1, 2)$  este eliminat din drumul admisibil parțial deoarece după operația de reetichetare a nodului 2

arcul  $(1,2)$  devine inadmisibil. În iterațiile următoare, algoritmul identifică drumurile admisibile  $\tilde{D}_2 = (1, 3, 8, 12)$ ,  $\tilde{D}_3 = (1, 4, 9, 12)$ ,  $\tilde{D}_4 = (1, 5, 10, 12)$ ,  $\tilde{D}_5 = (1, 6, 11, 12)$  și mărим fluxul cu  $r(\tilde{D}_i) = 1$ ,  $i = 2, 3, 4, 5$  pe aceste drumuri. Astfel se ajunge la un flux maxim.

**Lema 2.9.** *Algoritmul AODS păstrează validitatea etichetelor distanță după fiecare operație de înaintare, mărire de flux sau operație de înapoiere. Fiecare operație de reetichetare crește strict eticheta distanță a nodului curent.*

**Demonstratie.** Operația de înaintare nu modifică nici capacitatele reziduale nici etichetele distanță și deci păstrează validitatea etichetelor distanță.

Arătăm că algoritmul menține valide etichetele distanță după fiecare mărire de flux sau fiecare operație de înapoiere prin aplicarea inducției pe numărul de măriri de flux respectiv pe numărul de operații de înapoiere.

Mai întâi să presupunem că după  $k$  măriri de flux etichetele distanță sunt valide. Să arătăm că etichetele distanță rămân valide și după  $k+1$  măriri de flux. Deoarece în procedura mărire nu se fac modificări asupra etichetelor distanță, mărirea de flux  $k+1$  pe un arc  $(x, y)$  afectează condițiile 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  $d(y) = d(x) - 1 < d(x) + 1$  și prin urmare condiția de validitate pentru arcul  $(y, x)$  este verificată.

Acum să presupunem că după  $k$  operații de înapoiere eticheta distanță  $d(x)$  verifică condițiile de validitate:  $d(x) \leq d(y) + 1$ . La operația de înapoiere  $k+1$  eticheta distanță  $d(x)$  devine  $d'(x) := \min\{d(y) + 1 \mid (x, y) \in E^+(x), r(x, y) > 0\}$ . Deci  $d'(x) \leq d(y) + 1$  și  $d'(x)$  verifică condițiile de validitate. Algoritmul efectuează o operație de înapoiere de la nodul curent  $x$  dacă nu există arc  $(x, y)$  cu  $d(x) = d(y) + 1$  și  $r(x, y) > 0$ . Rezultă  $d(x) < d(y) + 1$  și prin urmare  $d(x) < \min\{d(y) + 1 \mid (x, y) \in E^+(x), r(x, y) > 0\} = d'(x)$ , relație care arată că fiecare operație de reetichetare crește strict eticheta distanță a nodului curent  $x$ . ■

**Teorema 2.10. (Teorema de corectitudine a algoritmului)** *Algoritmul AODS calculează un flux maxim în rețeaua  $G = (N, A, c)$ .*

**Demonstratie.** Algoritmul Ahuja - Orlin al drumului celui mai scurt se termină când  $d(s) \geq n$ . Conform Proprietății 2.2 rezultă că rețeaua reziduală nu conține 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 numește *structură de date arc current*. Reamintim că lista arcelor care au pe  $x$  ca extremitate inițială 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 executiei algoritmului. Arcul  $(x, y) \in E^+(x)$  care urmează a fi testat pentru condiția de admisibilitate se numește *arc current* al nodului  $x$ . Inițial, arcul current al nodului  $x$  este primul arc din lista  $E^+(x)$ . Dacă arcul current din  $E^+(x)$  este un arc admisibil atunci algoritmul execută o operație, altfel algoritmul indică arcul următor din  $E^+(x)$  ca arc current. Algoritmul execută acest proces până când fie găsește un arc admisibil, fie se ajunge la sfârșitul listei  $E^+(x)$  după care execută o operație.

**Exemplul 2.6.** Să considerăm lista arc a nodului 1 din figura 2.6.

În acest caz  $E^+(1) = \{(1, 2), (1, 3), (1, 4), (1, 5), (1, 6)\}$ . Inițial, 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 și 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  $d(4)$  crește prin operația de reetichetare.

**Lema 2.11.** *Dacă algoritmul AODS reetichetează orice nod de cel mult  $k$  ori, timpul necesar pentru determinarea arcelor admisibile și reetichetarea nodurilor este  $O(k \sum_N |E^+(x)|) = O(km)$ .*

**Demonstrație.** 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  $O(\sum_N |E^+(x)|) = O(m)$ . Ținând seama de faptul că orice nod se reetichetează de cel mult  $k$  ori rezultă afirmația lemei. ■

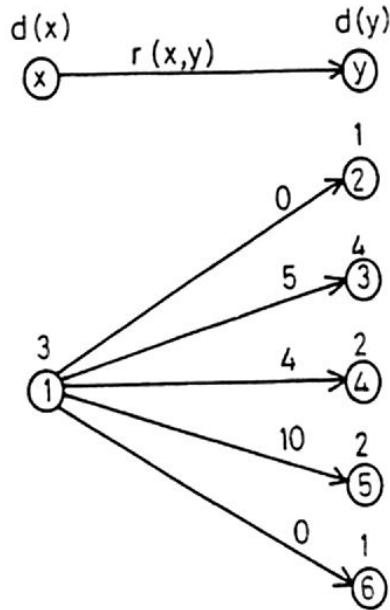


Fig.2.6

**Lema 2.12** *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.*

**Demonstrație.** Presupunem că o mărire de flux saturează arcul  $(x, y)$ . Deoarece acest arc este admisibil avem

$$d(x) = d(y) + 1 \quad (2.3)$$

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 distanță  $d'(x)$  și  $d'(y)$  verifică

$$d'(y) = d'(x) + 1 \quad (2.4)$$

În următoarea saturație a arcului  $(x, y)$ , trebuie să avem

$$d''(x) = d''(y) + 1 \quad (2.5)$$

Utilizând Lema 2.9 se obține  $d''(x) = d''(y) + 1 \geq d'(y) + 1 = d'(x) + 2 \geq d(x) + 2$ . Similar se arată că  $d''(y) \geq d(y) + 2$ . Deci, între două saturații consecutive ale arcului  $(x, y)$ , ambele etichete distanță  $d(x)$  și  $d(y)$  cresc prin cel puțin două unități. Deoarece, prin ipoteză, algoritmul crește fiecare etichetă distanță de cel mult  $k$  ori rezultă că algoritmul poate satura fiecare arc de cel mult  $k/2$  ori. Deci, numărul total de saturații arc va fi de cel mult  $km/2$  ori. ■

**Lema 2.13.** În execuția algoritmului AODS numărul operațiilor de înapoiere este cel mult  $n^2$ , iar numărul măririlor de flux este cel mult  $mn/2$ .

**Demonstrație.** Fiecare operație de reetichetare a nodului  $x$  crește valoarea lui  $d(x)$  cu cel puțin o unitate. După ce algoritmul a reetichetat nodul  $x$  de cel mult  $n$  ori, avem  $d(x) \geq n$  și algoritmul nu va mai selecta niciodată nodul  $x$  pentru o operație de înaintare, deoarece pentru fiecare nod  $z$  din drumul admisibil parțial avem  $d(z) < d(s) < n$ . Astfel algoritmul reetichetează un nod  $x$  de cel mult  $n$  ori și numărul total de operații de reetichetare este cel mult  $n^2$ . Fiecare operație de înapoiere reetichetează un nod, astfel numărul operațiilor de înapoiere este  $O(n^2)$ . Conform Lemei 2.12 și 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 puțin un arc rezultă că numărul măririlor de flux este cel mult  $mn/2$ . ■

**Teorema 2.14. (Teorema de complexitate a algoritmului)** Algoritmul AODS are complexitatea  $O(mn^2)$ .

**Demonstrație.** Conform Lemei 2.11 și Lemei 2.13 determinarea arcelor admisibile și 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 2.13 numărul operațiilor de înapoiere este  $O(n^2)$ . Înținând seama că numărul măririlor de flux este  $O(mn)$ , un drum admisibil are lungimea cel mult  $n$  și că numărul operațiilor de înapoiere este  $O(n^2)$  rezultă că numărul operațiilor de înaintare este  $O(mn^2 + n^2)$ . Combinarea acestor margini stabilește 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 obținut un flux maxim. Acest lucru se întâmplă deoarece algoritmul nu are un criteriu de testare dacă s-a obținut un flux maxim. În continuare descriem o tehnică de testare a prezenței unei tăieturi minime și deci testarea existenței unui flux maxim mult înainte ca  $d(s) \geq n$ . Introducerea acestei tehnici îmbunătățește performanța practică a algoritmului AODS.

Introducem tabloul unidimensional  $q = (q(0), \dots, q(n-1))$ . Valoarea  $q(k)$  este numărul de noduri care au etichetele distanță egale cu  $k$ . Algoritmul inițializează tabloul  $q := 0$  și îl actualizează după ce calculează etichetele distanță exacte. După fiecare operație de reetichetare actualizăm tabloul  $q$  în modul următor. Dacă algoritmul mărește eticheta distanță 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  $X^* = \{x \mid x \in N, d(x) > k_1\}$ ,  $\bar{X}^* = \{\bar{x} \mid \bar{x} \in N, d(\bar{x}) < k_1\}$ . Evident că  $s \in X^*$  și  $t \in \bar{X}^*$ . Deci  $[X^*, \bar{X}^*]$  este o tăietură  $s - t$

și  $d(x) > d(\bar{x}) + 1$ ,  $(x, \bar{x}) \in (X^*, \bar{X}^*)$ . Rezultă că  $r(x, \bar{x}) = 0$ ,  $(x, \bar{x}) \in (X^*, \bar{X}^*)$ . Prin urmare  $[X^*, \bar{X}^*]$  este o tăietură minimă și fluxul curent este un flux maxim.

**Exemplul 2.7.** Ilustrăm această tehnică prin aplicarea ei la exemplul prezentat la ilustrarea algoritmului AODS. Figura 2.7 prezintă rețeaua reziduală imediat după ultima mărire de flux.

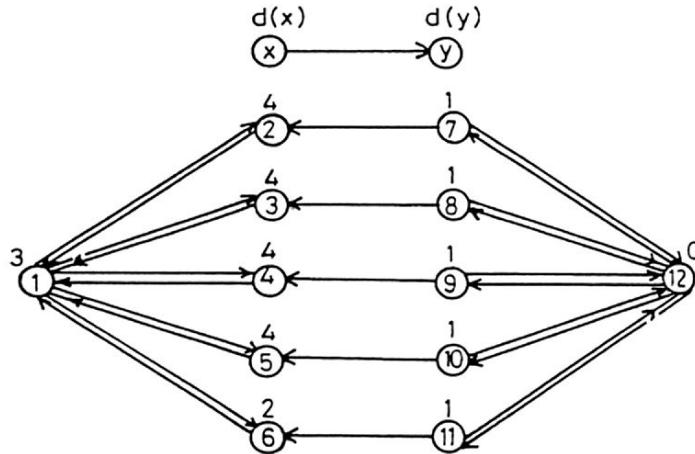


Fig.2.7

Deși fluxul este acum un flux maxim și cu toate că nodul sursă nu este conectat la nodul stoc printr-un drum în rețeaua reziduală, criteriul de terminare  $d(1) \geq 12$  este departe de a fi satisfăcut. Se poate verifica faptul că, în continuare, algoritmul va crește etichetele distanță ale nodurilor 6,1,2,3,4,5, în această ordine, fiecare prin 2 unități. În cele din urmă  $d(1) \geq 12$  și algoritmul se termină. Dacă folosim tabloul  $q$ , atunci pentru etichetele distanță arătate în figura 2.7 avem  $q = (1, 5, 1, 1, 4, 0, 0, 0, 0, 0, 0, 0)$ . Când continuăm algoritmul după ultima mărire de flux el construiește drumul admisibil parțial 1- 6. În continuare el reetichetează nodul 6 și  $d(6) = 2$  crește la  $d(6) = 4$ . În acest caz  $q(2) = 0$  și algoritmul se termină.

### 2.1.6 Algoritmul Dinic al rețelelor stratificate

Mulți algoritmi pentru problema fluxului maxim măresc fluxul de-a lungul drumurilor celor mai scurte de la nodul sursă  $s$  la nodul stoc  $t$  din rețeaua reziduală. Dinic (1970) introduce conceptul de *rețea stratificată*. Rețeaua stratificată Dinic  $\tilde{G}'(f) = (\tilde{N}', \tilde{A}', \bar{r}')$  în raport cu fluxul  $f$  se definește cu ajutorul *funcției distanță complementară*  $\bar{d}$ :  $\tilde{N} \rightarrow \mathcal{N}$ , definită în rețeaua reziduală  $\tilde{G}(f) = (\tilde{N}, \tilde{A}, r)$ .

**Definiția 2.7.** Funcția distanță complementară  $\bar{d}$  se numește *validă* dacă satisfac următoarele două condiții:

$$\bar{d}(s) = 0, \quad (2.6)$$

$$\bar{d}(y) \leq \bar{d}(x) + 1, \quad (x, y) \in \tilde{A}, \quad (2.7)$$

Valoarea  $\bar{d}(x)$  se numește *eticheta distanță complementară* a nodului  $x$  și condițiile (2.6), (2.7) se numesc *condiții de validitate*.

**Proprietatea 2.15.** Dacă etichetele distanță complementară sunt valide, atunci eticheta distanță complementară  $\bar{d}(x)$  este o margine inferioară pentru lungimea drumului cel mai scurt de la nodul  $s$  la nodul  $x$  în rețeaua reziduală  $\tilde{G}(f)$ .

**Demonstrație.** Se demonstrează la fel ca Proprietatea 2.1. ■

**Proprietatea 2.16.** Dacă  $\bar{d}(t) \geq n$ , atunci rețeaua reziduală  $\tilde{G}(f)$  nu conține drum de la nodul sursă  $s$  la nodul stoc  $t$ .

**Demonstrație.** Se demonstrează la fel ca Proprietatea 2.2. ■

**Definiția 2.8.** Etichetele distanță complementară se numesc *exacte* dacă pentru fiecare nod  $x$ ,  $\bar{d}(x)$  este egală cu lungimea drumului cel mai scurt de la nodul  $s$  la nodul  $x$  în rețeaua reziduală  $\tilde{G}(f)$ .

**Exemplul 2.8.** Să considerăm rețeaua reziduală din figura 2.8, în care  $s = 1$  și  $t = 4$ . Un vector al etichetelor distanță complementară valide este  $\bar{d} = (0, 0, 0, 0)$  și vectorul etichetelor distanță complementară exacte este  $\bar{d} = (0, 2, 1, 3)$ .

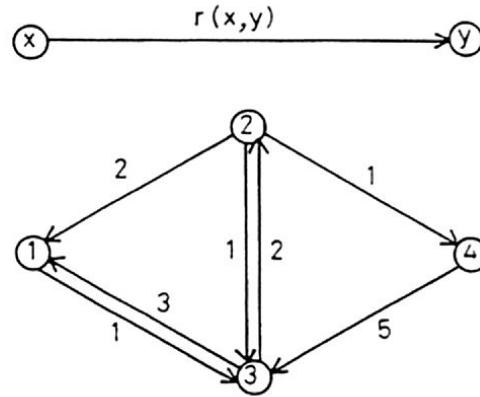


Fig.2.8

Etichetele distanță complementară exacte se pot determina cu ajutorul algoritmului parcurgerii BF aplicat rețelei reziduale  $\tilde{G}(f)$  plecând de la nodul sursă  $s$ . Acest algoritm are complexitatea  $O(m)$ .

**Definiția 2.9.** Un arc  $(x, y)$  din rețeaua reziduală  $\tilde{G}(f)$  se numește *admisibil* dacă el satisfac condiția  $\bar{d}(y) = \bar{d}(x) + 1$ , altfel arcul  $(x, y)$  se numește *inadmisibil*. Un drum de la nodul sursă  $s$  la nodul stoc  $t$  din rețeaua reziduală  $\tilde{G}(f)$  se numește *drum admisibil* dacă conține numai arce admisibile, altfel se numește *inadmisibil*.

**Proprietatea 2.17.** În rețeaua reziduală  $\tilde{G}(f)$  un drum admisibil de la nodul sursă  $s$  la nodul stoc  $t$  este un cel mai scurt drum de mărire a fluxului.

**Demonstrație.** Se demonstrează la fel ca Proprietatea 2.3. ■

Legătura dintre etichetele distanță exacte  $d(x)$  și etichetele distanță complementară exacte  $\bar{d}(x)$  este

$$d(x) + \bar{d}(x) = d(s) = \bar{d}(t), \quad x \in \tilde{D}, \quad \tilde{D} \text{ admisibil.} \quad (2.8)$$

Este evident că eticheta distanță complementară exactă  $\bar{d}(x)$  este o distanță complementară pentru eticheta distanță 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  $f$ , rețeaua stratificată Dinic  $\tilde{G}'(f) = (\tilde{N}', \tilde{A}', \bar{r}')$  se definește în modul următor. Se determină etichetele distanță complementară exakte  $\bar{d}(x)$  prin parcugerea BF a rețelei reziduale  $\tilde{G}(f) = (\tilde{N}, \tilde{A}, r)$  pornind de la nodul sursă  $s$ . Atunci avem  $\tilde{N}' = \tilde{N}$ ,  $\tilde{A}' = \{(x, y) \mid (x, y) \in \tilde{A}, \bar{d}(y) = \bar{d}(x) + 1\}$ ,  $\bar{r}'(x, y) = r(x, y)$ ,  $(x, y) \in \tilde{A}'$ . Pentru  $k = 0, \dots, \bar{d}(t)$  se definește un strat  $\tilde{N}'(k) = \{x \mid x \in \tilde{N}, \bar{d}(x) = k\}$ . Evident că  $\tilde{N}'(0) = \{s\}$  și  $\tilde{N}' = \cup \{\tilde{N}'(k) \mid k = 0, \dots, \bar{d}(t)\}$ . Dacă arcul  $(x, y) \in \tilde{A}'$  cu  $\bar{d}(x) = k$ , atunci  $x \in \tilde{N}'(k)$  și  $y \in \tilde{N}'(k+1)$ . Este evident că orice drum de la nodul sursă  $s$  la nodul stoc  $t$  din rețeaua stratificată Dinic  $\tilde{G}'(f)$  are lungimea  $\bar{d}(t)$ .

**Definiția 2.10.** Un flux  $\bar{f}$  din rețeaua stratificată Dinic  $\tilde{G}'(f)$  se numește *flux de blocare* dacă orice drum de la nodul sursă  $s$  la nodul stoc  $t$  din  $\tilde{G}'(f)$  are un arc  $(x, y)$  cu  $\bar{r}'(x, y) = 0$ .

Din definiție rezultă că dacă rețeaua stratificată Dinic  $\tilde{G}'(f)$  nu conține drum de mărire a fluxului, atunci fluxul  $\bar{f}$  este un flux de blocare.

Este posibil ca anumite arce și noduri din rețeaua stratificată Dinic  $\tilde{G}'(f)$  să nu aparțină la nici un drum de la nodul sursă  $s$  la nodul stoc  $t$ . Aceste arce și noduri pot fi eliminate din rețeaua stratificată Dinic.

Algoritmul rețelelor stratificate Dinic este următorul:

- (1) PROGRAM RETELE-STRATIFICATE-DINIC;
- (2) BEGIN
- (3)      $f := 0$ ;  $\bar{d}(t) = 0$ ;
- (4)     se determină rețeaua reziduală  $\tilde{G}(f)$ ;
- (5)     WHILE  $\bar{d}(t) < n$  DO
- (6)         BEGIN
- (7)             REȚEA-STRATIFICATĂ  $(\tilde{G}(f), \tilde{G}'(f), \bar{d}(t))$ ;
- (8)             FLUX-BLOCARE  $(\tilde{G}'(f), \bar{f})$ ;
- (9)             MĂRIRE FLUX  $(\tilde{G}(f), f, \bar{f})$ ;
- (10)        END;
- (11)      END.
  
- (1) PROCEDURA REȚEA-STRATIFICATĂ  $(\tilde{G}(f), \tilde{G}'(f), \bar{d}(t))$ ;
- (2) BEGIN
- (3)     se determină etichetele distanță complementară  $\bar{d}(x)$  în  $\tilde{G}(f)$ ;
- (4)     se construiește rețeaua stratificată  $\tilde{G}'(f)$ ;
- (5) END;
  
- (1) PROCEDURA FLUX-BLOCARE  $(\tilde{G}'(f), \bar{f})$ ;
- (2) BEGIN
- (3)      $\bar{f} := 0$ ;

```

(4) WHILE există DMF în  $\tilde{G}'(f)$  DO
(5)   BEGIN
(6)     se determină un DMF  $\tilde{D}$  în  $\tilde{G}'(f)$ ;
(7)     se determină  $r(\tilde{D}) := \min\{\tilde{r}'(x, y) \mid (x, y) \in \tilde{D}\}$ ;
(8)     se execută mărirea de flux;
(9)     se elimină arcele  $(x, y)$  cu  $\tilde{r}'(x, y) = 0$  și nodurile  $x \neq s, t$  cu
           $\rho^-(x) = 0$  sau  $\rho^+(x) = 0$  și arcele incidente cu aceste noduri;
(10)    END;
(11)  END;

(1) PROCEDURA MĂRIRE-FLUX ( $\tilde{G}(f), f, \bar{f}$ );
(2) BEGIN
(3)    $f := f + \bar{f}$ ;
(4)   se actualizează rețeaua reziduală  $\tilde{G}(f)$ ;
(5) END;

```

**Teorema 2.18. (Teorema de corectitudine a algoritmului)** *Algoritmul Dinic al rețelelor stratificate determină un flux maxim în rețeaua  $G = (N, A, c)$ .*

**Demonstrație.** Algoritmul rețelelor stratificate Dinic se termină când  $\bar{d}(t) \geq n$ . După un număr de iterații ale algoritmului condiția  $\bar{d}(t) \geq n$  va fi îndeplinită deoarece din definiția și proprietățile funcției  $\bar{d}$  rezultă că etichetele  $\bar{d}(t)$ , calculate pentru rețelele stratificate construite în algoritm, formează un sir strict crescător. Dacă  $\bar{d}(t) \geq n$ , atunci conform Proprietății 2.16, rețeaua reziduală  $\tilde{G}(f)$  nu conține DMF de la nodul sursă  $s$  la nodul stoc  $t$ . Deci, la terminare, algoritmul rețelelor stratificate Dinic determină un flux maxim. ■

**Teorema 2.19. (Teorema de complexitate a algoritmului).** *Algoritmul rețelelor stratificate Dinic are complexitatea  $O(mn^2)$ .*

**Demonstrație.** Procedura REȚEA-STRATIFICATĂ și procedura MĂRIRE-FLUX au fiecare complexitatea  $O(m)$ . Determinarea unui DMF  $D$  în  $\tilde{G}'$  cu parcurgerea DF are complexitatea  $O(n)$ . Ciclul WHILE din procedura FLUX-BLOCARE se execută de  $O(m)$  ori deoarece la fiecare execuție se elimină cel puțin un arc din  $\tilde{G}'$  și 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 rețelelor stratificate Dinic are complexitatea  $O(mn)$ . Acest ciclu se execută de  $O(n)$  ori. Prin urmare algoritmul are complexitatea  $O(mn^2)$ . ■

### 2.1.7 Algoritmul Ahuja - Orlin al rețelelor stratificate

Algoritmul Ahuja - Orlin al rețelelor stratificate se datorează lui Ahuja și Orlin (1991) și este o versiune a algoritmului rețelelor stratificate Dinic. În acest caz rețelele stratificate sunt definite cu ajutorul etichetelor distanță exacte  $d(x)$ .

În raport cu un flux dat  $f$ , rețeaua stratificată  $\tilde{G}'(f)$  se definește în modul următor. Se determină etichetele distanță exacte  $d(x)$  prin parcurgerea inversă BF a rețelei rezid-

uale  $\tilde{G}(f)$  pornind de la nodul stoc  $t$ . Rețeaua stratificată este  $\tilde{G}'(f) = (\tilde{N}', \tilde{A}', r')$  cu  $\tilde{N}' = \tilde{N}$ ,  $\tilde{A}' = \{(x, y) \mid (x, y) \in \tilde{A}, d(x) = d(y) + 1\}$ ,  $r'(x, y) = r(x, y)$ ,  $(x, y) \in \tilde{A}'$ . Pentru fiecare  $k$ ,  $k = 0, \dots, d(s)$ , se definește un strat  $\tilde{N}'(k) = \{x \mid x \in \tilde{N}, d(x) = k\}$ . Este evident că  $\tilde{N}'(0) = \{t\}$  și  $\tilde{N}' = \cup \{\tilde{N}'(k) \mid k = 0, \dots, d(s)\}$ . Legătura dintre straturile  $\overline{\tilde{N}'}(\bar{k})$  și straturile  $\tilde{N}'(k)$  este  $\overline{\tilde{N}'}(\bar{d}(t) - k) = \tilde{N}'(k)$  sau  $\overline{\tilde{N}'}(\bar{k}) = \tilde{N}'(d(s) - \bar{k})$ .

**Exemplul 2.9.** Se consideră rețeaua reziduală  $\tilde{G}(f)$  prezentată în figura 2.9. În figura

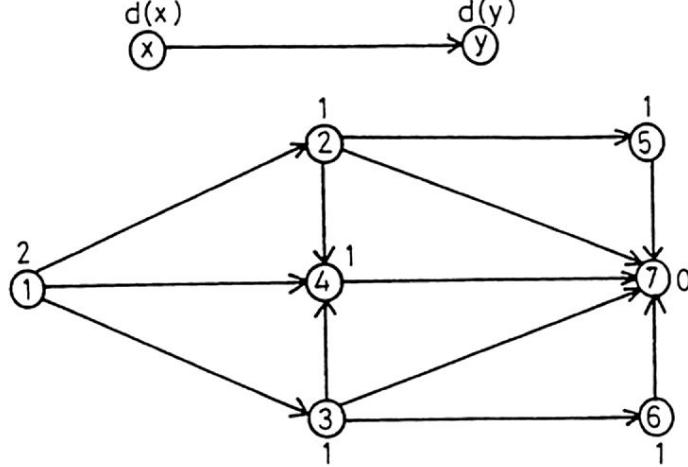


Fig.2.9

2.10 (a) se prezintă rețeaua stratificată  $\tilde{G}'(f)$  asociată rețelei reziduale  $\tilde{G}(f)$  prezentată în figura 2.9. Deoarece arcele  $(5,7)$ ,  $(6,7)$  și nodurile  $5,6$  nu aparțin la nici un drum de la nodul  $s = 1$  la nodul  $t = 7$  se elimină și se obține rețeaua stratificată prezentată în figura 2.10(b).

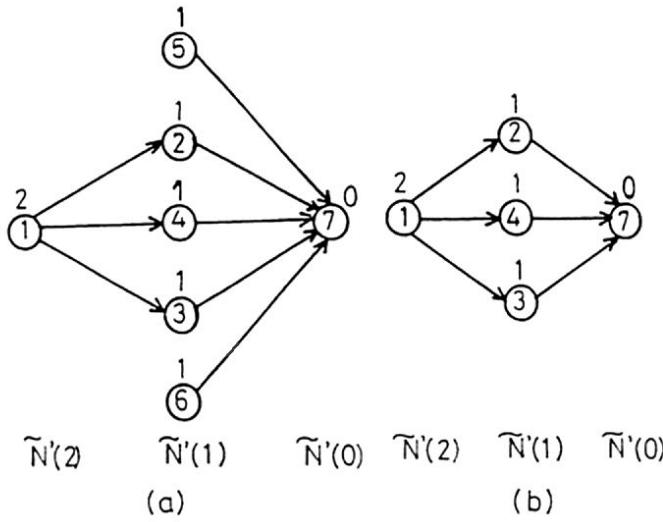


Fig.2.10

Evident că un arc  $(x, y) \in \tilde{A}'$  cu  $d(x) = k$  are proprietatea că  $x \in \tilde{N}'(k)$ ,  $y \in \tilde{N}'(k - 1)$ .

Algoritmul Ahuja - Orlin al rețelelor stratificate (în continuare algoritmul A - O al rețelelor stratificate) se obține făcând în algoritmul Ahuja - Orlin al drumului celui mai scurt (algoritmul AODS) următoarele modificări:

**Modificarea 1.** În operația înapoiere ( $x$ ), nu schimbăm eticheta distanță 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  $d(x) = d(y) + 1$ ,  $r(x, y) > 0$  și nodul  $y$  nu este blocat.

**Modificarea 3.** Când nodul sursă  $s$  devine blocat, se execută parcurgere înapoi BF pentru a recalcule etichetele distanță exacte care definesc o nouă rețea stratificată.

```

(1) PROGRAM A - O REȚELE STRATIFICATE;
(2) BEGIN
(3)    $f := 0$ ;
(4)   se determină etichetele distanță exacte  $d(x)$  în  $\tilde{G}(f)$ ;
(5)   FOR  $x \in \tilde{N}$  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 distanță exacte  $d(x)$  în  $\tilde{G}(f)$ ;
(23)        FOR  $x \in \tilde{N}$  DO  $B(x) := 1$ ;
(24)         $x := s$ ;
(25)      END;
(26)    END;
(27)  END.

(1) PROCEDURA INAINTARE( $x$ );
(2) BEGIN
(3)    $\tilde{p}(y) := x$ ;
(4)    $x := y$ ;
```

```

(5)   END;
(1)  PROCEDURA INAPOIERE( $x$ );
(2)  BEGIN
(3)     $B(x) := 0$ ;
(4)    IF  $x \neq s$ 
(5)      THEN  $x := \tilde{p}(x)$ ;
(6)  END;

(1)  PROCEDURA MĂRIRE;
(2)  BEGIN
(3)    se determină DMF  $\tilde{D}$  utilizând vectorul predecesor  $\tilde{p}$ ;
(4)    se determină  $r(\tilde{D}) = \min\{r(x, y) \mid (x, y) \in \tilde{D}\}$ ;
(5)    se execută mărirea de flux;
(6)    se actualizează rețeaua reziduală  $\tilde{G}(f)$ ;
(7)  END;

```

Algoritmul menține un drum parțial admisibil în rețeaua stratificată  $\tilde{G}'(f)$  și iterativ execută operații de înaintare sau înapoiere de la nodul curent  $x$ . Dacă nodul curent  $x$  este extremitatea inițială a unui arc admisibil  $(x, y)$  se execută o operație de înaintare și se adaugă arcul  $(x, y)$  la drumul parțial admisibil, altfel se execută o operație de înapoiere și se blochează nodul  $x$  prin  $B(x) := 0$ . Repetăm aceste operații până când drumul parțial admisibil atinge nodul stoc  $t$  la care timp executăm o mărire de flux. Repetăm acest proces până când se obține un flux de blocare în rețeaua stratificată  $\tilde{G}'(f)$ , adică până când nodul sursă  $s$  devine blocat. În acest caz se recalculează etichetele distanță  $d$  care definesc o nouă rețea stratificată și 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 2.10** Ilustrăm algoritmul A - O al rețelelor stratificate pe exemplul dat în figura 2.11 (a).

Inițial  $d := (3, 2, 2, 1, 1, 0)$  și  $B := (1, 1, 1, 1, 1, 1)$ . Algoritmul determină DMF:  $\tilde{D}_1 = (1, 2, 4, 6)$  cu  $r(\tilde{D}_1) = 2$ ,  $\tilde{D}_2 = (1, 2, 5, 6)$  cu  $r(\tilde{D}_2) = 1$ ,  $\tilde{D}_3 = (1, 3, 5, 6)$  cu  $r(\tilde{D}_3) = 2$ . Mărind fluxul de-a lungul acestor drumuri se obține rețeaua reziduală din figura 2.11 (b). În continuare algoritmul determină drumul parțial admisibil  $\tilde{D}_4 = (1, 3, 5)$ . Nodul 5 nu este extremitatea inițială a unui arc admisibil și se execută o operație de înapoiere blocându-se nodul 5, deci  $B(5) := 0$ . Analog  $\tilde{D}_4 = (1, 3)$ ,  $B(3) := 0$ ,  $\tilde{D}_4 = (1)$ ,  $B(1) := 0$ . Nodul sursă 1 devine nod blocat. Deci s-a obținut un flux de blocare. Se calculează  $d := (4, 3, 3, 1, 2, 0)$  și  $B := (1, 1, 1, 1, 1, 1)$  și rețeaua reziduală cu noile etichete care definesc o nouă rețea stratificată este arătată în figura 2.11 (c). Algoritmul determină DMF  $\tilde{D}_5 = (1, 3, 5, 4, 6)$  cu  $r(\tilde{D}_5) = 1$ . Mărind fluxul pe acest drum se obține rețeaua reziduală din figura 2.11 (d). În continuare algoritmul determină drumul parțial admisibil  $\tilde{D}_6 = (1, 3)$ . Nodul 3 devine nod blocat deci  $B(3) := 0$  și  $\tilde{D}_6 = (1)$ ,  $B(1) := 0$ . Din nou s-a obținut un flux de blocare. Se calculează  $d := (\infty, \infty, \infty, 1, \infty, 0)$ ,  $B := (1, 1, 1, 1, 1, 1)$ . Rețeaua reziduală cu noile etichete este prezentată în figura 2.11 (e). Deoarece  $d(1) \geq n$  algoritmul se oprește. Fluxul maxim este prezentat în figura 2.11 (f).

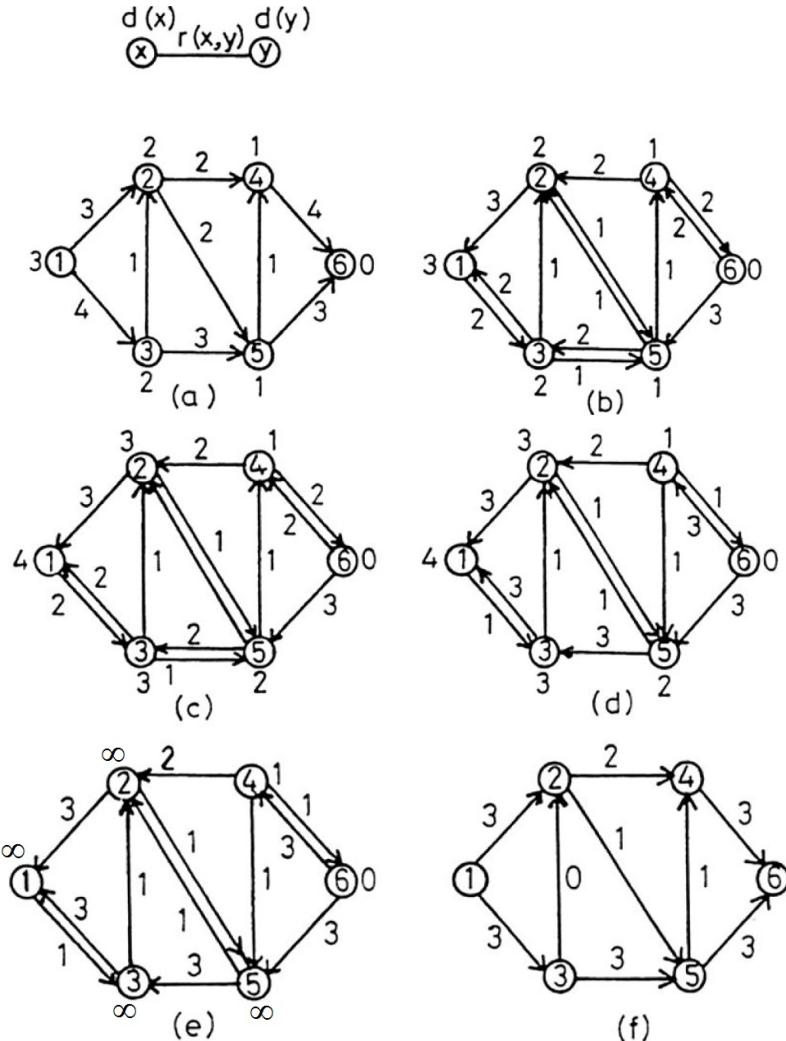


Fig.2.11

**Teorema 2.20. (Teorema de corectitudine a algoritmului)** *Algoritmul A - O al rețelelor stratificate determină un flux maxim în rețea G = (N, A, c).*

**Demonstrație.** Algoritmul rețelelor stratificate se termină când  $d(s) \geq n$ . Conform Proprietății 2.2 rezultă că rețea reziduală nu conține DMF de la nodul sursă  $s$  la nodul stoc  $t$ . Deci la terminare, algoritmul rețelelor stratificate determină un flux maxim. ■

În continuare se va analiza complexitatea algoritmului A - O al rețelelor stratificate.

**Lema 2.21.** *Algoritmul A - O al rețelelor stratificate calculează cel mult  $n$  fluxuri de blocare pentru determinarea unui flux maxim.*

**Demonstrație.** Când algoritmul calculează etichetele distanță  $d$ , mulțimea arcelor admisibile definește o rețea stratificată. Deoarece nu actualizează etichetele distanță în timpul calculării unui flux de blocare, fiecare drum admisibil din rețea stratificată are lungimea  $d(s)$ . Fluxul de blocare sumarizează toate fluxurile de-a lungul drumurilor admisibile de lungime  $d(s)$ . Deci când algoritmul recalculează etichetele distanță ex-

acte, care definesc o nouă rețea stratificată,  $d(s)$  trebuie să crească strict. Algoritmul se termină când  $d(s) \geq n$  ceea ce demonstrează afirmația lemei.■

**Teorema 2.22. (Teorema de complexitate a algoritmului).** *Algoritmul A - O al rețelelor stratificate are complexitatea  $O(mn^2)$ .*

**Demonstrație.** În timpul calculării unui flux de blocare nu se introduc noi arce admisibile deoarece în acest timp algoritmul nu actualizează etichetele distanță. Fiecare mărire de flux reduce capacitatea reziduală la zero a cel puțin unui arc. Deci algoritmul execută cel mult  $m$  măriri de flux pentru calculul unui flux de blocare. Din definiție rezultă că orice rețea stratificată este aciclică. Determinarea unui DMF în rețea stratificată are complexitatea  $O(n)$ . Rezultă că determinarea unui flux de blocare are complexitatea  $O(mn)$ . Înținând seama de Proprietatea 2.2 obținem că algoritmul are complexitatea  $O(mn^2)$ .■

## 2.2 Algoritmi polinomiali cu prefluxuri

### 2.2.1 Algoritmul preflux generic

Algoritmul preflux generic este obținut de Goldberg și Tarjan (1986) și noțiunea de preflux a fost introdusă de Karzanov (1974) pentru calcularea fluxurilor de blocare.

Un *preflux* este o funcție  $f : A \rightarrow \mathbb{R}^+$  care satisfac constrângările

$$f(N, x) - f(x, N) \geq 0, \quad x \in N - \{s, t\} \quad (2.9.a)$$

$$0 \leq f(x, y) \leq c(x, y), \quad (x, y) \in A. \quad (2.9.b)$$

Pentru un preflux dat  $f$  rețeaua reziduală  $\tilde{G}(f)$  și capacitatea reziduală  $r$  se definesc analog ca pentru fluxuri. *Excesul* fiecarui nod  $x \in N$  este

$$e(x) = f(N, x) - f(x, N) \quad (2.10)$$

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 numește *nod activ* dacă  $d(x) > 0$  și  $e(x) > 0$ . Deoarece întotdeauna  $e(s) \leq 0$  și  $d(t) = 0$  rezultă că nodurile sursă  $s$  și 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 trimite flux de la nodul sursă la nodul stoc de-a lungul unui drum de mărire a fluxului din rețeaua reziduală corespunzător unui lanț de mărire a fluxului din rețeaua originară. Această operație se descompune în mai multe operații elementare de trimitere a fluxului de-a lungul arcelor ce compun drumul. Fiecare operație elementară de trimitere a fluxului de-a lungul unui arc se va numi *înaintare*. Algoritmii cu prefluxuri utilizează aceste operații elementare (înaintări) și deci trimit fluxul pe arcele individuale în locul drumurilor de mărire a fluxului.

Prezența nodurilor active indică faptul că soluția este inadmisibilă. De aceea, operația de bază a algoritmului preflux încărtare este selectarea unui nod activ și

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 distanță. De aceea, trimitem flux pe arcele admisibile. Dacă nodul activ curent nu este extremitatea inițială a unui arc admisibil, îi creștem eticheta distanță astfel încât să creem cel puțin un arc admisibil. Algoritmul se termină când rețeaua nu conține nici un nod activ.

```

(1) PROGRAM PREFLUX GENERIC;
(2) BEGIN
(3)   INITIALIZARE;
(4)   WHILE rețeaua reziduală conține un nod activ DO
(5)     BEGIN
(6)       se selectează un nod activ  $x$ ;
(7)       INAINTARE/REETICHETARE( $x$ );
(8)     END;
(9)   END.

(1) PROCEDURA INITIALIZARE;
(2) BEGIN
(3)    $f := 0$ ;
(4)   se calculează etichetele distanță exacte  $d(x)$  în  $\tilde{G}(f)$ ;
(5)   se înaintează  $u := r(s, y)$  unități de flux pe fiecare arc  $(s, y) \in E^+(s)$ ;
(6)    $d(s) := n$ ;
(7) END;

(1) PROCEDURA INAINTARE/REETICHETARE( $x$ );
(2) BEGIN
(3)   IF rețeaua reziduală conține arc admisibil  $(x, y)$ 
(4)     THEN
(5)       se înaintează  $u := \min\{e(x), r(x, y)\}$  unități de flux de la  $x$  la  $y$ 
(6)     ELSE
(7)        $d(x) := \min\{d(y) + 1 \mid (x, y) \in E^+(x) \text{ și } r(x, y) > 0\}$ ;
(8)   END;

```

O înaintare de  $u$  unități de flux de la nodul  $x$  la nodul  $y$  constă din următoarele operații 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 numește *saturată*, iar dacă  $u = e(x)$  atunci înaintarea pe arcul  $(x, y)$  se numește *nesaturată*. O înaintare saturată descrește  $r(x, y)$  la zero, iar o înaintare nesaturată descrește  $e(x)$  la zero. Operația  $d(x) := \min\{d(y) + 1 \mid (x, y) \in E^+(x) \text{ și } r(x, y) > 0\}$  se numește *operatie de reetichetare*. Scopul operației de reetichetare este de a crea cel puțin un arc admisibil pe care algoritmul poate executa înaintări ulterioare.

Procedura INITIALIZARE realizează următoarele:

- i) fiecare nod  $y$  pentru care  $(s, y) \in E^+(s)$  primește un exces pozitiv și 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 și stabilind  $d(s) = n$  rămâne satisfăcută condiția de validitate a etichetelor;

iii) stabilind  $d(s) = n$ , rețeaua reziduală nu va conține drumuri de la nodul sursă  $s$  la nodul stoc  $t$ ; deoarece etichetele distanță sunt nedescrescătoare, garantăm că în iterațiile ulterioare ale algoritmului rețeaua reziduală nu va conține drum de la  $s$  la  $t$  niciodată și astfel niciodată nu vom mai înainta flux de la nodul  $s$ .

**Exemplul 2.11.** Se consideră rețeaua dată în figura 2.12 (a). Figura 2.12 (b) specifică prefluxul după inițializări.

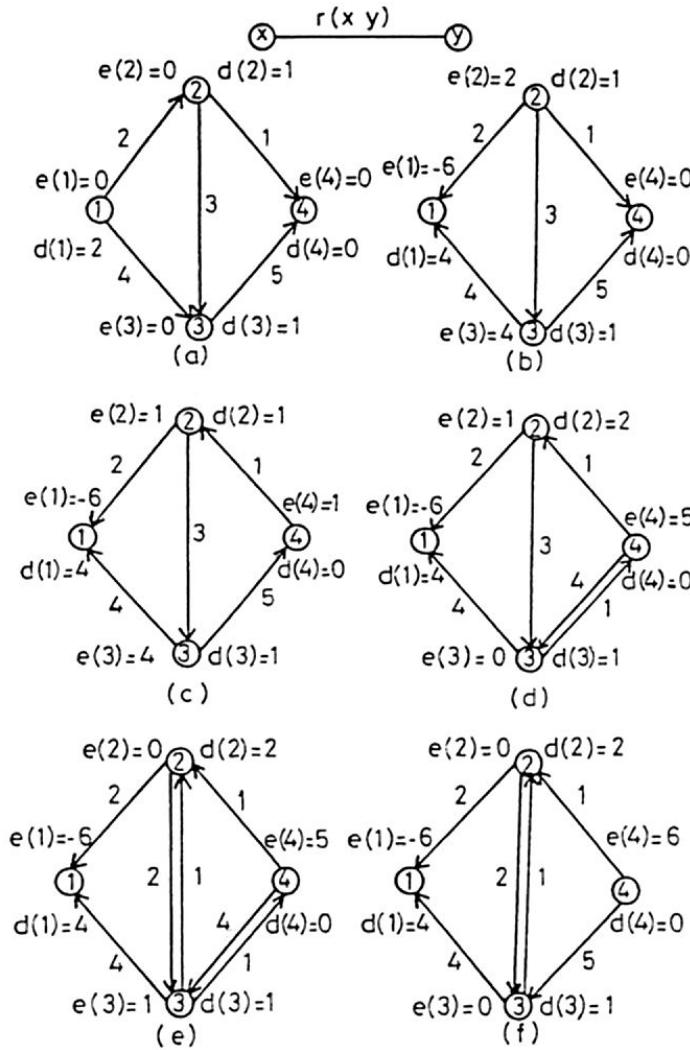


Fig.2.12

**Iterația 1.** Presupunem că algoritmul selectează nodul 2 pentru operația de înaintare/re-etichetare. Arcul  $(2, 4)$  este singurul arc admisibil și algoritmul execută o înaintare de valoare  $u := \min\{2, 1\} = 1$ . Această înaintare este saturată. Figura 2.12 (c) prezintă rețeaua reziduală după această iterație.

**Iterația 2.** Presupunem că algoritmul selectează din nou nodul 2. Deoarece nici un arc admisibil nu pleacă din nodul 2, algoritmul execută o operație de reetichetare și

$d(2) := \min\{d(3) + 1, d(1) + 1\} = \min\{2, 5\} = 2$ . Noua rețea reziduală este aceeași ca cea arătată în figura 2.12 (c) cu excepția că  $d(2) = 2$ .

**Iterația 3.** Presupunem că algoritmul selectează nodul 3. Arcul  $(3, 4)$  este singurul arc admisibil care pleacă din nodul 3 și algoritmul execută o înaintare cu  $u = \min\{e(3), r(3, 4)\} = \min\{4, 5\} = 4$ . Această înaintare este nesaturată. Figura 2.12 (d) prezintă rețeaua reziduală după această iterare.

**Iterația 4.** Algoritmul selectează nodul 2 și execută o înaintare nesaturată pe arcul  $(2, 3)$  cu  $u = \min\{1, 3\} = 1$ , obținându-se rețeaua reziduală dată în figura 2.12 (e).

**Iterația 5.** Algoritmul selectează nodul 3 și execută o înaintare saturată pe arcul  $(3, 4)$  cu  $u = \min\{1, 1\} = 1$ , obținându-se rețeaua reziduală prezentată în figura 2.12 (f).

Acum rețeaua nu conține noduri active și algoritmul se termină. Valoarea fluxului maxim este  $v^* = e(4) = 6$ .

**Teorema 2.23. (Teorema de corectitudine a algoritmului)** *Algoritmul preflux generic determină un flux maxim în rețeaua  $G = (N, A, c)$ .*

**Demonstrație.** Algoritmul se termină când  $e(x) = 0$ ,  $x \in N - \{s, t\}$  ceea ce implică faptul că  $f$  este un flux. Deoarece  $d(s) \geq n$ , rețeaua reziduală nu conține drum de la nodul sursă  $s$  la nodul stoc  $t$ . Deci fluxul curent este un flux maxim. ■

Pentru analiza complexității algoritmului preflux generic remarcăm faptul că rămâne valabilă Lema 2.9, deoarece, ca în cazul algoritmului AODS, algoritmul preflux generic înaintează fluxul numai pe arcele admisibile și reetichetează un nod  $x$  numai când nu există arc admisibil care îl are pe  $x$  ca extremitate inițială.

**Lema 2.24.** *La orice iterare 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 rețeaua reziduală.*

**Demonstrație.** 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 rețeaua 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  $x$  și (3) circuitelor. Fie  $x$  un nod activ relativ la prefluxul  $f$  în  $G$ . Descompunerea flux a lui  $f$  trebuie să conțină un drum  $D$  de la nodul  $s$  la nodul  $x$ , deoarece fluxurile de-a lungul drumurilor de la nodul  $s$  la nodul  $t$  și fluxurile în jurul circuitelor nu contribuie la excesul nodului  $x$ . Rețeaua reziduală conține inversul lui  $D$  ( $D$  cu orientarea fiecărui arc inversă), adică un drum de la nodul  $x$  la nodul  $s$ . ■

Această lemă ne asigură că operația de reetichetare calculează minimul elementelor unei mulțimi diferită de mulțimea vidă.

**Lema 2.25.** *Pentru fiecare nod  $x \in N$ ,  $d(x) < 2n$ .*

**Demonstrație.** Când algoritmul reetichetează ultima dată nodul  $x$ , acest nod a avut un exces pozitiv. Deci rețeaua reziduală conține un drum  $D$  de lungime cel mult  $n - 2$  de la nodul  $x$  la nodul  $s$ . Fie  $D = (x_1, x_2, \dots, x_k)$ ,  $k \leq n - 1$ ,  $x_1 = x$ ,  $x_k = s$ . Avem

$d(x_1) \leq d(x_2) + 1$ ,  $d(x_2) \leq d(x_3) + 1, \dots, d(x_{k-1}) \leq d(x_k) + 1$  sau  $d(x_1) \leq d(x_3) + 2 \leq \dots \leq d(x_k) + k - 1$ . Înținând seama că  $x_1 = x$ ,  $x_k = s$ ,  $k \leq n - 1$ ,  $d(s) = n$  avem  $d(x) < 2n$ . ■

**Lema 2.26.** *Fiecare etichetă distanță crește de cel mult  $2n$  ori și numărul total al operațiilor de reetichetare este cel mult  $2n^2$ .*

**Demonstrație.** Fiecare operație de reetichetare a nodului  $x$  crește eticheta distanță

$d(x)$  cu cel puțin o unitate și conform Lemei 2.25 fiecare etichetă distanță  $d(x)$  va crește de cel mult  $2n$  ori. În consecință numărul total al operațiilor de reetichetare este cel mult  $2n^2$ . ■

**Lema 2.27.** *Algoritmul execută cel mult  $mn$  înaintări saturate.*

**Demonstrație.** Conform Lemei 2.12 dacă algoritmul reetichetează oricărui nod de cel mult  $k$  ori, algoritmul saturează arcele de cel mult  $km/2$  ori. Din Lema 2.25 rezultă că algoritmul execută cel mult  $mn$  înaintări saturate. ■

**Lema 2.28.** *Timpul necesar pentru determinarea arcelor admisibile și reetichetarea nodurilor este  $O(mn)$ .*

**Demonstrație.** Conform Lemei 2.11 dacă algoritmul reetichetează orice nod de cel mult  $k$  ori, timpul necesar pentru determinarea arcelor admisibile și reetichetarea nodurilor este  $O(km)$ . Înținând seama de Lema 2.25 rezultă că timpul necesar pentru determinarea arcelor admisibile și reetichetarea nodurilor este  $O(mn)$ . ■

**Lema 2.29.** *Algoritmul preflux generic execută  $O(mn^2)$  înaintări nesaturate.*

**Demonstrație.** Pentru demonstrație utilizăm o funcție potențial. Notăm cu  $N^+$  mulțimea nodurilor active. Considerăm funcția potențial  $P = \sum_{N^+} d(x)$ . Deoarece  $|N^+| < n$  și  $d(x) < 2n$ ,  $x \in N^+$  valoarea inițială a lui  $P$  după inițializare este cel mult  $2n^2$ . La terminarea algoritmului  $N^+ = \emptyset$  și  $P$  are valoarea zero. În timpul execuției procedurii ÎNAINTARE/REETICHETARE( $x$ ), unul din următoarele două cazuri trebuie să apară.

**Cazul 1.** Rețeaua reziduală nu conține nici un arc admisibil  $(x, y)$ . În acest caz  $d(x)$  crește cu  $q \geq 1$  unități. Funcția potențial  $P$  crește cu cel mult  $q$  unități. Deoarece  $d(x) < 2n$ ,  $x \in N$ , creșterea totală în  $P$  datorată creșterii etichetelor este limitată de  $2n^2$ .

**Cazul 2.** Rețeaua reziduală conține 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  $y$  și deci crește numărul nodurilor active cu 1. Rezultă că funcția potențial  $P$  crește cu  $d(y) < 2n$  per o înaintare saturată și cu cel mult  $2mn^2$  pentru toate înaintările saturate. O înaintare nesaturată pe arcul  $(x, y)$  nu crește numărul de noduri active. Înaintarea nesaturată produce  $e(x) = 0$ , deci nodul  $x$  devine inactiv, dar nodul  $y$  poate să devină activ. Deci funcția potențial  $P$  descrește prin  $d(x)$  și poate să crească prin  $d(y) = d(x) - 1$ . În consecință, funcția potențial  $P$  descrește cu cel puțin 1 unitate per o înaintare nesaturată.

În concluzie valoarea inițială a lui  $P$  este cel mult  $2n^2$  și creșterea maximă posibilă a lui  $P$  este  $2n^2 + 2mn^2$ . Fiecare înaintare nesaturată descrește  $P$  prin cel puțin o unitate și  $P \geq 0$ . Deci algoritmul poate executa cel mult  $2n^2 + 2n^2 + 2n^2m$ , adică  $O(mn^2)$  înaintări nesaturate. ■

**Teorema 2.30. (Teorema de complexitate a algoritmului)** *Algoritmul preflux generic are complexitatea  $O(mn^2)$ .*

**Demonstrație.** Algoritmul se termină când mulțimea nodurilor active  $N^+ = \emptyset$ . Pentru  $N^+ = \emptyset$ , obținem  $P = 0$ . Înținând cont de Lema 2.29 rezultă afirmația teoremei. ■

Fie  $v^*$  valoarea fluxului maxim  $f^*$ . Un preflux  $f$  cu  $e(t) = v^*$  se numește *preflux maxim*. Algoritmul preflux generic stabilește un preflux maxim cu mult înainte ca el să stabilească un flux maxim ( $e(x) = 0$ ,  $x \in N - \{s, t\}$ ). Operațiile înaintare/reetichetare ulterioare cresc etichetele distanță ale nodurilor active până când ele sunt mai mari

decat  $n$ , astfel ele pot sa trimita excesul lor inapoi la nodul sursa a carui eticheta distanta este  $n$ . O posibila modificare a algoritmului consta in introducerea multimii  $N'$  a nodurilor cu proprietatea ca reteaua reziduala nu contine drum de la orice nod din  $N'$  la nodul stoc  $t$ .

Initial  $N' = \{s\}$  si definim tabloul  $q$  ca la algoritmul AODS. Daca  $q(k') = 0$  atunci pentru orice nod  $x$  cu  $d(x) > k'$  efectuam  $d(x) = n$  si nodul  $x$  il introducem in  $N'$ . Nu efectuam operatiile de inaintare/ reetichetare pentru nodurile din  $N'$  si algoritmul se termina cand  $e(x) = 0$ ,  $x \in \bar{N}' = N - N' - \{t\}$ . La terminare, prefluxul curent  $f$  este un preflux maxim si-l convertim intr-un flux maxim.

### 2.2.2 Algoritmul preflux FIFO

Algoritmul preflux FIFO este obtinut de Goldberg(1985). Mai intai definim conceptul de analizare a unui nod.

Algoritmul preflux generic, intr-o iteratie, selecteaza un nod activ  $x$  si efectueaza fie o inaintare saturata, fie o inaintare nesaturata, fie o reetichetare a nodului  $x$ . Daca se efectueaza o inaintare saturata atunci nodul  $x$  poate sa ramana nod activ. Algoritmul preflux generic poate selecta, in iteratia urmatoare, alt nod activ  $y \neq x$ . Impunem regula ca algoritmul selecteaza un nod  $x$  consecutiv pana cand  $e(x) = 0$  sau nodul  $x$  este reetichetat. In consecinta, algoritmul preflux generic poate executa de la un nod  $x$  mai multe inaintari nesaturate fie de o inaintare nesaturata ( $e(x)$  devine zero), fie de o operatie de reetichetare (nu exista arc admisibil  $(x, y)$ ). Numim aceasta secventa de operatii *analizarea nodului x*. In continuare presupunem ca toți algoritmii cu preflux adoptă această regulă de analizare a nodurilor.

Notam  $L$  lista nodurilor active. Algoritmul preflux FIFO analizeaza nodurile active in ordinea primul intrat, primul ieșit (FIFO). Lista nodurilor active  $L$  este organizata, sub aspectul unei structuri de date, ca o coadă. Deci algoritmul preflux FIFO selecteaza nodul  $x$  din capul listei  $L$ , execută o inaintare de la acest nod și adaugă noul nod activ  $y$  la urma listei  $L$ . Algoritmul selecteaza nodul  $x$  pana cand fie devine inactiv ( $e(x) = 0$ ), fie este reetichetat. In ultimul caz, se adaugă nodul  $x$  la urma listei  $L$ . Algoritmul preflux FIFO se termină cand coada nodurilor active  $L$  este vidă.

```

(1)  PROGRAM PREFLUX-FIFO;
(2)  BEGIN
(3)    INITIALIZARE;
(4)    WHILE L ≠ ∅ DO
(5)      BEGIN
(6)        se scoate primul nod x din L;
(7)        INAINTARE/REETICHETARE(x);
(8)      END;
(9)    END.

(1)  PROCEDURA INITIALIZARE;
(2)  BEGIN
(3)    f := 0; L := ∅;

```

```

(4)      se calculează etichetele distanță exacte  $d(x)$ ;
(5)      FOR  $(s, y) \in E^+(s)$  DO
(6)      BEGIN
(7)          se înaintează  $r(s, y)$  unități de flux pe arcul  $(s, y)$ ;
(8)          IF  $e(y) > 0$  și  $y \neq t$ 
(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  $u := \min\{e(x), r(x, y)\}$  unități de
                     flux de la  $x$  la  $y$ ;
(9)                 IF  $y \notin L$  și  $y \neq s, t$ 
(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)                        $d(x) := \min\{d(y) + 1 \mid (x, y) \in E^+(x) \text{ și}$ 
                            $r(x, y) > 0\}$ ;
(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 2.12.** Ilustrăm algoritmul preflux FIFO utilizând exemplul arătat în figura 2.13.

Se consideră rețeaua din figura 2.13 (a). După execuția procedurii INITIALIZARE se obține rețeaua prezentată în figura 2.13 (b). Algoritmul scoate nodul 2 din coada  $L$  și-l analizează. Presupunem că el execută o înaintare saturată de 5 unități pe arcul  $(2, 4)$  și o înaintare nesaturată de 5 unități pe arcul  $(2, 5)$ . După analizarea nodului 2 se obține rețeaua prezentată în figura 2.13 (c). Algoritmul în continuare scoate nodul 3 din coada  $L$  și-l analizează. Algoritmul execută o înaintare saturată de 5 unități pe arcul  $(3, 5)$ , urmată de o operație de reetichetare a nodului 3. După analizarea nodului 3 se obține rețeaua prezentată în figura 2.13 (d). În final se obțin rețelele din figura

2.13 (e) și (f).

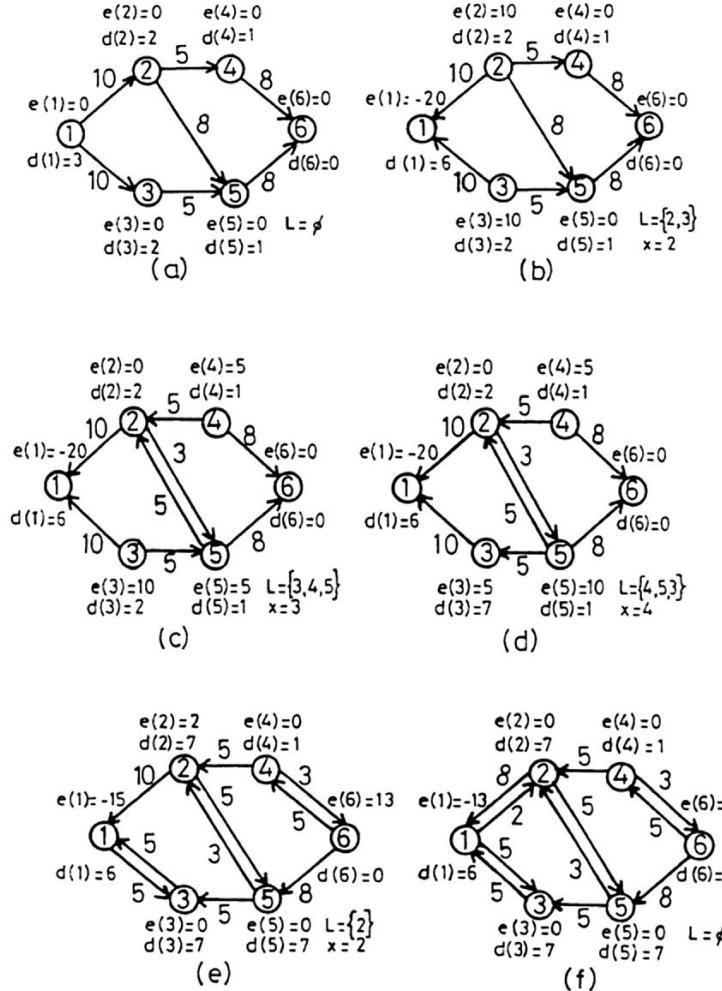


Fig.2.13

Corectitudinea algoritmului preflux FIFO rezultă din corectitudinea algoritmului preflux generic.

Pentru a analiza complexitatea algoritmului preflux FIFO, partitioñăm numărul total de examinări nod în faze. Prima fază constă din analizările nod pentru nodurile din coada  $L$  obținută imediat după execuþia procedurii INITIALIZARE. 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 și aşa mai departe. De exemplu, în ilustrarea precedentă, prima fază constă din analizările nod ale listei  $L = \{2, 3\}$  și 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 2.31. (Teorema de complexitate a algoritmului.)** *Algoritmul preflux FIFO are complexitatea  $O(n^3)$ .*

**Demonstrație.** Considerăm funcția potențial  $P = \max\{d(x) \mid x \in L\}$  și schimbarea totală a lui  $P$  în timpul unei faze, adică diferența dintre valorile inițială și finală ale funcției potențial în timpul fazei. Considerăm două cazuri.

**Cazul 1.** Algoritmul execută cel puțin o operație de reetichetare în timpul unei faze. Atunci  $P$  poate să crească cu cel mult creșterea maximă a etichetelor distanță. Conform Lemei 2.25 rezultă faptul că  $P$  poate să crească de cel mult  $2n^2$  ori.

**Cazul 2.** Algoritmul nu execută nici o operație 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 consecință  $P$  descrește prin cel puțin o unitate.

Inițial  $P < n$  și combinând cazurile 1 și 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 și 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  $O(n^3)$ . ■

### 2.2.3 Algoritmul preflux cu eticheta cea mai mare

Algoritmul preflux cu eticheta cea mai mare este obținut de Goldberg și Tarjan (1986) și întotdeauna înaintează fluxul de la un nod activ  $x$  cu eticheta distanță cea mai mare. Fie  $p = \max\{d(x) \mid x \text{ este nod activ}\}$ . Algoritmul analizează mai întâi nodurile cu etichetele distanță egale cu  $p$  și înaintează fluxul la nodurile cu etichetele distanță egale cu  $p - 1$ , și aceste noduri, la rândul lor, înaintează fluxul la nodurile cu etichetele distanță egale cu  $p - 2$ , și aşa mai departe, până când fie algoritmul reetichetează un nod, fie a epuizat toate nodurile active. Când a fost reetichetat un nod, algoritmul repetă același proces. Corectitudinea algoritmului preflux cu eticheta cea mai mare rezultă din corectitudinea algoritmului preflux generic.

Pentru selectarea unui nod cu eticheta distanță cea mai mare utilizăm următoarea structură de date:

$$L(i) := \{x \mid x \text{ este nod activ și } d(x) = i\}$$

sub forma unei stive înlănțuite sau a unei cozi înlănțuite. Deoarece conform Lemei 2.25  $d(x) < 2n$ ,  $x \in N$ , rezultă faptul că variabila  $i$  trebuie să aibă valorile  $i = 1, 2, \dots, 2n - 1$ . 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  $h = k$  și selectăm orice nod din  $L(k)$ . Dacă eticheta distanță a unui nod  $x$  crește de la  $d(x)$  la  $d'(x)$  în timpul analizării stabilim  $h = d'(x)$ . Conform Lemei 2.24 variabila nivel  $h$  crește de cel mult  $2n^2$  ori. Deoarece inițial  $h < n$  rezultă că  $h$  descrește de cel mult  $2n^2 + n$  ori.

Algoritmul preflux cu eticheta cea mai mare este următorul:

- (1) PROGRAM PREFLUX-ETICH;
- (2) BEGIN
- (3)     INITIALIZARE;
- (4)     WHILE  $L \neq \emptyset$  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 INITIALIZARE;
(2) BEGIN
(3)    $f := 0$ ;
(4)    $L := \emptyset$ ;
(5)   se calculează etichetele exacte  $d(\cdot)$ ;
(6)   FOR  $(s, y) \in E^+(s)$  DO
(7)     BEGIN
(8)       se înaintează  $r(s, y)$  unități de flux pe arcul  $(s, y)$ ;
(9)       IF  $e(y) > 0$  și  $y \neq t$ 
(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  $\tilde{E}^+(x)$  este arcul curent;
(4)    $B := 1$ ;
(5)   REPEAT
(6)     fie  $(x, y)$  arcul curent din  $\tilde{E}^+(x)$ ;
(7)     IF  $(x, y)$  este arc admisibil
(8)       THEN BEGIN
(9)         se înaintează  $u := \min\{e(x), r(x, y)\}$  unități de flux de la  $x$  la  $y$ ;
(10)        IF  $y \notin L$  și  $y \neq s, t$ 
(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  $\tilde{E}^+(x)$ 
(15)            THEN se consideră că următorul arc din  $\tilde{E}^+(x)$  este arcul
(16)              curent;
(17)            ELSE BEGIN
(18)               $d(x) := \min\{d(y) + 1 \mid (x, y) \in \tilde{E}^+(x)\}$ ;
(19)               $B := 0$ ;
(20)            END;
(21)            IF  $e(x) > 0$ 
(22)              THEN se adaugă  $x$  în coada  $L$  cu prioritatea  $d(x)$ ;
(23) END;

```

**Teorema 2.32.** (Prima teoremă de complexitate a algoritmului.) *Algoritmul preflux cu eticheta cea mai mare are complexitatea  $O(n^3)$ .*

**Demonstrație.** Conform precizărilor de mai sus variabila nivel  $h$  descrește de cel mult  $2n^2 + n$  ori. Deci se obține 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 2.13.** Se consideră exemplul prezentat în figura 2.14. Rețeaua inițială este prezentată în figura 2.14 (a). După execuția procedurii INITIALIZARE nodurile  $2, 3, \dots, n - 1$  au un exces de 1 unitate fiecare. Algoritmul preflux cu eticheta cea mai mare analizează nodurile  $2, 3, \dots, n - 1$  în această ordine și înaintează excesul la nodul stoc  $n$ . În contrast, algoritmul preflux FIFO poate executa mult mai multe înaintări. Să presupunem că după execuția procedurii INITIALIZARE coada nodurilor active este  $L = \{n - 1, n - 2, \dots, 3, 2\}$ . Soluția este descrisă în figura 2.14 (b). Algoritmul va

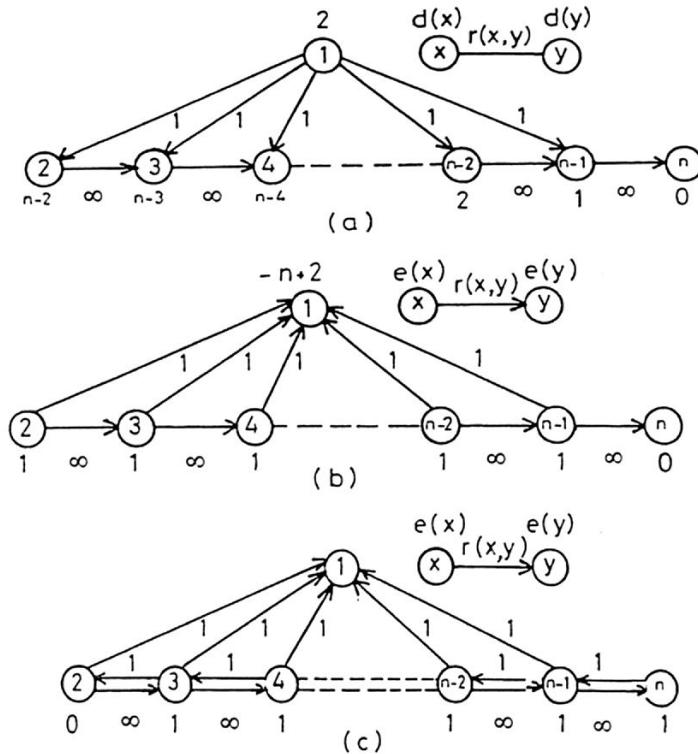


Fig.2.14

analiza fiecare din aceste noduri în prima fază și va obține soluția descrisă în figura 2.14 (c). După această fază  $L = \{n - 1, n - 2, \dots, 4, 3\}$ . Se poate arăta că algoritmul preflux FIFO va executa în total  $n - 2$  faze și utilizează  $(n - 2) + (n - 3) + \dots + 1 = \Omega(n^2)$  înaintări nesaturante.

Din exemplul prezentat rezultă că algoritmul preflux cu eticheta cea mai mare evită înaintări repetitive 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  $O(n^2m^{\frac{1}{2}})$  înaintări nesaturate. Astfel avem următoarea teoremă.

**Teorema 2.33. (A doua teoremă de complexitate a algoritmului.)** *Algoritmul preflux cu eticheta cea mai mare are complexitatea  $O(n^2m^{\frac{1}{2}})$ .*

**Demonstrație.** Pentru demonstrație se recomandă cititorului monografilele precizate la bibliografie. ■

Algoritmul preflux cu eticheta cea mai mare poate fi implementat ușor dacă în algoritmul FIFO, lista  $L$  a nodurilor active o structurăm nu ca o coadă obișnuită, ci ca o coadă prioritată cu prioritatea funcția distanță  $d$ .

#### 2.2.4 Algoritmul de scalare a excesului

Acest algoritm este o variantă îmbunătățită a algoritmului preflux generic și se datorează lui Ahuja și 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  $O(n^2 \log \bar{c})$  prin folosirea tehnicii scalării, adică prin impunerea condiției ca să se efectueze doar înaintări nesaturate de cantități suficient de mari de flux.

Fie  $\bar{r}$  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  $e(x) \geq \bar{r}/2$ ; în caz contrar nodul  $x$  are un *exces mic*.

Algoritmul de scalare a excesului încearcă să intotdeauna 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  $\bar{r}$ .

Algoritmul de scalare a excesului este următorul:

- (1) PROGRAM SCALARE EXCES;
- (2) BEGIN
- (3)     INITIALIZARE;
- (4)      $\bar{r} := 2^{\lceil \log \bar{c} \rceil}$ ;
- (5)     WHILE  $\bar{r} \geq 1$  DO
- (6)         BEGIN
- (7)             WHILE în rețeaua 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 distanță;
- (10)                     ÎNAINTARE/REETICHETARE( $x$ );
- (11)                 END;
- (12)              $\bar{r} := \bar{r}/2$ ;
- (13)         END;
- (14) END

PROCEDURA INITIALIZARE este aceeași ca la algoritmul preflux generic.

- (1) PROCEDURA ÎNAINTARE/REETICHETARE( $x$ );

```

(2) BEGIN
(3)   IF în rețeaua reziduală  $\tilde{G}(f)$  există un arc admisibil  $(x, y)$ 
(4)     THEN
(5)       se înaintează  $u := \min\{e(x), r(x, y), \bar{r} - e(y) > 0\}$  unități 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  $\bar{r}$  rămâne constant. O fază de scalare pentru o valoare dată a lui  $\bar{r}$  se numește *fază de  $\bar{r}$ -scalare*.

**Exemplul 2.14.** Se consideră rețeaua din figura 2.15(a). În figura 2.15(b) este reprezentată rețeaua reziduală obținută după executarea PROCEDURII INITIALIZARE. Se efectuează  $\bar{r} = 8$ . Singurul nod cu exces mare este nodul 3, care este selectat de algoritm și de la care se înaintează  $u := \min\{e(3), r(3, 4), \bar{r} - e(4)\} = \min\{4, 5, 8\} = 4$  unități de flux pe arcul admisibil  $(3, 4)$ . În rețeaua reziduală astfel obținută care este reprezentată în figura 2.15(c) nu mai există noduri cu excese mari. Se înjumătățește valoarea lui  $\bar{r}$  și se începe o nouă fază de scalare pentru  $\bar{r} = 4$ . Se selectează nodul 2 care este unicul nod cu exces mare și se înaintează o unitate de flux pe arcul admisibil  $(2, 4)$ , obținându-se rețeaua din figura 2.15(d). În această rețea nu mai există noduri cu exces mare, se efectuează  $\bar{r} := 2$  și se începe o nouă fază de scalare. Există un singur nod cu exces mare, nodul 2, care va fi selectat, reetichetat  $d(2) = 2$  și apoi selectat din nou. Se înaintează o unitate de flux pe arcul admisibil  $(2, 3)$  și se obține rețeaua reziduală din figura 2.15(e), în care există un singur nod cu exces mare: nodul 3. Acesta este selectat, se înaintează o unitate de flux pe arcul admisibil  $(3, 4)$  și se obține rețeaua reziduală din figura 2.15(f) în care nu mai există noduri cu exces mare. Se efectuează  $\bar{r} := 1$  și, deoarece nu mai există noduri active, algoritmul se termină. În figura 2.15(g) este reprezentat fluxul maxim obținut prin aplicarea algoritmului de scalare a excesului.

**Teorema 2.34.** *Algoritmul de scalare a excesului determină un flux maxim în rețeaua  $G = (N, A, c)$ .*

*Demonstrație.* La începutul algoritmului  $\bar{r} := 2^{\lceil \log \bar{c} \rceil}$ , deci  $\bar{c} \leq \bar{r} < 2\bar{c}$ . Pe parcursul fazei de  $\bar{r}$ -scalare,  $e_{\max}$  poate să crească și să descrească, dar trebuie să fie îndeplinite condițiile  $\bar{r}/2 \leq e_{\max} \leq \bar{r}$ . Când  $e_{\max} < \bar{r}/2$  atunci se înjumătățește valoarea lui  $\bar{r}$  și se începe o nouă fază de scalare. După  $\lceil \log \bar{c} \rceil + 1$  faze de scalare,  $e_{\max}$  devine nul și fluxul astfel obținut este un flux maxim. ■

**Teorema 2.35.** *Pe parcursul fiecărui fază de  $\bar{r}$ -scalare sunt îndeplinite următoarele două condiții:*

- (a) *la fiecare înaintare nesaturată se înaintează cel puțin  $\bar{r}/2$  unități de flux*
- (b)  $e_{\max} \leq \bar{r}$ .

*Demonstrație.* (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 distanță dintre toate nodurile cu exces mare. Deci,  $e(x) \geq \bar{r}/2$  și  $e(y) < \bar{r}/2$ . Deoarece pe arcul  $(x, y)$  se efectuează o înaintare nesaturată, rezultă că se mărește fluxul pe arcul  $(x, y)$  cu  $\min\{e(x), \bar{r} - e(y)\} \geq \bar{r}/2$  unități.

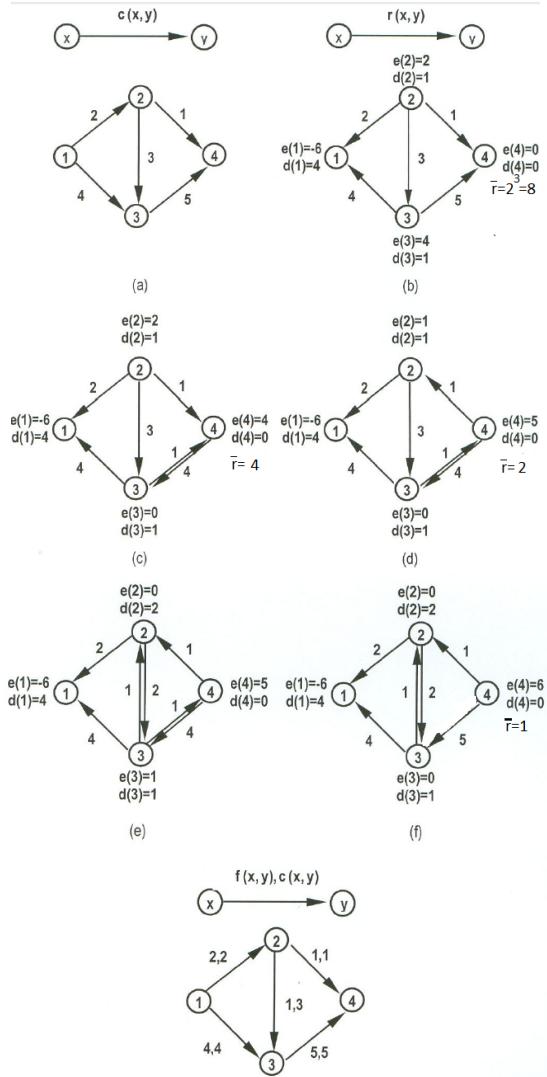


Fig.2.15

(b) O înaintare de flux pe arcul  $(x, y)$  mărește doar excesul nodului  $y$ . După înaintare, noul exces al nodului  $y$  va fi  $e(y) + \min\{e(x), \bar{r} - e(y)\} \leq \bar{r}$ . Deci,  $e_{\max} \leq \bar{r}$ . ■

**Teorema 2.36.** Pe parcursul fiecărei faze de scalare, algoritmul de scalare a excesului efectuează  $O(n^2)$  înaintări nesaturante.

*Demonstrație.* Considerăm funcția potențial  $P = \sum_{x \in N} e(x)d(x)/\bar{r}$ . Valoarea inițială a lui  $P$  la începutul fazei de  $\bar{r}$ –scalare este mărginită superior de  $2n^2$  deoarece  $e(x) \leq \bar{r}$  și  $d(x) < 2n$ ,  $x \in N$ , conform Teoremei 2.35(b) și Lemei 2.25. 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 distanță a nodului  $x$  crește cu  $q \geq 1$  unități, ceea ce duce la creșterea lui  $P$  cu cel mult

$q$  unități pentru că  $e(x) \leq \bar{r}$ . Deoarece, pentru fiecare nod  $x$ ,  $d(x)$  poate crește cu cel mult  $2n$  unități pe parcursul execuției algoritmului, rezultă că valoarea lui  $P$  crește cu cel mult  $2n^2$  unități 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  $P$  descrește. După o înaintare nesaturată pe arcul  $(x, y)$ , fluxul de la nodul  $x$  la nodul  $y$  crește, conform Teoremei 2.35(a) cu cel puțin  $\bar{r}/2$  unități, deci  $P$  descrește cu cel puțin  $1/2$  unități pentru că  $d(y) = d(x) - 1$ . Deoarece valoarea inițială a lui  $P$  la începutul fazei de  $\bar{r}$ - scalare era cel mult  $2n^2$  și putea să crească cu încă cel mult  $2n^2$  unități, 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 2.37** Complexitatea algoritmului de scalare a excesului este  $O(nm + n^2 \log \bar{c})$ .

*Demonstrație.* Conform Teoremei 2.36 rezultă că numărul total de înaintări nesaturate este  $O(n^2 \log \bar{c})$  deoarece algoritmul efectuează  $O(\log \bar{c})$  faze de scalare. Celelalte operații (înaintări saturate, reetichetări de noduri și determinări de arce admisibile) au complexitatea  $O(nm)$ , ca în algoritmul preflux generic. Deci, complexitatea algoritmului de scalare a excesului este  $O(nm + n^2 \log \bar{c})$ . ■

## 2.3 Aplicații

### 2.3.1 Problema reprezentanților

Un oraș are  $q$  locuitori,  $q'$  asociații și  $r$  partide politice. Fiecare locuitor face parte din cel puțin o asociație și poate să fie membru doar la un singur partid. Fiecare asociație trebuie să-și desemneze un membru care să-l reprezinte la consiliul de guvernare al orașului 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 rețea  $G = (N, A, c)$ . Multimea 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,  $N_2 = \{a_1, \dots, a_{q'}\}$  este multimea nodurilor care reprezintă asociațiile  $N_3 = \{l_1, \dots, l_q\}$  este multimea nodurilor care reprezintă locuitorii,  $N_4 = \{p_1, \dots, p_r\}$  este multimea nodurilor care reprezintă partidele. Multimea arcelor este  $A = A_1 \cup A_2 \cup A_3 \cup A_4$ , unde  $A_1 = \{(s, a_i) \mid a_i \in N_2\}$ ,  $A_2 = \{(a_i, l_j) \mid a_i \in N_2, l_j \in N_3$  și  $l_j$  face parte din  $a_i\}$ ,  $A_3 = \{(l_j, p_k) \mid l_j \in N_3, p_k \in N_4$  și  $l_j$  este membru la  $p_k\}$ ,  $A_4 = \{(p_k, t) \mid p_k \in N_4\}$ . Funcția capacitate  $c : A \rightarrow \mathcal{N}$  este  $c(x, y) = 1$  dacă  $(x, y)$  este din  $A_1$  sau  $A_2$  sau  $A_3$  și  $c(p_k, t) = u_k$ ,  $(p_k, t) \in A_4$ .

Dacă valoarea fluxului maxim în rețeaua  $G = (N, A, c)$  este egală cu  $q'$ , atunci orașul are un consiliu echilibrat, altfel consiliul este neechilibrat.

**Exemplul 2.15.** Pentru problema reprezentanților considerăm  $q = 7, q' = 4, r = 3, u_1 = 1, u_2 = 2, u_3 = 2$ . Rețeaua  $G = (N, A, c)$  este reprezentată în figura 2.16. Există un flux maxim cu valoarea egală cu  $q' = 4$ . Deci există un consiliu echilibrat. Soluția nu este unică.

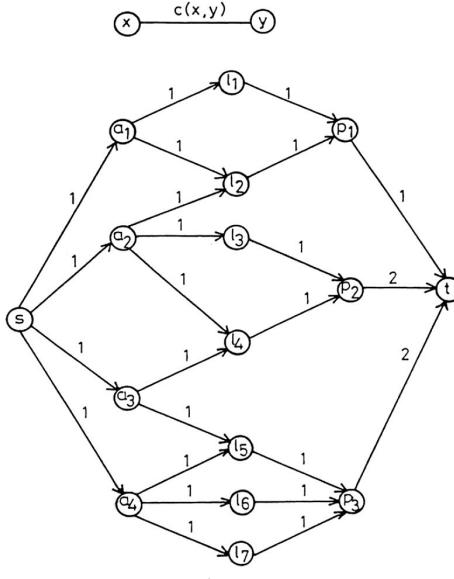


Fig.2.16

### 2.3.2 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

$$l_i = \sum_{j=1}^q r_{ij}, \quad i = 1, \dots, p; \quad c_j = \sum_{i=1}^p r_{ij}, \quad j = 1, \dots, q.$$

Se pune problema determinării matricei rotunjite  $[R] = ([r_{ij}])$  și a sumelor rotunjite  $[l_i]$ ,  $i = 1, \dots, p$ ;  $[c_j]$ ,  $j = 1, \dots, q$ , astfel încât

$$[l_i] = \sum_{j=1}^q [r_{ij}], \quad i = 1, \dots, p; \quad [c_j] = \sum_{i=1}^p [r_{ij}], \quad j = 1, \dots, q.$$

O astfel de rotunjire se numește *rotunjire consistentă*.

Problema rotunjirii consistente a unei matrice se poate modela ca o problemă de flux admisibil într-o rețea  $G = (N, A, l, c)$ . Multimea nodurilor este  $N = N_1 \cup N_2 \cup N_3$ , unde  $N_1 = \{s, t\}$ ,  $s$  este nodul sursă,  $t$  este nodul stoc,  $N_2 = \{x_1, \dots, x_p\}$ ,  $N_3 = \{y_1, \dots, y_q\}$ . Multimea arcelor este  $A = A_1 \cup A_2 \cup A_3$ , unde  $A_1 = \{(s, x_i) \mid x_i \in N_2\}$ ,  $A_2 = \{(x_i, y_j) \mid x_i \in N_2, y_j \in N_3\}$ ,  $A_3 = \{(y_j, t) \mid y_j \in N_3\}$ . Funcțiile  $l : A \rightarrow \mathcal{N}$ ,  $c : A \rightarrow \mathcal{N}$  sunt definite în modul următor:  $l(s, x_i) = [l_i]$ ,  $c(s, x_i) = [l_i]$ ,  $i = 1, \dots, p$ ;  $l(x_i, y_j) = [r_{ij}]$ ,  $c(x_i, y_j) = [r_{ij}]$ ,  $i = 1, \dots, p$ ,  $j = 1, \dots, q$ ;  $l(y_j, t) = [c_j]$ ,  $c(y_j, t) = [c_j]$ ,  $j = 1, \dots, q$ .

Problema rotunjirii matricelor apare în multe aplicații. De exemplu, Institutul de Statistică al României utilizează informații înregistrate în tabele. Informațiile conținute

în anumite tabele trebuie protejate. Putem proteja informațiile dintr-un tabel după cum urmează. Rotunjim fiecare înregistrare din tabel, inclusiv sumele pe linii și coloane, la un multiplu al unei constante corespunzătoare  $k$ , astfel încât înregistrările rotunjite din tabel adunate pe linii și coloane să dea sumele rotunjite pe linii și coloane. Această problemă este aceeași ca problema rotunjirii matricei discutată mai sus cu excepția 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 rețelei  $G = (N, A, l, c)$  asemănător ca mai sus, dar acum definim  $l(x, y)$  și  $c(x, y)$  unde  $(x, y)$  este asociat unui număr real  $r$ , ca cel mai mare multiplu de  $k$  mai mic sau egal cu  $r$  și respectiv cel mai mic multiplu de  $k$  mai mare sau egal cu  $r$ .

**Exemplul 2.16.** Să considerăm următoarea matrice

$$R = \begin{array}{ccc} & l & \\ \left[ \begin{array}{ccc} 3.1 & 6.8 & 7.3 \\ 9.6 & 2.4 & 0.7 \\ 3.6 & 1.2 & 6.5 \end{array} \right] & 17.2 & \\ & 12.7 & \\ & 11.3 & \\ c & 16.3 & 10.4 & 14.5 \end{array}$$

Rețeaua  $G = (N, A, l, c)$  pentru cazul  $k = 1$  este prezentată în figura 2.17 (a) și pentru cazul  $k = 2$  în figura 2.17 (b).

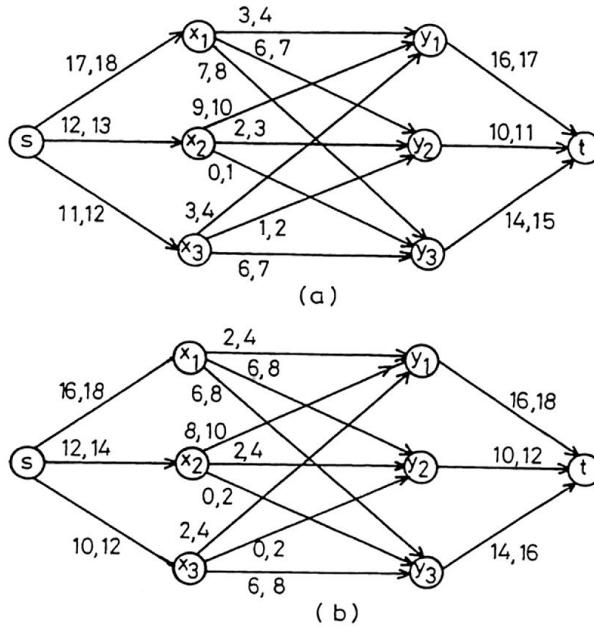


Fig.2.17

Rezolvând problema fluxului maxim, în cazul  $k = 1$ , obținem

$[l]$

$$[R] = \left[ \begin{array}{ccc} 4 & 6 & 8 \\ 9 & 3 & 1 \\ 4 & 2 & 6 \end{array} \right] \begin{array}{c} 18 \\ 13 \\ 12 \end{array}$$

$[c] \quad 17 \quad 11 \quad 15$



## Capitolul 3

# Fluxuri minime în rețele

### 3.1 Noțiuni introductive

Problema fluxului minim de la nodul sursă  $s$  la nodul stoc  $t$  în rețeaua  $G = (N, A, \ell, c, s, t)$  constă în a determina funcția flux  $f : A \leftarrow \mathbb{R}_+$  care îndeplinește constrângările:

$$\text{minimizează } v \quad (3.1.a)$$

$$\sum_y f(x, y) - \sum_y f(y, x) = \begin{cases} v, & \text{dacă } x = s \\ 0, & \text{dacă } x \neq s, t \\ -v, & \text{dacă } x = t \end{cases} \quad (3.1.b)$$

$$\ell(x, y) \leq f(x, y) \leq c(x, y), \quad (x, y) \in A. \quad (3.1.c)$$

Pentru problema fluxului minim, un preflux este o funcție  $f : A \rightarrow \mathbb{R}_+$  care îndeplinește condițiile:

$$\sum_y f(x, y) - \sum_y f(y, x) \leq 0, \quad x \in N \setminus \{s, t\} \quad (3.2.a)$$

$$\ell(x, y) \leq f(x, y) \leq c(x, y). \quad (3.2.b)$$

Fie  $f$  un preflux. *Deficitul* fiecărui nod  $x \in N$  este definit astfel:

$$e(x) = \sum_y f(x, y) - \sum_y f(y, x). \quad (3.3)$$

Spunem că un nod  $x$  este *echilibrat* dacă  $e(x) = 0$ . Un preflux  $f$  care îndeplinește condiția

$$e(x) = 0, \quad x \in N \setminus \{s, t\}$$

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 definește astfel

$$\hat{r}(x, y) = c(y, x) - f(y, x) + f(x, y) - \ell(x, y).$$

Reamintim ipoteza 4 din paragraful 1.2 : dacă arcul  $(x, y)$  aparține rețelei atunci și arcul  $(y, x)$  aparține rețelei. Capacitatea reziduală  $\hat{r}(x, y)$  a arcului  $(x, y)$  reprezintă cantitatea maximă cu care se poate micșora fluxul de la nodul  $x$  la nodul  $y$ . Rețeaua reziduală  $\hat{G}(f) = (\hat{N}, \hat{A})$  corespunzătoare prefluxului  $f$  este formată doar din arcele cu capacitatele reziduale pozitive, adică din arcele  $(x, y)$  cu  $\hat{r}(x, y) > 0$ . În figura 3.1(b) este reprezentată rețeaua reziduală  $\hat{G}(f)$  corespunzătoare prefluxului  $f$  din rețeaua din figura 3.1(a).

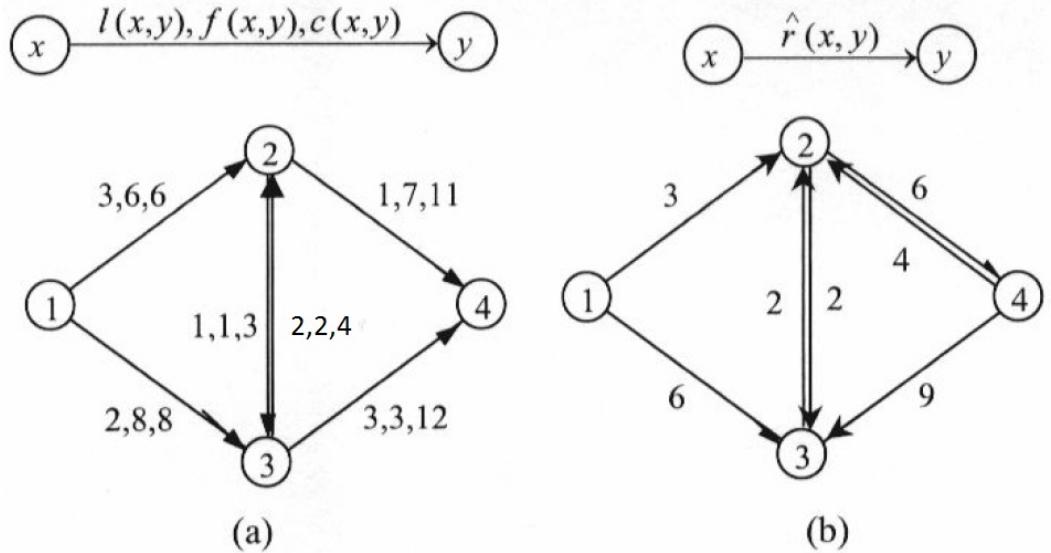


Fig.3.1

Pentru problema fluxului minim, definim capacitatea  $\hat{c}[X, \bar{X}]$  a unei tăieturi  $s - t$   $[X, \bar{X}]$  ca fiind diferența dintre suma marginilor inferioare ale arcelor directe și suma capacitateilor arcelor inverse, adică

$$\hat{c}[X, \bar{X}] = \ell(X, \bar{X}) - c(\bar{X}, X).$$

O tăietură  $s - t$  a cărei capacitate este maximă se numește *tăietură maximă*.

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

**Teorema 3.1. (Teorema valorii fluxului pentru problema fluxului minim).** În rețeaua  $G = (N, A, \ell, c, s, t)$  dacă  $f$  este un flux de valoare  $v$  și  $[X, \bar{X}]$  este o tăietură  $s - t$  atunci au loc relațiile:

$$v = f[X, \bar{X}] \geq \hat{c}[X, \bar{X}].$$

*Demonstrație.* Deoarece  $f$  este un flux în rețeaua  $G$ , rezultă că  $f$  satisfac constrângerile de conservare a fluxului (3.1.b). Însumând aceste ecuații pentru  $x \in X$  se obține  $v = f(X, N) - f(N, X)$ . Înlocuind  $N = X \cup \bar{X}$  în această relație și dezvoltând rezultă că  $v = f(X, X \cup \bar{X}) - f(X \cup \bar{X}, X) = f(X, X) + f(X, \bar{X}) - f(X, X) - f(\bar{X}, X) = f[X, \bar{X}]$ . De asemenea,  $f$  satisfac constrângerile de mărginire a fluxului (3.1c), deci  $f(X, \bar{X}) \geq \ell(X, \bar{X})$  și  $f(\bar{X}, X) \leq c(\bar{X}, X)$ . Rezultă că  $v = f[X, \bar{X}] = f(X, \bar{X}) - f(\bar{X}, X) \geq \ell(X, \bar{X}) - c(\bar{X}, X) = \hat{c}[X, \bar{X}]$ . ■

**Teorema 3.2. (Teorema fluxului minim și a tăieturii maxime).** *Dacă există un flux admisibil în rețeaua  $G = (N, A, \ell, c, s, t)$  atunci valoarea fluxului minim de la nodul sursă  $s$  la nodul stoc  $t$  este egală cu capacitatea tăieturii  $s - t$  maxime.*

*Demonstrație.* Rezultă din Teorema 3.1.

Determinarea unui flux minim într-o rețea  $G = (N, A, \ell, c, s, t)$  se face în două etape:

Etapa 1. Se determină un flux admisibil (a se vedea paragraful 1.6).

Etapa 2. Pornind de la un flux admisibil, se determină un flux minim.

În continuare vom prezenta algoritmi pentru Etapa 2.

## 3.2 Algoritmi pseudopolinomiali pentru fluxul minim

### 3.2.1 Algoritmul generic

Fie  $f$  un flux de valoare  $v$ ,  $L$  un lanț de la nodul  $s$  la nodul  $t$ , cu  $L^+$  mulțimea arcelor directe și  $L^-$  mulțimea arcelor inverse, în rețeaua  $G = (N, A, \ell, c, s, t)$ . Un lanț  $L$  de la nodul sursă  $s$  la nodul stoc  $t$  se numește *lanț de micșorare a fluxului* (LmF) în raport cu fluxul  $f$  dacă  $f(x, y) > \ell(x, y)$  pentru  $(x, y) \in L^+$  și  $f(y, x) < c(y, x)$  pentru  $(y, x) \in L^-$ .

Un LmF  $L$  în raport cu fluxul  $f$  în rețeaua  $G$  corespunde unui drum  $\hat{D}$  de micșorare a fluxului (DmF) în rețeaua reziduală  $\hat{G}(f)$  și reciproc. Definim capacitatea reziduală a unui LmF  $L$  în modul următor:

$$\begin{aligned} r(L^+) &= \min\{f(x, y) - \ell(x, y) \mid (x, y) \in L^+\}, \quad r(L^-) = \\ &= \min\{c(y, x) - f(y, x) \mid (y, x) \in L^-\}, \quad r(L) = \min\{r(L^+), r(L^-)\}. \end{aligned}$$

Evident că  $r(L) > 0$ . Se poate efectua micșorarea 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 3.3 (Teorema drumului de micșorare a fluxului).** *Un flux  $f^*$  este un flux minim dacă și numai dacă rețeaua reziduală  $\hat{G}(f^*)$  nu conține nici un drum de micșorare a fluxului.*

*Demonstrație.* Dacă  $f^*$  este un flux minim atunci în rețeaua reziduală  $\hat{G}(f^*)$  nu există nici un drum de micșorare a fluxului, altfel  $f^*$  nu ar fi un flux minim. Reciproc, dacă rețeaua reziduală  $\hat{G}(f^*)$  nu conține nici un drum de micșorare a fluxului rezultă că nodul  $t$  nu este accesibil din nodul  $s$  în  $\hat{G}(f^*)$ . Fie  $X^*$  mulțimea nodurilor accesibile din nodul  $s$  în  $\hat{G}(f^*)$ . Rezultă că  $[X^*, \bar{X}^*]$  este o tăietură  $s - t$  cu  $\hat{c}[X^*, \bar{X}^*] = v^*$ , unde  $v^*$  este valoarea fluxului  $f^*$ . Deci,  $f^*$  este un flux minim. ■

Cât timp rețeaua reziduală  $\hat{G}(f)$  conține un DmF putem micșora 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 rețeaua reziduală  $\hat{G}(f)$ ;
- (5)     WHILE  $\hat{G}(f)$  conține 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 micșorarea de flux;
- (10)        se actualizează  $\hat{G}(f)$ ;
- (11)     END
- (12) END.

Algoritmul generic pentru fluxul minim lucrează pe rețeaua reziduală și, la terminarea lui, se obțin capacitați reziduale optime. Pornind de la aceste capacitați 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,

$$f'(x, y) = \max\left(\hat{r}'(x, y) - c'(y, x), 0\right)$$

și

$$f'(y, x) = \max\left(\hat{r}'(y, x) - c'(x, y), 0\right).$$

Revenind la variabilele initiale, obținem

$$f(x, y) = \ell(x, y) + \max(\hat{r}(x, y) - c(y, x) + \ell(y, x), 0) \quad (3.4.a)$$

și

$$f(y, x) = \ell(y, x) + \max(\hat{r}(y, x) - c(x, y) + \ell(x, y), 0). \quad (3.4.b)$$

**Exemplul 3.1.** Ilustrăm algoritmul generic pentru fluxul minim pe rețeaua din figura 3.2(a). În figura 3.2(b) este reprezentată rețeaua reziduală corespunzătoare fluxului admisibil ilustrat în figura 3.2(a). Se determină  $DmF \hat{D} = (1, 2, 4)$  a cărui capacitate reziduală este  $\hat{r}(\hat{D}) = \min\{\hat{r}(1, 2), \hat{r}(2, 4)\} = \min\{3, 6\} = 3$  și se micșorează cu trei unități fluxul de-a lungul drumului  $\hat{D}$ . Rețeaua reziduală obținută după micșorarea fluxului de-a lungul drumului  $\hat{D} = (1, 2, 4)$  este reprezentată în figura 3.2(c). Se determină  $DmF \hat{D} = (1, 3, 2, 4)$ ,  $\hat{r}(\hat{D}) = \min\{\hat{r}(1, 3), \hat{r}(3, 2), \hat{r}(2, 4)\} = \min\{6, 2, 3\} = 2$  și se micșorează cu două unități fluxul de-a lungul drumului  $\hat{D} = (1, 3, 2, 4)$ , obținându-se rețeaua reziduală ilustrată în figura 3.2(d). Se determină  $DmF \hat{D} = (1, 3, 4)$ ,  $\hat{r}(\hat{D}) = \min\{\hat{r}(1, 3), \hat{r}(3, 4)\} = \min\{4, 4\} = 4$  și după ce se micșorează cu patru unități fluxul de-a lungul drumului  $\hat{D} = (1, 3, 4)$  se obține rețeaua reziduală reprezentată în figura 3.2(e), în care nu mai există  $DmF$  și algoritmul se termină. Pornind de la capacitatele reziduale optime determinate cu ajutorul algoritmului generic pentru fluxul minim și folosind relațiile (3.4) se determină fluxul minim care este ilustrat în figura 3.2(f).

**Teorema 3.4. (Teorema valorilor întregi).** *Dacă funcția capacitate  $c$  și funcția margine inferioară  $\ell$  sunt cu valori întregi și dacă există un flux admisibil atunci algoritmul generic pentru fluxul minim converge într-un număr finit de iterații și la terminarea execuției lui se obține un flux minim cu valori întregi.*

*Demonstrație.* Dacă funcția capacitate  $c$  și funcția limită inferioară  $\ell$  sunt cu valori întregi atunci fluxul admisibil  $f$  care este determinat rezolvând o problemă de flux maxim într-o rețea extinsă (a se vedea paragraful 1.6) are valori întregi, conform Teoremei 1.8. Dacă  $c, \ell$  și  $f$  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 iterații valoarea fluxului va ajunge la valoarea minimă și în rețeaua reziduală nu vor mai exista  $DmF$ . Deci, algoritmul generic pentru fluxul minim converge într-un număr finit de iterații. Deoarece  $c, \ell$  și  $f$  sunt cu valori întregi este evident că după fiecare micșorare de flux, atât fluxul cât și capacitatea reziduală sunt cu valori întregi. Deci, fluxul minim este cu valori întregi. ■

**Teorema 3.5.** *Dacă funcția capacitate  $c$ , funcția margine inferioară  $\ell$  și fluxul admisibil inițial  $f$  sunt cu valori întregi, atunci complexitatea algoritmului generic pentru fluxul minim este  $O(nm\bar{c})$ .*

*Demonstrație.* La fiecare iterație algoritmul efectuează o micșorare de flux în urma căreia valoarea fluxului scade cu cel puțin o unitate. Complexitatea unei micșorări de flux este  $O(m)$ , iar numărul de micșorări este  $O(n\bar{c})$ , pentru că valoarea fluxului este cuprinsă între 0 și  $n\bar{c}$ . Deci, complexitatea algoritmului generic pentru fluxul minim este  $O(nm\bar{c})$ . ■

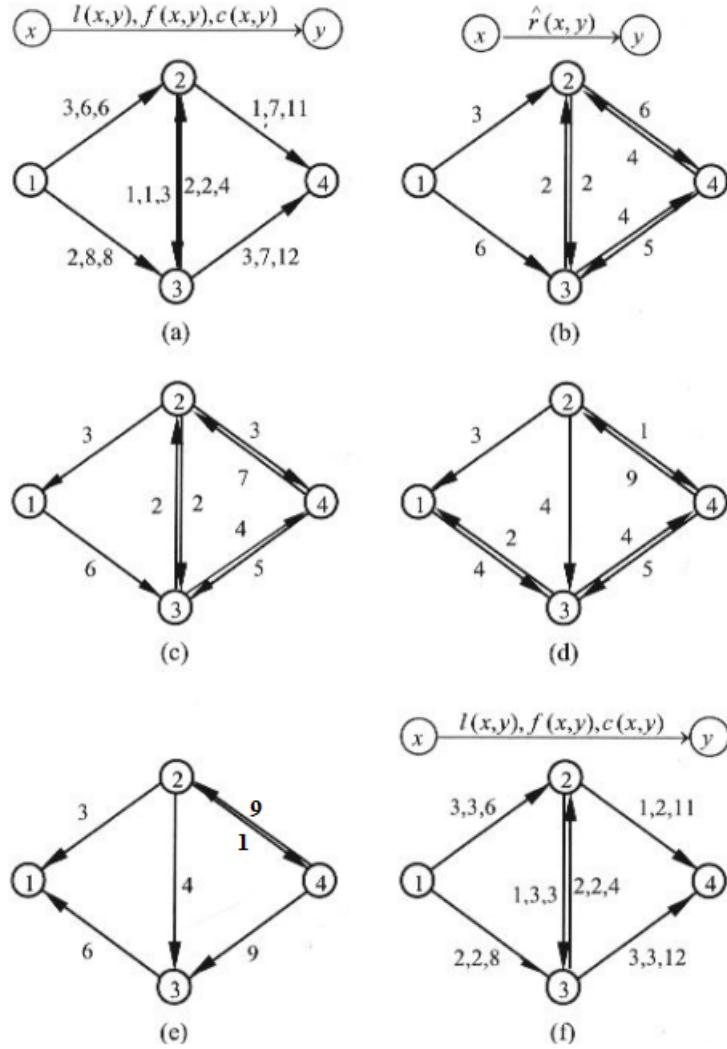


Fig.3.2

### 3.2.2 Varianta algoritmului Ford-Fulkerson

Varianta algoritmului Ford-Fulkerson este o versiune îmbunătățită a algoritmului generic pentru fluxul minim. Acest algoritm determină un DmF  $\hat{D}$  în rețeaua reziduală cu ajutorul unui vector  $n$ -dimensional numit predecesor și notat cu  $\hat{p}$ . Dacă, pe parcursul execuției 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 rețeaua reziduală. În plus, pentru fiecare succesor  $y$  al lui  $x$  către care nu s-a identificat deocamdată un drum care pornește din nodul  $s$ , se poate determina drumul  $\hat{D}_{sy} = \hat{D}_{sx} \cup \{(x, y)\}$  și 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 rețeaua 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  $\hat{V} \neq \emptyset$  și  $\hat{p}(t) = 0$  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  $\hat{p}(y) = 0$  și  $y \neq s$ 
(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 MICSORARE;
(19)       UNTIL  $\hat{p}(t) = 0$ ;
(20)     END.

```

```

(1) PROCEDURA MICSORARE;
(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 micșorarea de flux de-a lungul drumului  $\hat{D}$ ;
(6)   se actualizează  $\hat{G}(f)$ ;
(7) END;

```

**Exemplul 3.2.** Ilustrăm varianta algoritmului Ford-Fulkerson pe rețeaua din figura 3.3 (a). În figura 3.3(b) este reprezentată rețeaua reziduală corespunzătoare fluxului admisibil ilustrat în figura 3.3(a). La prima iterare vectorul predecesor  $\hat{p}$  poate fi  $(0, 1, 1, 2)$ . Rezultă că  $\hat{D} = (1, 2, 4)$  și  $\hat{r}(\hat{D}) = 3$ . Rețeaua reziduală obținută după micșorarea fluxului de-a lungul DmF  $\hat{D} = (1, 2, 4)$  este reprezentată în figura 3.3(c). La a doua iterare vectorul predecesor este  $\hat{p} = (0, 3, 1, 3)$ , iar DmF este  $\hat{D} = (1, 3, 4)$  cu  $\hat{r}(\hat{D}) = 4$ . Se micșorează cu patru unități fluxul de-a lungul DmF  $\hat{D} = (1, 3, 4)$  și se obține rețeaua reziduală din figura 3.3(d). La a treia iterare se determină  $\hat{p} = (0, 3, 1, 2)$ ,  $\hat{D} = (1, 3, 2, 4)$ ,  $\hat{r}(\hat{D}) = 2$  și se micșorează cu două unități fluxul de-a

lungul drumului  $\hat{D}$ , obținându-se rețeaua reziduală reprezentată în figura 3.3(e). În această rețea, se initializează  $\hat{V} = \{s\}$ , se obține  $\hat{V} = \emptyset$  și  $\hat{p}(t) = 0$  și algoritmul se termină. Pornind de la capacitatele reziduale optime determinate cu ajutorul variantei algoritmului Ford-Fulkerson și folosind relațiile (3.4) se determină fluxul minim care este ilustrat în figura 3.3(f).

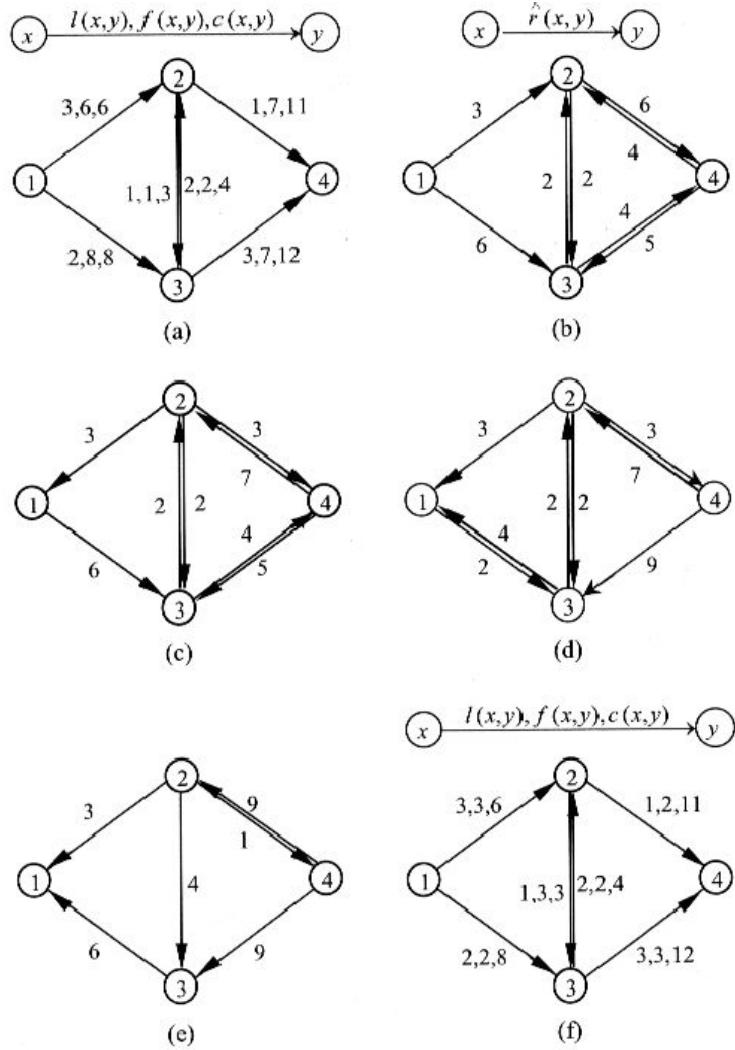


Fig.3.3

**Teorema 3.6.** Varianta algoritmului Ford-Fulkerson determină un flux minim de la nodul sursă  $s$  la nodul stoc  $t$ .

*Demonstrație.* Execuția algoritmului se termină atunci când nodului stoc  $t$  nu încă poate atribui un predecesor nenul.

Fie  $X^* = \{x \in N \mid \hat{p}(x) \neq 0\} \cup \{s\}$  și  $\bar{X}^* = N \setminus X^*$ . Evident că  $[X^*, \bar{X}^*]$  este o tăietură  $s - t$ . Deoarece  $\hat{p}(\bar{x}) = 0$  pentru oricare  $\bar{x} \in \bar{X}^*$  rezultă că  $\hat{r}(x, \bar{x}) = 0$  pentru orice arc  $(x, \bar{x}) \in (X^*, \bar{X}^*)$ . Dar  $\hat{r}(x, \bar{x}) = c(\bar{x}, x) - f(\bar{x}, x) + f(x, \bar{x}) - \ell(x, \bar{x})$ ,  $c(\bar{x}, x) - f(\bar{x}, x) \geq 0$ ,  $f(x, \bar{x}) - \ell(x, \bar{x}) \geq 0$  și condițiile  $\hat{r}(x, \bar{x}) = 0$  implică faptul că  $f(x, \bar{x}) = \ell(x, \bar{x})$  pentru orice arc  $(x, \bar{x}) \in (X^*, \bar{X}^*)$  și  $f(\bar{x}, x) = c(\bar{x}, x)$  pentru orice arc  $(\bar{x}, x) \in (\bar{X}^*, X^*)$ . Rezultă că  $v = f[X^*, \bar{X}^*] = f(X^*, \bar{X}^*) - f(\bar{X}^*, X) = \ell(X^*, \bar{X}^*) - c(\bar{X}^*, X^*) = \hat{c}[X^*, \bar{X}^*]$ , adică fluxul  $f$  de valoare  $v$  obținut la terminarea algoritmului este un flux minim. ■

**Teorema 3.7.** *Dacă funcția capacitate  $c$ , funcția margine inferioară  $\ell$  și fluxul admisibil initial  $f$  sunt cu valori întregi atunci varianta algoritmului Ford-Fulkerson pentru fluxul minim are complexitatea  $O(nm\bar{c})$ .*

*Demonstrație.* La fiecare iterare algoritmul efectuează o micșorare de flux în urma căreia valoarea fluxului scade cu cel puțin o unitate. Cum valoarea fluxului este cuprinsă între 0 și  $n\bar{c}$ , rezultă că algoritmul efectuează  $O(n\bar{c})$  iterării. Deoarece pentru determinarea unui DmF sunt necesare cel mult  $m$  examinări de arce, rezultă că complexitatea unei iterări este  $O(m)$ , iar complexitatea algoritmului este  $O(nm\bar{c})$ . ■

### 3.3 Algoritmi polinomiali pentru fluxul minim

#### 3.3.1 Noțiuni introductive

Vom defini etichetele distanță pentru problema fluxului minim.

**Definiția 3.1.** Într-o rețea reziduală  $\hat{G}(f) = (\hat{N}, \hat{A}, \hat{r})$  o funcție  $d : \hat{N} \rightarrow \mathcal{N}$  se numește funcție *distanță*. Funcția distanță  $d$  se numește *validă* dacă satisfac următoarele două condiții:

$$d(s) = 0 \quad (3.5)$$

$$d(y) \leq d(x) + 1, \quad (x, y) \in \hat{A} \quad (3.6)$$

Valoarea  $d(x)$  se numește *eticheta distanță* a nodului  $x$ , iar condițiile (3.5) și (3.6) se numesc *condiții de validitate*.

**Proprietatea 3.8.** (a) *Dacă etichetele distanță sunt valide, atunci eticheta distanță  $d(x)$  este o margine inferioară pentru lungimea drumului celui mai scurt de la nodul  $s$  la nodul  $x$  în rețeaua reziduală  $\hat{G}(f)$ .*

(b) *Dacă  $d(t) \geq n$  atunci în rețeaua reziduală  $\hat{G}(f)$  nu există drum de la nodul  $s$  la nodul  $t$ .*

*Demonstrație.* (a) Fie  $D = (x_1, x_2, \dots, x_k, x_{k+1})$  cu  $x_1 = s$ ,  $x_{k+1} = x$  un drum oarecare de lungime  $k$  de la nodul  $s$  la nodul  $x$  în rețeaua reziduală. Condițiile de validitate implică faptul că:

$$d(x_2) \leq d(x_1) + 1 = d(s) + 1 = 1$$

$$d(x_3) \leq d(x_2) + 1 \leq 2$$

...

$$d(x_{k+1}) \leq d(x_k) + 1 \leq k.$$

Deci  $d(x) = d(x_{k+1}) \leq k$ , ceea ce demonstrează prima parte a proprietății.

(b) Eticheta distanță  $d(t)$  este o margine inferioară pentru lungimea drumului celui mai scurt de la nodul  $s$  la nodul  $t$  în rețeaua 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 rețeaua reziduală  $\hat{G}(f)$  nu există drum de la nodul  $s$  la nodul  $t$ .

**Definiția 3.2.** Etichetele distanță 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 rețeaua reziduală  $\hat{G}(f)$ .

Etichetele distanță exacte se pot determina cu ajutorul algoritmului de parcurgere  $BF$  aplicat rețelei reziduale  $\hat{G}(f)$  plecând de la nodul sursă  $s$ .

**Definiția 3.3.** Un arc  $(x, y)$  din rețeaua reziduală  $\hat{G}(f)$  se numește *admisibil* dacă satisfac condiția  $d(y) = d(x) + 1$ ; în caz contrar arcul  $(x, y)$  se numește *inadmisibil*. Un drum de la nodul sursă  $s$  la nodul stoc  $t$  în rețeaua reziduală  $\hat{G}(f)$  se numește *drum admisibil* dacă este format doar din arce admisibile, altfel se numește *inadmisibil*.

**Proprietatea 3.9.** În rețeaua reziduală  $\hat{G}(f)$  un drum admisibil de la nodul sursă  $s$  la nodul stoc  $t$  este un cel mai scurt drum de micșorare a fluxului.

*Demonstrație.* Deoarece fiecare arc  $(x, y)$  dintr-un drum admisibil  $D$  este admisibil, capacitatea reziduală  $\hat{r}(x, y)$  și etichetele distanță  $d(x)$  și  $d(y)$  îndeplinesc următoarele condiții: (1)  $\hat{r}(x, y) > 0$  și (2)  $d(y) = d(x) + 1$ . Condiția (1) implică faptul că  $D$  este un drum de micșorare a fluxului, iar condiția (2) implică faptul că dacă  $D$  conține  $k$  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 rețeaua reziduală  $\hat{G}(f)$ , rezultă că  $D$  trebuie să fie un cel mai scurt drum de micșorare a fluxului. ■

### 3.3.2 Algoritmi polinomiali cu drumuri de micșorare

Algoritmii pseudopolinomiali cu drumuri de micșorare prezentați în paragraful 3.2 au fost obținuți din algoritmii pentru fluxul maxim în modul următor: s-a construit rețeaua reziduală  $\hat{G}(f)$  pentru problema fluxului minim în locul rețelei 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 aceleași reguli după care se determină drumuri de la  $s$  la  $t$  în  $\tilde{G}(f)$  în algoritmii corespunzători pentru fluxul maxim și s-a micșorat fluxul de-a lungul acestor drumuri. Această transformare poate fi aplicată oricărui algoritm cu drumuri de mărire pentru fluxul maxim, obținându-se astfel un algoritm similar pentru fluxul minim, care va avea aceeași complexitate ca algoritmul pentru fluxul maxim din care a fost obținut. Algoritmii polinomiali cu drumuri de mărire din paragraful 2.1 sunt prezentați în Tabelul 3.1.

Tabelul 3.1

Algoritm polinomial cu drumuri de mărire	Complexitate
Algoritmul Gabow al scalării bit a capacitații	$O(nm \log \bar{c})$
Algoritmul Ahuja - Orlin al scalării maxime a capacitații	$O(nm \log \bar{c})$
Algoritmul Edmonds - Karp al drumului celui mai scurt	$O(nm^2)$
Algoritmul Ahuja - Orlin al drumului celui mai scurt	$O(n^2m)$
Algoritmul Dinic al rețelelor startificate	$O(n^2m)$
Algoritmul Ahuja - Orlin al rețelelor stratificate	$O(n^2m)$

În continuare vom prezenta, pentru exemplificare, varianta algoritmului Ahuja - Orlin al drumului celui mai scurt. Acest algoritm folosește etichetele de distanță exacte și micșorează fluxul de-a lungul drumurilor admisibile.

**Definiția 3.4** În rețea reziduală, un drum de la un nod oarecare  $x$  la nodul stoc  $t$  format doar din arce admisibile se numește drum *partial admisibil*. Nodul  $x$  se numește 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 rețea  $G$ ;
- (4)     se determină rețea reziduală  $\hat{G}(f)$ ;
- (5)     se calculează etichetele distanță exacte  $d(\cdot)$  în  $\hat{G}(f)$ ;
- (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)                             MICȘORARE;
- (15)                              $y := t$ ;
- (16)                         END;
- (17)                 END
- (18)                 ELSE ÎNAPOIERE  $(y)$ ;
- (19)         END;
- (20)     END.
  
- (1) PROCEDURA ÎNAINTARE  $(x, y)$ ;
- (2) BEGIN
- (3)      $\hat{u}(x) := y$
- (4)      $y := x$ ;
- (5) END;

```

(1) PROCEDURA ÎNAPOIERE ( $y$ );
(2) BEGIN
    (3)    $d(y) := \min\{d(x) \mid (x, y) \in \hat{A}\} + 1$ ;
    (4)   IF  $y \neq t$ 
        (5)       THEN  $y := \hat{u}(y)$ ;
    (6) END;

(1) PROCEDURA MICȘORARE;
(2) BEGIN
    (3)   se determină un DmF  $\hat{D}$  utilizând vectorul succesor  $\hat{u}$ ;
    (4)   se determină  $g(\hat{D}) = \min\{\hat{r}(x, y) \mid (x, y) \in \hat{D}\}$ ;
    (5)   se execută micșorarea de flux;
    (6)   se actualizează rețeaua reziduală  $\hat{G}(f)$ ;
(7) END;

```

Algoritmul menține un drum parțial admisibil memorat cu ajutorul vectorului succesor  $\hat{u}$  și execută operații 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 operație de înaintare: se adaugă arcul  $(x, y)$  la începutul drumului parțial admisibil și se reține prin  $\hat{u}(x) = y$ , altfel se execută o operație de înapoiere. Operația de înapoiere constă în reetichetarea nodului  $y$ ,  $d(y) = \min\{d(x) \mid (x, y) \in \hat{A}\} + 1$  și eliminarea arcului  $(y, \hat{u}(y))$  din drumul parțial admisibil dacă  $y \neq t$ . Repetăm aceste operații până când drumul parțial admisibil devine un drum admisibil și atunci executăm o micșorare de flux. Continuăm acest proces până când fluxul devine minim.

**Exemplul 3.3.** Ilustrăm varianta algoritmului Ahuja-Orlin al drumului celui mai scurt pe rețeaua din figura 3.4(a). În figura 3.4(b) este reprezentată rețeaua reziduală corespunzătoare fluxului admisibil ilustrat în figura 3.4(a). Pornim de la nodul stoc  $t = 4$  cu un drum admisibil parțial vid. Executăm o operație de înaintare și adaugăm arcul  $(2, 4)$  la drumul parțial admisibil. Memorăm acest drum utilizând vectorul succesor  $\hat{u}$ , astfel  $\hat{u}(2) = 4$ . Acum nodul 2 este nodul curent și algoritmul execută o operație de înaintare de la nodul 2. Se efectuează  $\hat{u}(1) = 2$  și se adaugă arcul  $(1, 2)$  la începutul drumului admisibil parțial, obținându-se drumul admisibil  $\hat{D} = (1, 2, 4)$ . Se determină  $g(\hat{D}) = \min\{\hat{r}(1, 2), \hat{r}(2, 4)\} = \min\{3, 6\} = 3$  și se micșorează cu 3 unități fluxul de-alungul drumului  $\hat{D} = (1, 2, 4)$  obținându-se rețeaua reprezentată în figura 3.4(c). Din nou, pornim de la nodul stoc  $t = 4$  cu un drum admisibil parțial vid. Executăm o operație de înaintare și adaugăm arcul  $(2, 4)$  la drumul parțial admisibil și efectuăm  $\hat{u}(2) = 4$ . Deoarece nodul curent, care este nodul 2, nu este extremitatea finală a nici unui arc admisibil, efectuăm o operație de înapoiere:  $d(2) = \min\{d(3), d(4)\} + 1 = 2$ , nodul curent devine nodul  $\hat{u}(2) = 4$  și eliminăm arcul  $(2, 4)$  din drumul parțial ad-

misibil. La următoarele iterații, se vor determina drumurile admisibile  $\hat{D} = (1, 3, 4)$ , apoi  $\hat{D} = (1, 3, 2, 4)$ . Rețeaua obținută după micșorarea cu 4 unități a fluxului de-a lungul drumului  $\hat{D} = (1, 3, 4)$  este reprezentată în figura 3.4(d). După micșorarea cu 2 unități a fluxului de-a lungul drumului  $\hat{D} = (1, 3, 2, 4)$  se obține rețeaua reziduală corespunzătoare fluxului minim care este ilustrată în figura 3.4(e). Pornind de la capacitatele reziduale optime și folosind relațiile (3.4) se determină fluxul minim care este reprezentat în figura 3.4(f).

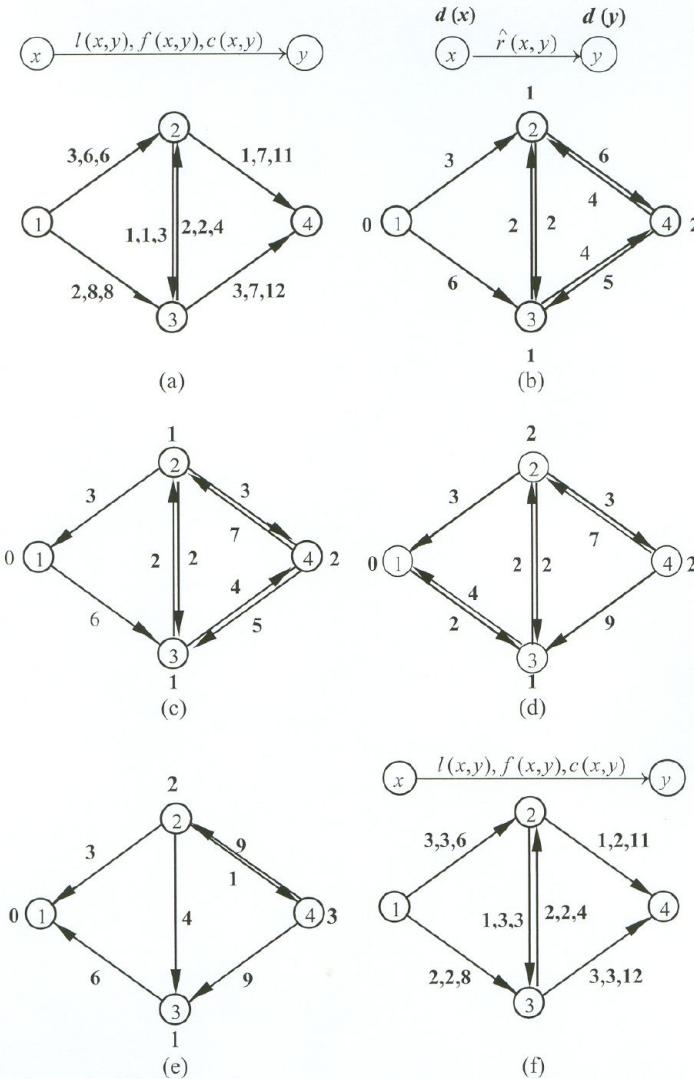


Fig.3.4

**Teorema 3.10.** Varianta algoritmului Ahuja - Orlin al drumului celui mai scurt determină un flux minim în rețeaua  $G = (N, A, \ell, c, s, t)$ .

*Demonstrație.* Analog cu demonstrația Teoremei 2.10 de corectitudine a algoritmului Ahuja - Orlin al drumului celui mai scurt pentru fluxul maxim (a se vedea paragraful 2.1.5.). ■

**Teorema 3.11.** *Varianta algoritmului Ahuja - Orlin al drumului celui mai scurt are complexitatea  $O(n^2m)$ .*

*Demonstrație.* Analiza complexității algoritmului se bazează pe următoarele două rezultate:

- (1) complexitatea micșorărilor de flux este  $O(n^2m)$ .
- (2) numărul operațiilor 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 2.1.5.), rezultă că varianta algoritmului Ahuja - Orlin are complexitatea  $O(n^2m)$ . ■

### 3.3.3 Algoritmi polinomiali cu prefluxuri

#### Algoritmul preflux generic

Fie  $f$  un preflux în rețeaua  $G = (N, A, \ell, c, s, t)$ . Un nod  $x \in N \setminus \{s, t\}$  se numește *nod activ* dacă  $e(x) < 0$ . Prin convenție nodurile  $s$  și  $t$  nu sunt active niciodată. Existența nodurilor active indică faptul că prefluxul nu este flux. De aceea, operația de bază a algoritmului constă în a selecta un nod activ și 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 față de nodul sursă se măsoară cu ajutorul etichetelor distanță 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 distanță astfel încât să se creeze cel puțin un arc admisibil. Această mărire a etichetei distanță va fi numită operație de reetichetare. Algoritmul se termină atunci când în rețea 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 rețeaua  $G$ ;
- (4)     se determină rețeaua reziduală  $\hat{G}(f)$ ;
- (5)     se calculează etichetele distanță exacte  $d(\cdot)$  în rețeaua 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 rețeaua reziduală  $\hat{G}(f)$  există un nod activ DO

```

(12)      BEGIN
(13)          se selectează un nod activ  $y$ ;
(14)          IF în rețeaua reziduală  $\hat{G}(f)$  există un arc
                  admisibil  $(x, y)$ 
(15)              THEN se retrag  $g = \min(-e(y), \hat{r}(x, y))$  unități
                  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 3.4.** Ilustrăm algoritmul preflux generic pentru fluxul minim pe rețeaua din figura 3.5(a). În figura 3.5(b) este reprezentată rețeaua reziduală corespunzătoare fluxului admisibil ilustrat în figura 3.5(a). În rețeaua reziduală obținută după inițializările din liniile (9) și (10) și reprezentată în figura 3.5(c) există două noduri active: 2 și 3. Presupunem că algoritmul selectează nodul activ 3. Rezultă că va retrage  $g = \min(-e(3), \hat{r}(1, 3)) = \min(4, 6) = 4$  unități de flux pe arcul admisibil (1,3). În urma acestei operații, nodul 3 nu mai este activ. Apoi algoritmul va selecta singurul nod activ, nodul 2 și va retrage  $g = \min(-e(2), \hat{r}(1, 2)) = 3$  unități de flux pe arcul admisibil (1,2). Nodul 2 a rămas activ și nu este extremitatea finală a nici unui arc admisibil, deci va fi reetichetat:  $d(2) = \min\{d(3), d(4)\} + 1 = 2$ . Rețeaua reziduală astfel obținută este reprezentată în figura 3.5(d). Prin operația de reetichetare s-a creat arcul admisibil (3,2), pe care se vor retrage 2 unități de flux. Cum nodul 2 este încă continuare activ și nu mai există arce admisibile cu extremitatea finală nodul 2, rezultă că va fi reetichetat:  $d(2) = 5$ . În rețeaua reziduală obținută, care este reprezentată în figura 3.5(e) există două noduri active: 2 și 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 unități de flux pe arcul admisibil (1,3). Astfel s-a obținut rețeaua din figura 3.5(f), în care nu mai există noduri active și algoritmul se termină. Pornind de la capacitatele reziduale optime determinate cu algoritmul preflux generic pentru fluxul minim și folosind relațiile (3.4) se determină fluxul minim care este ilustrat în figura 3.5(g).

**Teorema 3.12.** *Algoritmul preflux generic pentru fluxul minim determină un flux minim în rețeaua  $G = (N, A, \ell, c, s, t)$ .*

*Demonstrație.* 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 rețeaua 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 rețeaua reziduală; în caz contrar spunem că este o retragere de flux *incompletă*.

**Teorema 3.13.** *Complexitatea algoritmului preflux generic pentru fluxul minim este  $O(n^2m)$ .*

*Demonstrație.* Analiza complexității algoritmului preflux generic pentru fluxul minim se bazează pe următoarele trei rezultate:

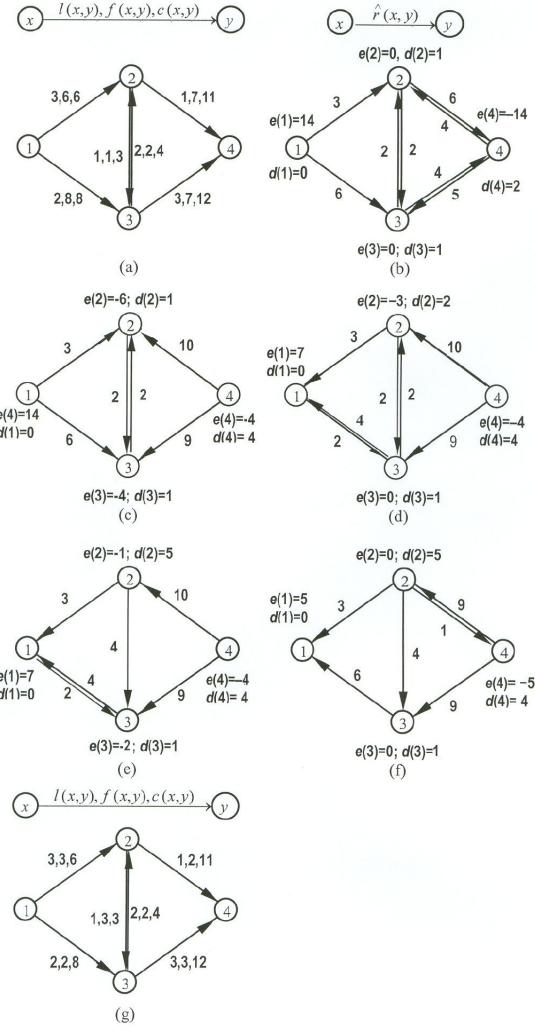


Fig.3.5

- (1) numărul total de operații 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 algoritm preflux generic pentru fluxul maxim (a se vedea paragraful 2.2.1.).

Dacă nodurile active sunt păstrate într-o listă înlănțuită atunci complexitatea operațiilor de adăugare, stergere și selectare a unui nod activ este  $O(1)$ . Deci, complexitatea algoritmului generic pentru fluxul minim este  $O(n^2m)$ . ■

În continuare sugerăm o îmbunătățire practică a algoritmului preflux generic pentru fluxul minim. Fie  $v^*$  valoarea fluxului minim. Un preflux  $f$  cu  $e(s) = v^*$  se numește *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 operații de reetichetare până când etichetele distanță ale nodurilor active devin mai mari decât  $n$  și se poate retrage flux de la nodurile active către nodul stoc  $t$ , a cărui etichetă distanță este  $n$ . O posibilă modificare a algoritmului constă în introducerea mulțimii  $N'$  a nodurilor  $x$  cu proprietatea că în rețeaua reziduală nu există drum de la  $s$  la  $x$ . Inițial  $N' = \{t\}$ . Folosim vectorul  $q$ , cu  $q(k)$  egal cu numărul de noduri care au etichetele distanță egale cu  $k$ . După fiecare operație de reetichetare actualizăm vectorul  $q$  în modul următor: Dacă algoritmul mărește eticheta distanță 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 rețeaua 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 mulțimea  $N'$ . 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 obținut este un preflux minim. Conform Teoremei 1.1, orice preflux poate fi descompus într-o sumă de cel mult  $m + n$  fluxuri nenule de-a lungul unor drumuri și circuite. Fie  $M$  – mulțimea acestor drumuri și circuite și  $M_0 \subseteq M$  – mulțimea 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.

### Algoritmul preflux FIFO

La fiecare iterare, algoritmul preflux generic pentru fluxul minim selectează un nod activ  $y$  și 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 iterarea următoare. Algoritmul preflux FIFO lucrează după următoare regulă: un nod  $y$  este selectat la intervale consecutive până când  $e(y) = 0$  sau  $y$  este reetichetat. În consecință, 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 operație de reetichetare (nu există arc admisibil  $(x, y)$ ). Numim această secvență de operații *examinarea nodului*  $y$ .

Notăm cu  $L$  lista nodurilor active. Algoritmul preflux FIFO pentru fluxul minim examinează nodurile active în ordinea primul intrat, primul ieșit (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 rețeaua  $G$ ;
- (4)     se determină rețeaua reziduală  $\hat{G}(f)$ ;
- (5)     se calculează etichetele distanță exacte  $d(\cdot)$  în rețeaua reziduală  $\hat{G}(f)$ ;
- (6)     IF  $t$  nu este etichetat
- (7)         THEN  $f$  este un flux minim
- (8)         ELSE BEGIN
- (9)              $L := \emptyset$ ;

```

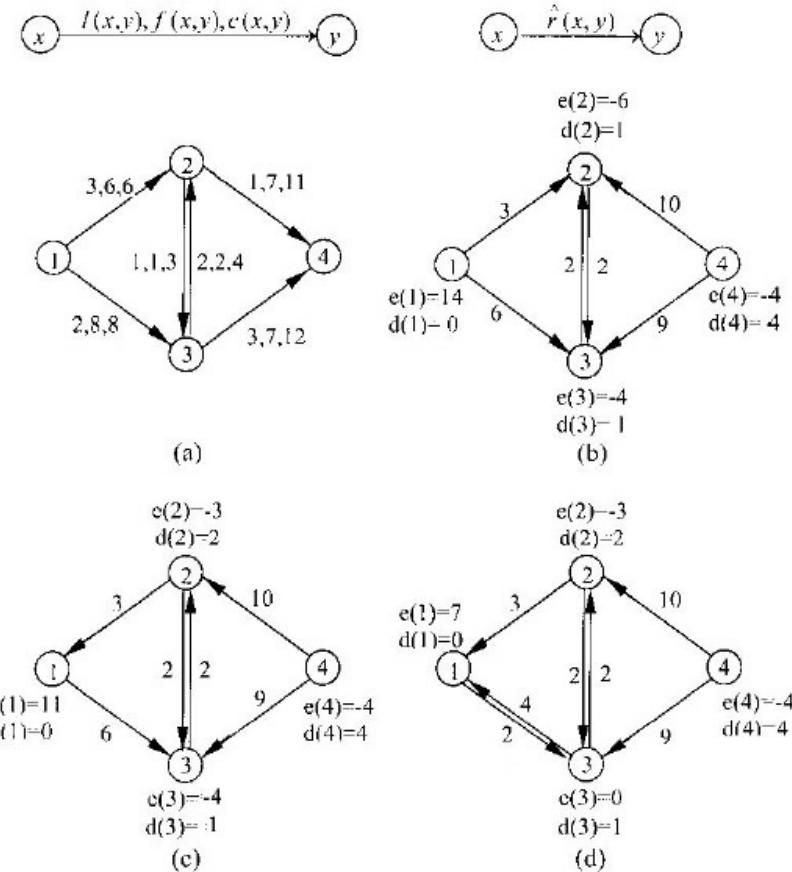
(10)      FOR  $(x, t) \in E^-(t)$  DO
(11)      BEGIN
(12)           $f(x, t) := \ell(x, t);$ 
(13)          IF  $e(x) < 0$  și  $x \neq s$ 
(14)              THEN se adaugă  $x$  la urma cozii  $L$ ;
(15)          END;
(16)           $d(t) := n;$ 
(17)          WHILE  $L \neq \emptyset$  DO
(18)              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  $(x, y)$  din  $\hat{G}(f)$ ;  
(4)  $B := 1$ ;  
(5) REPEAT  
(6) IF  $(x, y)$  este arc admisibil  
(7) THEN BEGIN  
(8) se retrag  $g := \min(-e(y), \hat{r}(x, y))$  unități de flux  
de la nodul  $y$  la nodul  $x$ ;  
(9) IF  $x \notin L$  și  $x \neq s$  și  $x \neq t$   
(10) THEN se adaugă  $x$  la urma 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  $(x, y)$   
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  $y$  la urma cozii  $L$ ;  
(22) END;

**Exemplul 3.5.** Ilustrăm algoritmul preflux FIFO pentru fluxul minim pe rețeaua din figura 3.6(a), în care este ilustrat și un flux admisibil. După inițializările din liniile

(9) - (16) se obține rețeaua reziduală din figura 3.6(b) și  $L = \{2, 3\}$ . Algoritmul scoate nodul 2 din coada  $L$  și îl examinează: retrage 3 unități de flux pe arcul admisibil (1,2), iar apoi, pentru că nu mai există arce admisibile cu extremitatea finală 2 îl reetichetează și-l adaugă la sfârșitul cozii  $L$ . Deci,  $L = \{3, 2\}$ , iar rețeaua reziduală este reprezentată în figura 3.6(c). Apoi algoritmul scoate nodul 3 din coada  $L$  și-l examinează retrăgând 4 unități de flux pe arcul (1,3). Astfel nodul 3 nu mai este activ,  $L = \{2\}$ . Rețeaua reziduală obținută după examinarea nodului 3 este reprezentată în figura 3.6(d). În continuare algoritmul scoate din coada  $L$  nodul 2, retrage 2 unități de flux pe arcul admisibil (3,2), adaugă nodul 3 în coadă deoarece a redevenit activ, reetichetează nodul 2:  $d(2) = 5$  și-l adaugă la sfârșitul cozii  $L$ . După examinarea nodului 2 se obține rețeaua reziduală din figura 3.6(e), iar  $L = \{3, 2\}$ . Apoi algoritmul scoate nodul 3 din coada  $L$  și-l examinează. Se retrag 2 unități de flux pe arcul admisibil (1,3) și nodul 3 nu mai este activ. Rețeaua reziduală astfel obținută este reprezentată în figura 3.6(f). Algoritmul va scoate nodul 2 din coadă și va retrage o unitate de flux pe arcul (4,2). Astfel nodul 2 nu mai este activ,  $L = \emptyset$  și algoritmul se termină. Pornind de la capacitatele reziduale optime determinate cu algoritmul preflux FIFO pentru fluxul minim, care sunt ilustrate în figura 3.6(g), se determină fluxul minim din figura 3.6(h).



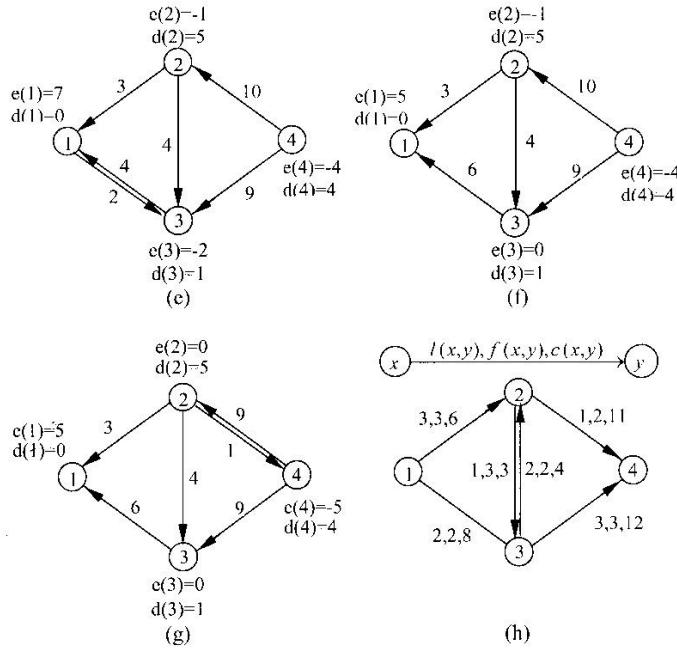


Fig.3.6

**Teorema 3.14.** Algoritmul preflux FIFO pentru fluxul minim determină un flux minim în rețeaua  $G = (N, A, \ell, c, s, t)$ .

*Demonstrație.* Rezultă din Teorema 3.12. ■

**Teorema 3.15.** Complexitatea algoritmului preflux FIFO pentru fluxul minim este  $O(n^3)$ .

*Demonstrație.* Analog cu demonstrația complexității algoritmului preflux FIFO pentru fluxul maxim. ■

### 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 distanță cea mai mare. Fie  $p = \max\{d(x) \mid x \text{ este nod activ}\}$ . Algoritmul examinează mai întâi nodurile active cu etichetele distanță egale cu  $p$  și retrage flux la nodurile cu etichetele distanță egale cu  $p - 1$ . Apoi de la aceste noduri retrage fluxul la nodurile cu etichetele distanță egale cu  $p - 2$  și aşa mai departe până când fie algoritmul reetichetează un nod, fie a epuizat toate nodurile active. După reetichetarea unui nod, algoritmul repetă același proces. Dacă algoritmul nu efectuează nici o reetichetare pe parcursul a  $n$  examinări successive de noduri atunci tot deficitul ajunge în nodul sursă sau înapoi în nodul stoc și 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 ușor dacă în algoritmul preflux FIFO pentru fluxul minim lista  $L$  a nodurilor active este organizată nu ca o coadă obișnuită, ci ca o coadă priorită cu prioritatea funcția distanță  $d$ .

```

(1) PROGRAM PREFLUX ETICH-Fm;
(2) BEGIN
(3)   fie  $f$  un flux admisibil în rețeaua  $G$ ;
(4)   se determină rețeaua reziduală  $\hat{G}^{\wedge}(f)$ ;
(5)   se calculează etichetele distanță exacte  $d(\cdot)$  în rețeaua reziduală  $\hat{G}^{\wedge}(f)$ ;
(6)   IF  $t$  nu este etichetat
(7)     THEN  $f$  este un flux minim
(8)   ELSE BEGIN
(9)      $L := \emptyset$ ;
(10)    FOR  $(x, t) \in E^-(t)$  DO
(11)      BEGIN
(12)         $f(x, t) := \ell(x, t)$ ;
(13)        IF  $e(x) < 0$  și  $x \neq s$  THEN
(14)          se adaugă  $x$  în coada  $L$  având prioritatea  $d(x)$ ;
(15)      END;
(16)       $d(t) := n$ ;
(17)      WHILE  $L \neq \emptyset$  DO
(18)        BEGIN
(19)          se scoate din  $L$  nodul  $y$  cu prioritatea cea mai mare;
(20)          RETRAGERE / REETICHETARE( $y$ );
(21)        END;
(22)      END;
(23)  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
(8)       THEN BEGIN
(9)         se retrag  $g := \min(-e(y), \hat{r}^{\wedge}(x, y))$  unități de flux
(10)        de la nodul  $y$  la nodul  $x$ ;
(11)        IF  $x \notin L$  și  $x \neq s$  și  $x \neq t$ 
(12)          THEN se adaugă  $x$  în coada  $L$  având prioritatea  $d(x)$ ;
(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  $e(y) < 0$ 
(22)              THEN se adaugă  $y$  în coada  $L$  având prioritatea  $d(y)$ ;
(23)      END;

```

**Teorema 3.16.** *Dacă există un flux admisibil în rețeaua  $G$ , atunci algoritmul preflux cu eticheta cea mai mare pentru fluxul minim determină un flux minim în rețeaua  $G = (N, A, \ell, c, s, t)$ .*

*Demonstrație.* Rezultă din Teorema 3.12. ■

**Teorema 3.17.** *Complexitatea algoritmului preflux cu eticheta cea mai mare pentru fluxul minim este  $O(n^2 m^{1/2})$ .*

*Demonstrație.* Analog cu demonstrația complexității algoritmului preflux cu eticheta cea mai mare pentru fluxul maxim, pentru care se recomandă cititorului monografiile precizate la bibliografie ■

**Exemplul 3.6.** Ilustrăm algoritmul preflux cu eticheta cea mai mare pentru fluxul minim pe rețeaua din figura 3.7(a). După inițializările din liniile (8) - (15) se obține rețeaua reziduală din figura 3.7(b), iar în coada  $L$  se găsesc nodurile 2 cu prioritatea 1 și 3 cu prioritatea 1. Algoritmul scoate nodul 2 din coada  $L$  și îl examinează, adică retrage  $g = \min\{-e(2), \hat{r}(1, 2)\} = 3$  unități pe arcul  $(1, 2)$ , apoi reetichetează nodul 2:  $d(2) = 2$  și-l adaugă în coada  $L$  cu prioritatea 2. Rețeaua reziduală obținută după examinarea nodului 2 este reprezentată în figura 3.7(c), iar coada  $L$  va conține nodul 2 cu prioritatea 2 și nodul 3 cu prioritatea 1. La următoarea iterare, algoritmul va scoate din nou nodul 2 din coadă și-l va examina, ceea ce înseamnă că va retrage 2 unități de flux pe arcul  $(3, 2)$  și apoi va efectua o reetichetare:  $d(2) = 5$  și va adăuga nodul 2 cu prioritatea 5 în coada  $L$ . Deci coada  $L$  va conține nodul 2 cu prioritatea 5 și nodul 3 cu prioritatea 1, iar rețeaua reziduală este cea din figura 3.7(d). Din nou nodul 2 este scos din coadă și examinat. Se retrage o unitate de flux pe arcul  $(4, 2)$ , obținându-se rețeaua din figura 3.7(e). La următoarea iterare singurul nod din coada  $L$ , nodul 3, va fi scos și examinat. Se retrag 6 unități de flux pe arcul  $(1, 3)$  și algoritmul se termină, coada  $L$  rămânând vidă. Rețeaua reziduală obținută la terminarea algoritmului este ilustrată în figura 3.7(f), iar fluxul minim este reprezentat în figura 3.7(g).

### Algoritmul de scalare a deficitului

Acest algoritm este o variantă îmbunătățită a algoritmului preflux generic pentru fluxul minim. În algoritmul generic cu prefluxuri cele mai mari consumatoare de timp sunt retragerile incomplete de flux ( $O(n^2 m)$ ). În algoritmul de scalare a deficitului, vom reduce numărul acestora la  $O(n^2 \log \bar{c})$  folosind tehnica scalării, adică impunând

condiția ca să se efectueze doar retrageri incomplete de cantități suficient de mari de flux.

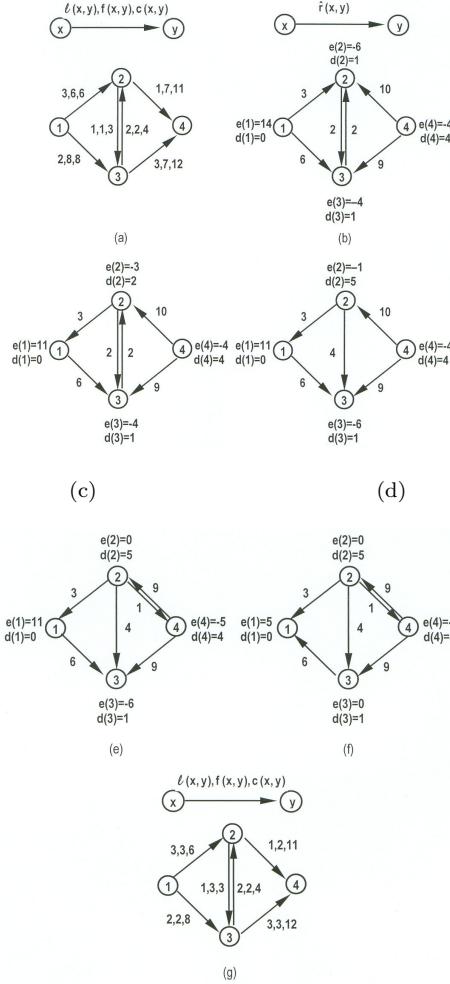


Fig.3.7

Fie  $\bar{r}$  o margine superioară pentru  $e_{\max} = \max\{|e(x)| \mid x \text{ este nod activ}\}$ . Spunem că un nod  $y$  are un *deficit mare* dacă  $e(y) \leq -\bar{r}/2$ ; în caz contrar nodul  $y$  are un *deficit mic*.

Algoritmul de scalare a deficitului retrage întotdeauna flux de la un nod cu deficit mare pentru ca 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 rețeaua  $G$ ;
- (4)   se determină rețeaua reziduală  $\hat{G}(f)$ ;
- (5)   se calculează etichetele distanță exakte  $d(\cdot)$  în rețeaua 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)            $\bar{r} := 2^{\lceil \log \bar{c} \rceil};$ 
(12)           WHILE  $\bar{r} \geq 1$  DO
(13)               BEGIN
(14)                   WHILE în rețeaua reziduală  $\hat{G}(f)$  există un nod activ cu deficit
                     mare DO
(15)                   BEGIN
(16)                       se selectează nodul  $y$  astfel încât
                            $d(y) = \min\{d(z) \mid z \text{ nod cu deficit mare}\};$ 
(17)                       RETRAGERE/REETICHETARE( $y$ );
(18)                   END;
(19)                    $\bar{r} := \bar{r}/2;$ 
(20)               END;
(21)           END;
(22)       END.

(1) PROCEDURA RETRAGERE/REETICHETARE( $y$ );
(2) BEGIN
(3)     IF în rețeaua reziduală  $\hat{G}(f)$  există un arc admisibil  $(x, y)$ 
(4)     THEN se retrag  $g = \min(-e(y), \hat{r}(x, y), \bar{r} + e(x) > 0)$  unități 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  $\bar{r}$  rămâne constant. O fază de scalare pentru o valoare dată a lui  $\bar{r}$  se numește *fază de  $\bar{r}$ -scalare*.

**Exemplul 3.7.** Ilustrăm algoritmul de scalare a deficitului pe rețeaua din figura 3.8(a). După inițializările din liniile (4) - (10) se obține rețeaua reziduală din figura 3.8(b) și  $\bar{r} = 8$ , iar nodurile cu deficit mare sunt 2 și 3. Dacă algoritmul selectează nodul 2 atunci se retrag  $g = \min\{-e(2), \hat{r}(1, 2), \bar{r} + e(1)\} = \min\{6, 3, 22\} = 3$  unități de flux pe arcul admisibil  $(1, 2)$  și se obține rețeaua din figura 3.8(c). Apoi se selectează singurul nod cu deficit mare, nodul 3, se retrag 4 unități de flux pe arcul admisibil  $(1, 3)$  și se obține rețeaua reprezentată în figura 3.8(d), în care nu mai există noduri cu deficite mari. Deci, se înjumătățește valoarea lui  $\bar{r}$  și se începe o nouă fază de scalare pentru  $\bar{r} = 4$ . Singurul nod cu deficit mare este nodul 2 pe care algoritmul îl va selecta și îl va reeticheta:  $d(2) = 2$ , creând astfel arcul admisibil  $(3, 2)$ . La următoarea iterație se selectează din nou nodul 2, se retrag 2 unități de flux pe arcul  $(3, 2)$  și se obține rețeaua din figura 3.8(e). După această retragere de flux nodul 2 va avea deficit mic, iar nodul 3 va avea deficit mare, deci va fi selectat și se vor retrage 2 unități de flux pe arcul

admisibil  $(1,3)$ . În figura 3.8(f) este reprezentată rețeaua reziduală astfel obținută, în care nu mai există niciun nod cu deficit mare. Deci, se înjumătăște valoarea lui  $\bar{r}$  și se începe o nouă fază de scalare pentru  $\bar{r} = 2$ . Algoritmul va selecta nodul 2 care este singurul nod cu deficit mare și-l va reeticheta:  $d(2) = 5$ . Apoi se va selecta din nou nodul 2 și se va retrage o unitate de flux pe arcul admisibil  $(4,2)$  obținându-se rețeaua reziduală reprezentată în figura 3.8(g) în care nu mai există noduri cu deficit mare. Se efectuează  $\bar{r} = 1$  și, deoarece nu mai există noduri active algoritmul se termină. În figura 3.8(h) este reprezentat fluxul minim determinat de algoritmul de scalare a deficitului.

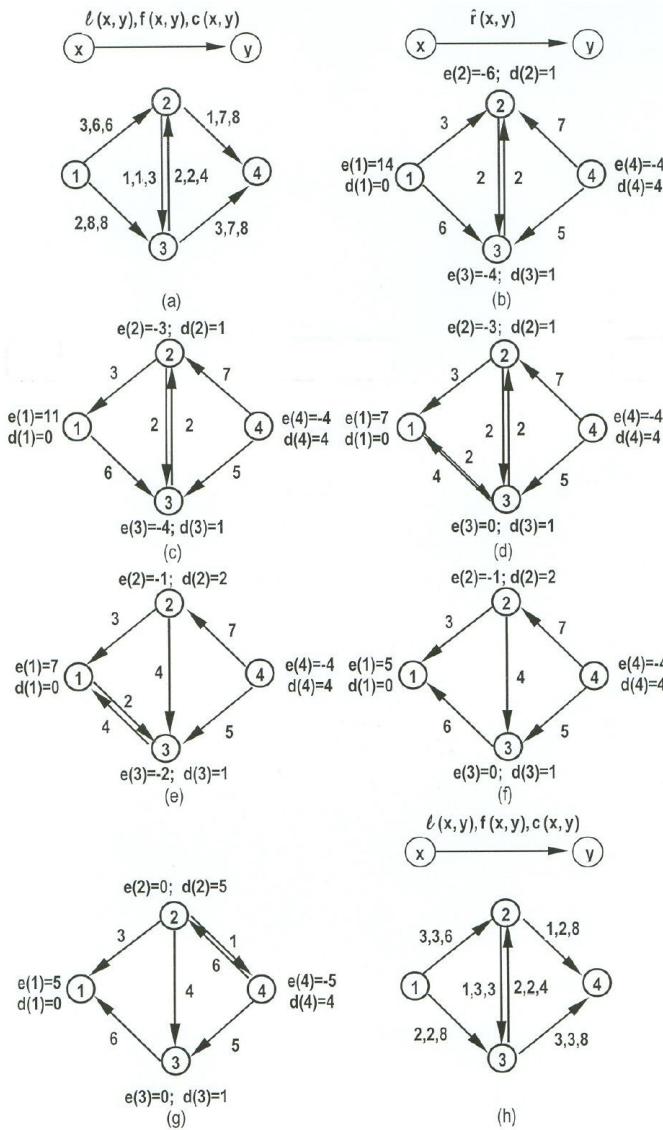


Fig.3.8

**Teorema 3.18.** Dacă există un flux admisibil în rețeaua  $G$ , atunci algoritmul de scalare a deficitului determină un flux minim în  $G$ .

*Demonstrație.* La începutul algoritmului  $\bar{r} := 2^{\lceil \log \bar{c} \rceil}$ , deci  $\bar{c} \leq \bar{r} < 2\bar{c}$ . Pe parcursul fazei de  $\bar{r}$ -scalare,  $e_{\max}$  poate să crească și să descrească, dar trebuie să fie îndeplinite condițiile  $\bar{r}/2 \leq e_{\max} \leq \bar{r}$ . Când  $e_{\max} < \bar{r}/2$  atunci se înjumătățește valoarea lui  $\bar{r}$  și se începe o nouă fază de scalare. După  $1 + \lceil \log \bar{c} \rceil$  faze de scalare,  $e_{\max}$  devine nul și fluxul astfel obținut este un flux minim. ■

**Teorema 3.19.** Pe parcursul fiecărei faze de  $\bar{r}$ -scalare sunt îndeplinite următoarele două condiții:

- (a) la fiecare retragere incompletă de flux se retrag cel puțin  $\bar{r}/2$  unități de flux
- (b)  $e_{\max} \leq \bar{r}$ .

*Demonstrație.* (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 distanță dintre nodurile cu deficit mare. Deci,  $e(y) \leq -\bar{r}/2$  și  $e(x) > -\bar{r}/2$ . Deoarece pe arcul  $(x, y)$  se efectuează o retragere incompletă de flux, rezultă că se micșorează fluxul cu  $\min\{-e(y), \bar{r} + e(x)\} \geq \bar{r}/2$  unități.

(b) O retragere de flux pe arcul  $(x, y)$  mărește doar valoarea absolută a deficitului nodului  $x$ . Noul deficit al nodului  $x$  este  $e'(x) = e(x) - \min\{-e(y), r(x, y), \bar{r} + e(x)\} \geq e(x) - (\bar{r} + e(x)) = -\bar{r}$ . Deci,  $e'(x) \geq -\bar{r}$  și  $e_{\max} \leq \bar{r}$ . ■

**Teorema 3.20.** Pe parcursul fiecărei faze de scalare, algoritmul de scalare a deficitului efectuează  $O(n^2)$  retrageri incomplete de flux.

*Demonstrație.* Considerăm funcția potențial  $P = -\sum_{x \in N} e(x)d(x)/\bar{r}$ . Valoarea inițială a lui  $P$  la începutul fazei de  $\bar{r}$ -scalare este mărginită superior de  $2n^2$  deoarece  $e(x) \geq -\bar{r}$  și  $d(x) \leq 2n$ ,  $x \in N$ , conform Teoremei 3.19.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 distanță a nodului  $y$  crește cu  $q \geq 1$  unități, ceea ce duce la creșterea lui  $P$  cu cel mult  $q$  unități pentru că  $e(y) \geq -\bar{r}$ . Deoarece, pentru fiecare nod  $y$ ,  $d(y)$  poate crește cu cel mult  $2n$  unități pe parcursul execuției algoritmului, rezultă că valoarea lui  $P$  crește cu cel mult  $2n^2$  unități 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  $P$  descrește. După o retragere incompletă de flux pe arcul  $(x, y)$ , fluxul de la nodul  $x$  la nodul  $y$  scade, conform Teoremei 3.19.a, cu cel puțin  $\bar{r}/2$  unități, deci  $P$  descrește cu cel puțin  $1/2$  pentru că  $d(y) = d(x) + 1$ . Deoarece valoarea inițială a lui  $P$  la începutul fazei de scalare era cel mult  $2n^2$  și putea să crească cu încă cel mult  $2n^2$  unități, 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 3.21** Complexitatea algoritmului de scalare a deficitului este  $O(nm + n^2 \log \bar{c})$ .

*Demonstrație.* Numărul total de retrageri incomplete de flux este  $O(n^2 \log \bar{c})$ , conform Teoremei 3.20, deoarece algoritmul efectuează  $O(\log \bar{c})$  faze de scalare. Celelalte operații (retrageri complete, reetichetări de noduri și determinări de arce admisibile) necesită

$O(nm)$  timp, ca și în cazul algoritmului preflux generic pentru fluxul minim. Deci, complexitatea algoritmului de scalare a deficitului este  $O(nm + n^2 \log \bar{c})$ . ■

### 3.4 Algoritmul minimax

Putem rezolva problema fluxului minim de la nodul sursă  $s$  la nodul stoc  $t$  în rețeaua  $G$  aplicând un algoritm de flux maxim de la  $t$  la  $s$  în rețeaua reziduală. Această abordare se bazează pe următoarea idee: obiectivul în cazul problemei fluxului minim este de a trimite cât mai puțin 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 rețeaua 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 rețeaua  $G$ .

Algoritmul minimax este următorul:

- (1) PROGRAM MINIMAX;
- (2) BEGIN
- (3)     fie  $f$  un flux admisibil în  $G$ ;
- (4)     se determină rețeaua reziduală  $\tilde{G}(f)$ ;
- (5)     se determină un flux maxim  $f^*$  de la  $t$  la  $s$  utilizând  $\tilde{G}(f)$ ;
- (6)      $f^*$  este un flux minim de la  $s$  la  $t$  în  $G$ ;
- (7) END.

**Exemplul 3.8.** Ilustrăm algoritmul minimax pe rețeaua din figura 3.9(a) în care  $s = 1$  și  $t = 4$ . În figura 3.9(b) este reprezentată rețeaua reziduală  $\tilde{G}(f)$  corespunzătoare fluxului admisibil ilustrat în figura 3.9(a). În figura 3.9(c) sunt reprezentate capacitatele reziduale corespunzătoare unui flux maxim de la nodul 4 la nodul 1, care pot fi obținute aplicând orice algoritm pentru fluxul maxim. Pornind de la aceste capacitați reziduale și folosind relațiile

$$f(x, y) = \ell(x, y) + \max(0, c(x, y) - r(x, y) - \ell(x, y))$$

și

$$f(y, x) = \ell(y, x) + \max(0, c(y, x) - r(y, x) - \ell(y, x))$$

obținem fluxul din figura 3.9(d) care este un flux minim de la nodul 1 la nodul 4.

**Teorema 3.22.** *Algoritmul minimax determină un flux minim de la nodul  $s$  la nodul  $t$  în rețeaua  $G = (N, A, \ell, c, s, t)$ .*

*Demonstrație.* Fie  $f^*$  fluxul obținut la sfârșitul execuției algoritmului. Deci,  $f^*$  este un flux maxim de la nodul  $t$  la nodul  $s$ . Conform Teoremei 1.11, rezultă că în rețeaua reziduală  $\tilde{G}(f)$  nu există drum de la nodul  $t$  la nodul  $s$ . Deci, există o tăietură  $t - s$   $[X, \bar{X}]$  astfel încât

$$r(x, y) = 0, \quad (x, y) \in (X, \bar{X}).$$

Sau echivalent,

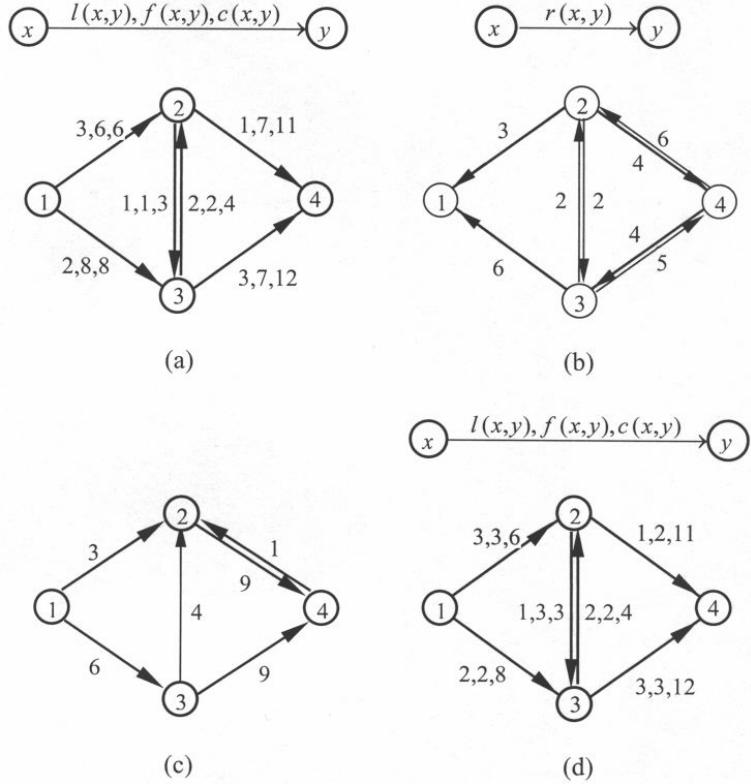


Fig.3.9

$$c(x, y) - f(x, y) + f(y, x) - \ell(y, x) = 0, \quad (x, y) \in (X, \bar{X}).$$

Dar  $f(x, y) \leq c(x, y)$ ,  $(x, y) \in A$  și  $f(y, x) \geq \ell(y, x)$ ,  $(y, x) \in A$ .

Deci,

$$f(x, y) = c(x, y), \quad (x, y) \in (X, \bar{X}), \quad f(y, x) = \ell(y, x), \quad (y, x) \in (\bar{X}, X)$$

Sau echivalent,

$$c(x, y) - f(x, y) + f(y, x) - \ell(y, x) = 0, \quad (y, x) \in (\bar{X}, X)$$

adică

$$\hat{r}(y, x) = 0, \quad (y, x) \in (\bar{X}, X). \quad (3.7)$$

Deoarece  $[X, \bar{X}]$  este o căietură  $t - s$ , rezultă că  $[\bar{X}, X]$  este o căietură  $s - t$ . Din relația (3.7), rezultă că în rețeaua reziduală  $\hat{G}^*(f^*)$  definită pentru problema fluxului minim nu există drum de la nodul  $s$  la nodul  $t$ . Deci, conform Teoremei 3.3,  $f^*$  este un flux minim de la nodul  $s$  la nodul  $t$ .

**Teorema 3.23.** *Complexitatea algoritmului minimax este egală cu complexitatea algoritmului folosit pentru determinarea fluxului maxim de la nodul stoc la nodul sursă.*

*Demonstrație.* Evident. ■

## 3.5 Aplicații

### 3.5.1 Problema planificării lucrărilor

La un atelier trebuie efectuate într-o singură zi  $p$  lucrări. Se cunosc timpii  $\tau(i)$  și  $\tau'(i)$  de începere și de terminare a fiecărei lucrări  $i$ ,  $i = 1, \dots, p$ . 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 și care nu poate lucra în același timp la mai multe lucrări.

Problema planificării lucrărilor se poate modela ca o problemă de flux minim într-un rețea  $G = (N, A, l, c)$ , unde:  $N = N_1 \cup N_2 \cup N_3 \cup N_4$ ,  $N_1 = \{s\}$ ,  $N_2 = \{i \mid i = 1, \dots, p\}$ ,  $N_3 = \{i' \mid i' = 1, \dots, p\}$ ,  $N_4 = \{t\}$ ,  $A = A_1 \cup A_2 \cup A_3 \cup A_4$ ,  $A_1 = \{(s, i) \mid i \in N_2\}$ ,  $A_2 = \{(i, i') \mid i, i' = 1, \dots, p\}$ ,  $A_3 = \{(i', j) \mid \tau'(i') + \tau_2(i', j) \leq \tau(j)\}$ ,  $A_4 = \{(i', t), \mid i' \in N_3\}$ ,  $l(s, i) = 0$ ,  $c(s, i) = 1$ ,  $(s, i) \in A_1$ ,  $l(i, i') = 1$ ,  $c(i, i') = 1$ ,  $(i, i') \in A_2$ ,  $l(i', j) = 0$ ,  $c(i', j) = 1$ ,  $(i', j) \in A_3$ ,  $l(i', t) = 0$ ,  $c(i', t) = 1$ ,  $(i', t) \in A_4$ . Deoarece  $i$  și  $i'$  reprezintă aceeași lucrare avem  $\tau(i') = \tau(i)$  și  $\tau'(i') = \tau'(i)$ .

**Exemplul 3.9** 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

Rețeaua  $G = (N, A, l, c)$  este prezentată în figura 3.10.

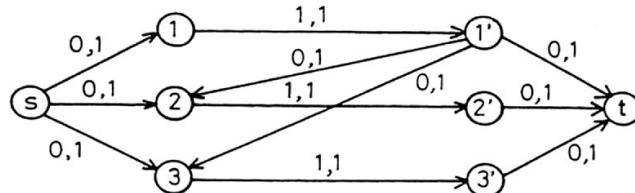


Fig.3.10

Un flux minim este reprezentat pe rețeaua din figura 3.11.

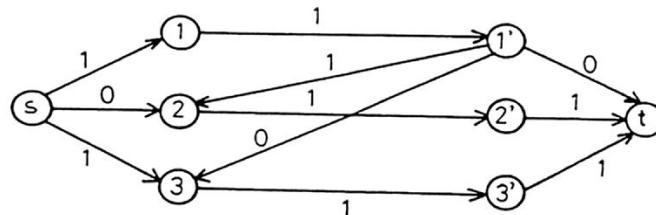


Fig.3.11

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. Soluția nu este unică. Cititorul poate să determine o a doua soluție.

# Bibliografie

- [1] Ahuja, R. K., Magnanti, T. L. and Orlin, J. B. (1993): *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, Englewood, Cliffs, N. J.
- [2] Bang-Jensen, J. and Gutin, G. (2001): *Digraph: Theory, Algorithms and Applications*. Springer-Verlag, London.
- [3] Bazaraa, M. S., Jarvis J. J. and Sherali, H. D. (2005): *Linear Programming and Network Flows (third edition)*. Wiley, New York.
- [4] Berge, C. (1973): *Graphs and Hypergraphs*. North Holland, Amsterdam.
- [5] Bertsekas, D. P. (1991): *Liniar Network Optimization: Algorithms and Codes*. The MIT Press Cambridge, Massachusetts.
- [6] Chen, W. K. (1990): *Theory of Nets: Flows in Networks*. Wiley, New York.
- [7] Christofides, N. (1975): *Graph Theory: An Algorithmic Approach*. Academic Press, New York.
- [8] Ciupală L. (2007) *Algoritmi fundamentali din teoria grafurilor. Aplicații*. Editura Universității Transilvania Brașov.
- [9] Ciurea, E (2001): *Introducere în algoritmica grafurilor*. Editura Tehnică București.
- [10] Ciurea, E., Ciupală, L. (2006): *Algoritmi. Introducere în algoritmica fluxurilor în rețele*. Editura Matrix Rom, București.
- [11] Croitoru, C. (1992): *Tehnici de bază în optimizarea combinatorie*. Editura Universității 'Al. I. Cuza', Iași.
- [12] Evans, J. R. and Minieka, E. (1992): *Optimization Algorithms for Networks and Graphs*. Marcel Dekker, Inc., New York.
- [13] Ford, L. R. and Fulkerson, D. R. (1962): *Flows in Networks*. Princeton University Press, Princeton, N. J.
- [14] Gibbons, A. (it Algorithmic Graphs, Networks and Algorithms). Springer, Berlin, 1999.
- [15] Glover, F., Klingman, D. and Philips, N. V. (1992): *Network Models in Optimization and their Applications in Practice*. Wiley, New York.
- [16] Godran, M. and Minoux, N. *Graphs and Algorithms*. Wiley, New York, 1984.
- [17] Gross, J. and Yellen, J. (1999): *Graph Theory and its Applications*, CRC Press, New York.
- [18] Jungnickel, D. (1999): *Graphs, Networks and Algorithms*. Springer, Berlin.
- [19] Rühe, G. (1991): *Algorithmic Aspects of Flows in Networks*. Kluwer Academic Publishers, Dordrecht.
- [20] Tarjan, R. E. (1983): *Data Structures and Network Algorithms*. SIAM, Philadelphia.

- [21] **Toadere, T.** (2002): *Grafe. Teorie, algoritmi și aplicații*. Editura Albatros, Cluj-Napoca.
- [22] **Tomescu, I.** (1981): *Probleme de combinatorică și teoria grafurilor*. Editura Didactică și Pedagogică, București.
- [23] **Tutte, W. T.** (1984): *Graph Theory*. Cambridge University Press, Cambridge.