Cuplaj MAXIM de valoare MINIMĂ într-un GRAF BIPARTIT

Ilie Vieru

În cadrul acestui articol vom prezenta un algoritm interesant pentru determinarea cuplajului maxim de cost minim într-un graf bipartit. În momentul de față algoritmul este sub tipar și la prestigioasa revistă Mathematics Magazine.

Sunt cunoscute problemele în care se solicită asocierea optimă a elementelor unei mulțimi $A = \{a_1, a_2, ..., a_n\}$ cu elementele mulțimii $B = \{b_1, b_2, ..., b_m\}$, în condițiile unor limitări ale posibilităților de asociere.

În general, fiecare asociere posibilă $a_i \leftrightarrow b_j$ aduce un anumit efect (cost, profit etc.) pe care îl presupunem cunoscut.

Limitările asupra asocierilor înseamnă:

- un element a_i din A poate fi asociat doar cu anumite elemente din B și reciproc;
- în final, fiecărui element din A i s-a asociat cel mult un element din B și reciproc.

Asocierea optimă înseamnă găsirea unui cuplaj maxim într-un graf bipartit și presupune, de obicei, două obiective:

- să se determine maximul de asocieri;
- suma efectelor asocierilor să fie optimă (maximă sau minimă).

Dintre problemele practice care se reduc la a determina un cuplaj maxim de valoare optimă, amintim [1]:

- repartizarea muncitorilor unei secții pe utilaje în funcție de pregătirea și preferințele muncitorilor dar și de complexitatea mașinilor;
- repartizarea angajaților pe posturi;
- formarea grupelor de lucru după afinitățile dintre membrii colectivului;
- transferarea unor informații într-un grup.

Observație

Presupunem cunoscute definițiile cu privire la *graf bipartit*, *cuplaj*, *cuplaj maxim*, *valoarea cuplajului* ([1], [2], [4]).

În anul 1931 *König* a formulat teorema prezentată în continuare.

Teoremă

Numărul maxim de muchii ale unui cuplaj într-un graf bipartit $G = (A \cup B, U)$ este $\min_{C \subset A}(|A - C| + |U(C)|)$.

În anul 1955, bazându-se pe teorema lui *König*, *H.W. Kuhn* a elaborat un algoritm cunoscut sub denumirea de ***algoritmul ungar*** cu ajutorul căruia se poate determina un cuplaj maxim de valoare minimă într-un graf bipartit pentru $|A| = |B| = n$. El se bazează pe observația prezentată în continuare.

Observație

Dacă se adună (sau se scade) același număr la toate valorile muchiilor, atunci nu se schimbă ierarhia cuplajelor maxime.

Algoritmul pornește de la matricea pătratică $M = (m_{ij})_{1 \le i,j \le n}$, unde:

$$m_{ij} = \begin{cases} \text{valoarea muchiei } [i, j], \\ \quad \text{dacă există muchia } [i, j]; \\ \infty, \text{dacă nu există muchia } [i, j]. \end{cases}$$

Acest algoritm, este laborios, greu de implementat, deși are complexitate polinomială ([1]).

Pentru determinarea cuplajului bipartit maxim se poate folosi și metoda *Ford-Fulkerson* pentru graful bipartit $G = (X = A \cup B, U)$, într-un timp polinomial în $|X|$ și $|U|$.

Soluția constă în construirea unei rețele de transport în care fluxurile reprezintă cuplajele.

Graful inițial se completează cu încă două vârfuri s (sursa) și t (desti-

nația), legate de vârfurile din A, respectiv B ca în exemplul de la p. 517 din [2].

Se presupune că fiecare muchie are o capacitate pozitivă. În acest context se aplică algoritmul *Ford-Fulkerson* obținându-se soluția optimă.

În continuare vă propun un algoritm original, rapid și foarte ușor de implementat. Pentru aceasta pornim de la câteva elemente de algebră superioară. Se știe că pentru a calcula valoarea determinantului asociat unei matrice $A = (a_{ij})_{1 \le i,j \le n}$ avem formula:

$$|A| \overset{def}{=} \sum_{\sigma \in S_n} (-1)^{n(\sigma)} a_{1,\sigma(1)} a_{2,\sigma(2)} \cdots a_{n,\sigma(n)},$$

unde $(-1)^{n(\sigma)}$ reprezintă *signatura* permutării σ, iar fiecare din termenii sumei reprezintă câte un produs cu n elemente ale matricei A, în așa fel încât ele epuizează toate liniile și toate coloanele matricei.

Cu alte cuvinte, nu există în nici un termen al sumei doi factori aflați pe aceeași linie sau aceeași coloană a matricei.

Presupunem mai întâi că avem $|A| = |B| = n$. În acest caz va trebui să determinăm într-un mod asemănător definiției prezentate anterior termenul $S_n \overset{def}{=} a_{1,\sigma(1)} a_{2,\sigma(2)} \cdots a_{n,\sigma(n)}$ de valoare minimă, reținând în același timp și permutarea care l-a generat.

Introducem notația:

$$A_k = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{pmatrix}.$$

Astfel putem scrie expresia:

$$A_{k+1} = \begin{pmatrix} A_k & | & a_{1,k+1} \\ - & \backslash & - \\ a_{k+1,1} & | & a_{k+1,k+1} \end{pmatrix}.$$

Cu alte cuvinte, A_{k+1} se obține din A_k prin bordare la dreapta și jos cu elementele corespunzătoare coloanei $k + 1$, respectiv ale liniei $k + 1$.

Atașăm matricei A_n vectorul de decizie $d = (d_i)_{1 \le i \le n}$, cu semnificația: $d_i = j_0$ unde a_{ij_0} reprezintă termenul de pe linia i ales la un pas k ($1 \le i, j_0 \le k$) în suma S_n, $i \le k \le n$. De asemenea, construim vectorul $v = (v_i)_{1 \le i \le n}$, care memorează permutarea inversă atașată permutării d, adică $v_{d_i} = i$, sau $d_{v_j} = j$ ($1 \le i, j_0 \le k$).

Iterativ construim:

$A_1 = (a_{11})$; $d_1 = 1$; $S_1 = a_{11}$; $v_1 = 1$.

Presupunem construite cele patru șiruri până în pasul k, $k \ge 1$.

Pentru pasul $k + 1$ întreaga construcție urmărește:

- obținerea elementului S_{k+1} cu valoare minimă, în sensul definiției, cu termeni din matricea A_{k+1} folosind rezultatele obținute până la pasul k;
- actualizarea vectorilor $(d_1, d_2, ..., d_k)$, respectiv $(v_1, v_2, ..., v_k)$.

La acest pas se disting trei situații:

- la S_k se adaugă $a_{k+1,k+1}$, caz în care efectuăm următoarele operații: $S_{k+1} = S_k + a_{k+1,k+1}$; $d_{k+1} = 1$; $v_{k+1} = 1$.
- S_{k+1} se formează adăugând termenii $a_{i,k+1}$ și $a_{k+1,j}$ pentru care avem $d_i = j$, caz în care efectuăm operațiile: $S_{k+1} = S_k - a_{ij} + a_{i,k+1} + a_{k+1,j}$; $d_i = k + 1$; $d_{k+1} = j$; $v_{k+1} = i$; $v_j = k + 1$.
- S_{k+1} se formează adăugând termenii $a_{i,k+1}$ și $a_{k+1,j}$ pentru care avem $d_i \ne j$, caz în care se efectuează operațiile: $S_{k+1} = S_k - a_{i,d_i} - a_{v_j j} + a_{v_j d_i} + a_{i,k+1} + a_{k+1,j}$; $d_{v_j} = d_i$; $v_{d_i} = v_j$; $d_i = k + 1$; $v_{k+1} = i$; $d_{k+1} = j$; $v_j = k + 1$.

În final S_n reprezintă valoarea minimă a cuplajului maxim: $(1, d_1)$, ..., (n, d_n).

Pentru $n = |A| < |B| = m$, algoritmul parcurge următorii pași:

- se aplică algoritmul anterior și se determină S_n, d_n, v_n;
- pentru valorile j cuprinse între $n + 1$ și m avem $l_{i_0} = \min_{1 \le i \le n} \{a_{i,j} - a_{i,d_i}\}$ dacă $l_{i_0} < 0$, atunci se efectuează următoarele actualizări $S_n = S_n + l_{i_0}$; $d_{i_0} = j$.

Observație

Soluția se exprimă la fel ca în cazul algoritmului prezentat anterior (când mulțimile A și B au același număr de elemente).

Pentru $n = |B| < |A| = m$, algoritmul parcurge următorii pași:

- se aplică algoritmul anterior și se determină S_n, d_n, v_n;
- pentru valorile i cuprinse între $n + 1$ și m avem $l_{j_0} = \min_{1 \le i \le n} \{a_{i,j} - a_{v_j,j}\}$ dacă $l_{j_0} < 0$, atunci se efectuează următoarele actualizări $S_n = S_n + l_{j_0}$; $d_{v_{j_0}} = 0$; $v_{j_0} = i$; $d_i = j_0$.

Observație

În acest caz, la afișarea vectorului de decizii ([3]), se poate parcurge șirul pe coloane: $(v_1, 1)$, ..., (v_n, n), sau se selectează acele linii, crescător după i, pentru care $d_i > 0$.

Algoritmul urmărește la fiecare pas k optimizarea funcției obiectiv și de aceea putem spune că se încadrează în strategia generală *Greedy*.

Urmărind etapele prezentate, se poate demonstra, folosind cunoștințe de nivelul clasei a XI-a teorema care urmează.

Teoremă

Algoritmul prezentat determină un cuplaj maxim cu valoare minimă.

Exemplu

La o secție au fost angajați șase lucrători pentru a lucra pe șase mașini.

În urma unor probe de lucru s-au determinat numerele a_{ij} care reprezintă riscul pe care și-l asumă patronul angajând lucrătorul i pentru a lucra pe mașina j.

Rezultatele au fost înregistrate în matricea următoare:

$$A_k = \begin{pmatrix} 17 & 43 & 27 & 14 & 39 & 52 \\ 29 & 24 & 69 & 90 & 23 & 13 \\ 18 & 90 & 62 & 12 & 16 & 70 \\ 58 & 14 & 6 & 18 & 73 & 64 \\ 15 & 41 & 38 & 36 & 40 & 60 \\ 25 & 44 & 18 & 44 & 13 & 50 \end{pmatrix}.$$

Ne propunem să determinăm repartizarea optimă a angajaților pe cele șase mașini astfel încât riscul total să fie minim.

Pentru aceasta vom aplica algoritmul prezentat, arătând valorile obținute la fiecare pas.

Pasul #1

- $A_1 = (17)$;
- $S_1 = 17$;
- $d_1 = 1$;
- $v_1 = 1$.

Pasul #2

- $A_2 = \begin{pmatrix} 17 & 43 \\ 29 & 24 \end{pmatrix}$;
- $S_2 = S_1 + a_{22} = 41$;
- $d_2 = 2$;
- $v_2 = 2$.

Pasul #3

- $A_3 = \begin{pmatrix} 17 & 43 & 27 \\ 29 & 24 & 69 \\ 18 & 90 & 62 \end{pmatrix}$;
- $S_3 = S_2 + a_{13} + a_{31} - a_{11} = 69$;
- $d_1 = 3$;
- $v_1 = 3$;
- $v_3 = 1$;
- $d_3 = 1$.

Pasul #4

- $A_4 = \begin{pmatrix} 17 & 43 & 27 & 14 \\ 29 & 24 & 69 & 90 \\ 18 & 90 & 62 & 12 \\ 58 & 14 & 6 & 18 \end{pmatrix}$;
- $S_4 = S_3 + a_{14} + a_{43} - a_{13} = 62$;
- $d_1 = 4$;
- $v_4 = 1$;
- $d_4 = 3$;
- $v_3 = 4$.

Pasul #5

- $A_5 = \begin{pmatrix} 17 & 43 & 27 & 14 & 39 \\ 29 & 24 & 69 & 90 & 23 \\ 18 & 90 & 62 & 12 & 16 \\ 58 & 14 & 6 & 18 & 79 \\ 15 & 41 & 38 & 36 & 40 \end{pmatrix}$;
- $S_5 = S_4 + a_{14} + a_{43} - a_{13} = 75$;
- $d_3 = 5$;
- $v_5 = 3$;
- $d_5 = 1$;
- $v_1 = 5$.

Pasul #6

- $A_6 = \begin{pmatrix} 17 & 43 & 27 & 14 & 39 & 52 \\ 29 & 24 & 69 & 90 & 23 & 13 \\ 18 & 90 & 62 & 12 & 16 & 70 \\ 58 & 14 & 6 & 18 & 79 & 64 \\ 15 & 41 & 38 & 36 & 40 & 60 \\ 25 & 44 & 18 & 44 & 13 & 50 \end{pmatrix}$;
- $S_6 = S_5 + a_{26} + a_{63} - a_{22} - a_{43} + a_{42} = 90$;
- $d_2 = 6$;
- $v_6 = 2$;
- $d_4 = 2$;
- $v_2 = 4$;
- $d_6 = 3$;
- $v_3 = 6$.

Rezultatul

În final vom obține:

$$A = \begin{pmatrix} 17 & 43 & 27 & 14 & 39 & 52 \\ 29 & 24 & 69 & 90 & 23 & 13 \\ 18 & 90 & 62 & 12 & 16 & 70 \\ 58 & 14 & 6 & 18 & 79 & 64 \\ 15 & 41 & 38 & 36 & 40 & 60 \\ 25 & 44 & 18 & 44 & 13 & 50 \end{pmatrix}.$$

Așadar, ordinea lucrătorilor pe mașini va fi: (1, 4), (2, 6), (3, 5), (4, 2), (5, 1),(6, 3).

Aplicație

Acest algoritm poate fi utilizat pentru a rezolva problema ***Culori***, propusă spre rezolvare la runda #04 a ediției 2004/2005 a concursului de programare *Bursele Agora*.

Bibliografie:

1. E. Tigănescu, D. Mitruț, *Bazele cercetării operaționale*, Editura Academiei de Studii Economice, București, 1999;
2. T. H. Cormen, C. E. Leiserson, R. R. Rivest, *Introducere în algoritmi*, Editura Computer Libris Agora, Cluj-Napoca, 2000;
3. I. Vieru, *Rolul deciziei în programarea dinamică*, GInfo 10/3, Editura Agora Media, Târgu Mureș, 2000;
4. I. Tomescu, *Combinatorică și teoria grafurilor*, Editura Universității București, 1978.

Exemplu de implementare:

```
void calcul() {
 d[1] = 1;
 v[1] = 1;
 s = a[1][1];

 for (k = 2; k <= n; k++) {
  li0 = a[1][k] +
     a[k][d[1]] - a[1][d[1]];
  i0 = 1;
  c0 = d[i0]
  j  = d[i0];
  for (i = 1; i < k; i++) {
   j = d[i];
   R = a[i][k] + a[k][j]
                   - a[i][j];
   if (R < li0) {
    li0 = R;
    i0  = i;
    j   = d[i];
    c0  = j
    j0  = j;
   }
   for (int c = 1;
                  c < k; c++)
    if (c - j) {
     int u = v[c];
     R = a[i][k] + a[k][c] -
          a[i][j] - a[u][c] +
                    a[u][j];
     if (R < li0) {
      li0 = R;
      c0  = c;
      i0  = i;
      u0  = u;
      j0  = j;
     }
    }
  }

  if (li0 >= a[k][k]) {
   s += a[k][k];
   d[k] = k;
   v[k] = k;
  }
  else {
    s += li0;
    if (c0 == j0) {
     d[i0] = k;
     d[k]  = c0;
     v[c0] = k;
     v[k]  = i0;
    }
    else {
     d[i0] = k;
     v[k]  = i0;
     d[u0] = j0;
     v[j0] = u0;
     d[k]  = c0;
     v[c0] = k;
    }
  }
  }

  printf(" S= %d\n",s);
  for (i = 1; i <= n; i++)
    printf("%d ",
                 a[i][d[i]]);
  printf("\n");
}
```