Algoritmul lui Kuhn

Tiberiu-Lucian Florea

November 21, 2005

1 Introducere

Voi prezenta în continuare problema afectării împreună cu una din cele mai eficiente soluţii pentru rezolvarea ei: *Algoritmul lui Kuhn*. În ultima parte a articolului, voi oferi un exemplu de implementare uşor de folosit în concursurile de informatică. Se presupun cunoscute noţiunile de "cuplaj maxim" şi "suport minim" într-un graf bipartit, precum şi modul de obţinere a unui cuplaj maxim într-un graf bipartit cu ajutorul drumurilor de creştere. Să începem cu definiţia problemei discutate.

Problema de afectare Exista m muncitori şi n lucrări, fiecare muncitor putând efectua una sau mai multe lucrări. Notăm muncitorii cu $x_1, x_2, \ldots, x_m$, lucrările cu $y_1, y_2, \ldots, y_n$, şi asociem costul $v_{ij} \geq 0$ efectuării de către muncitorul x_i a lucrării y_j, faptul că $v_{ij} = \infty$ având semnificaţia că muncitorul x_i nu poate efectua lucrarea y_j. Să se găsească o repartizare a celor m muncitori la cele n lucrări astfel încât un muncitor să efectueze cel mult o lucrare, iar o lucrare să fie efectuată de cel mult un muncitor, cu condiţia ca costul total de execuţie să fie minim.

Definim "matricea de afectare" astfel: $X = (x_{ij}), i = \overline{1,m}, j = \overline{1,n}, x_{ij} = 1$ dacă muncitorul x_i efectuează lucrarea y_j, si $x_{ij} = 0$ în caz contrar. Când $\sum_{i,j} = min(m, n)$, matricea de afectare se numeşte "saturantă". Voi prezenta algoritmul elaborat in 1955 de Kuhn, numit de acesta "algoritmul ungar pentru rezolvarea problemelor de afectare definite de matrice saturante", cunoscut şi sub numele de "metoda ungară", deoarece se bazează pe rezultatele a doi matematicieni maghiari: D. Kőnig si E. Egerváry. Demonstraţia va presupune că $m = n$; se va vedea apoi că algoritmul se poate extinde foarte uşor la matrice care nu sunt pătrate.

2 Descrierea algoritmului

Algoritmul utilizează următoarea proprietate:

Propoziţia 1 *Mulţimea soluţiilor minime ale unei probleme de afectare nu se modifică dacă la toate elementele unei linii sau coloane a matricei costurilor se adună acelaşi număr real λ.*

De asemenea, este necesară cunoasterea următoarei teoreme:

Teorema 1 (Teorema lui Kőnig) *Fie G un graf bipartit, $\nu(G)$ cardinalul unui cuplaj maxim în acest graf, si $\tau(G)$ cardinalul unui suport minim. Atunci există egalitatea:*

$$\nu(G) = \tau(G), \tag{1}$$

adică numărul maxim de muchii ale unui cuplaj este egal cu numărul minim de vârfuri ale unui suport.

Deoarece $m = n$ şi matricea de afectare este saturantă, orice soluţie a problemei de afectare conţine un singur 1 în fiecare linie şi în fiecare coloană, deci, prin transformarea matricei costurilor, costul asociat oricărei solutii creste cu λ. Din acest motiv, mulţimea soluţiilor minime ale problemei de afectare nu se modifică prin transformări de acest tip ale matricei costurilor.
Algoritmul lui Kuhn se bazează şi pe faptul că o soluţie a problemei de afectare corespunde unui cuplaj al grafului bipartit cu părţile $X = \{x_1, x_2, \ldots, x_n\}$ şi $Y = \{y_1, y_2, \ldots, y_n\}$, şi se desfăşoară în mai mulţi paşi, fiind prezentat aici într-o variantă modificată şi simplificată faţă de cea originală. Ideea este următoarea: dacă prin anumite operaţii de adunare a unei constante la toate elementele unei linii sau coloane a matricei costurilor toate numerele din matrice ramân pozitive şi există un cuplaj de cardinal n în graful bipartit definit de zerourile matricei, atunci soluţia problemei se gaseşte pe poziţiile nule din matrice.
Acest lucru se demonstrează imediat pe baza propoziţiei 1, având în vedere faptul că în matricea care conţine numai numere nenegative nu poate exista o transversală cu costul negativ. Vom încerca, deci, să aplicăm în mod repetat transformările descrise, până când va exista un cuplaj de cardinal maxim.
Vom adopta următoarele convenţii de transpunere a limbajului de teoria grafurilor în termeni de matrice:

- Spunem despre o linie sau despre o coloană că este "acoperită" dacă la un anumit pas face parte din mulţimea nodurilor marcate în cadrul procesului de determinare a unui suport minim.
- Spunem despre un element al matricei că este "încercuit" dacă face parte din cuplajul găsit până în momentul respectiv.
- Spunem despre un element al matricei că este "tăiat" dacă corespunde unei muchii care va putea fi folosită pentru obţinerea unui drum de creştere, într-un sens ce va fi clarificat ulterior.

3 Algoritmul propriu-zis

Iniţial, toate liniile si coloanele sunt decuplate. Algoritmul se va desfăşura în n paşi, fiecare pas decurgând după cum urmează: Se descoperă toate liniile şi coloanele matricei şi se acoperă toate coloanele cuplate. Se păstrează elementele încercuite, dar se consideră că nici un element nu este tăiat. Pornind de la

cuplajul obţinut la pasul anterior, se determină un suport minim de acelaşi cardinal, adică o mulţime de linii şi coloane care "acoperă" toate zerourile. Modul de obţinere a suportului minim a fost dezvoltat de Egerváry, iar demonstraţia corectitudinii sale depăşeşte scopul acestui articol. Cititorii interesaţi pot consulta [1] pentru o demonstraţie completă. Atâta timp cât putem găsi un zero descoperit pe o linie i şi o coloana j, îl tăiem şi considerăm următoarele doua cazuri:

Linia i este cuplată Acoperim linia i şi descoperim coloana cu care este cuplată. Coloana cu care este cuplată linia i va fi mereu acoperită din cauza modului în care lucrează algoritmul: în orice moment ori linia ori coloana pe care se află un zero încercuit vor fi acoperite, deoarece iniţial toate coloanele cuplate sunt acoperite, şi singura operaţie pe care o efectuăm este descoperirea unei coloane si acoperirea liniei cu care aceasta este cuplată.

Linia i nu este cuplată Încercăm să construim un drum de creştere, după cum urmează: Pornim de pe poziţia pe care tocmai am tăiat-o. Dacă coloana în care se află nu este cuplată, cuplăm linia i cu coloana j, şi drumul de creştere este complet. Dacă coloana j este cuplată, continuăm drumul de creştere cu zeroul încercuit din coloana j şi zeroul tăiat care se găseşte pe linia acelui zero. Fie k linia cu care este cuplată coloana j. (k, j) este încercuit, coloana j nu este acoperită, ceea ce înseamnă că linia k este acoperită. Dar o linie nu poate fi acoperită decât în condiţiile în care la un pas anterior am găsit un zero descoperit în acea linie, l-am tăiat, si am descoperit coloana din care făcea parte. În concluzie, vom putea găsi mereu un zero tăiat pe linia care ne interesează. Deoarece numărul elementelor încercuite este finit, fiecare din ele va fi selectat cel mult o dată, şi vom putea mereu să găsim un zero tăiat cu care să continuăm drumul de creştere, rezultă că în cele din urmă ori vom găsi un zero tăiat într-o coloană necuplată, moment în care drumul de creştere va fi complet.

Dacă la pasul anterior nu am găsit un drum valid de creştere, vom considera matricea formată din elementele care nu aparţin unor linii sau coloane acoperite şi vom calcula minimul λ al elementelor acestei submatrice. Obţinem $\lambda > 0$, deoarece elementele nule ale matricei se găsesc numai in liniile şi coloanele acoperite. Vom scădea λ din elementele coloanelor neacoperite şi îl vom aduna la elementele liniilor acoperite. Deoarece toate zerourile sunt acoperite o singură dată conform modului de obţinere a suportului minim, aceste operaţii nu vor face să dispară din zerourile obţinute până la acest pas, şi nici nu vor provoca apariţia unor elemente negative, însă vor produce cel puţin încă un zero, pe poziţia pe care a fost găsit minimul. Suportul minim al matricei costurilor va creşte cu cel puţin o unitate prin aplicarea repetată a acestor paşi. Conform teoremei lui Kőnig, aceasta înseamnă că numărul elementelor unui cuplaj maxim al matricei obţinute creşte cu cel puţin o unitate, deci în cele din urmă vom reuşi să găsim un drum de creştere cu ajutorul căruia să mărim cuplajul obţinut. Concluzia pe care o tragem este următoarea: la fiecare din cei n paşi

vom repeta aplicarea operaţiilor descrise până la obţinerea unui drum de ameliorare, iar cuplajul maxim va creşte cu 1. După cei n paşi, vom avea un cuplaj de cardinal n care va corespunde transversalei minime din matricea costurilor. Complexitatea algoritmului descris este $O(N^4)$, însă se comportă bine în practică, în funcţie de costurile muchiilor grafului pe care se realizează cuplajul.

4 Detalii de implementare

Algoritmul lui Kuhn este destul de cunoscut de către participanţii la concursurile de informatică, dar nu este folosit foarte des, deoarece se consideră că este mult prea greu de implementat. În continuare se va vedea că, cu puţină atenţie, metoda descrisă mai sus este implementabilă în timp de concurs, în cel mult 10 minute.
Avem nevoie de o matrice G care va reţine graful propriu-zis. Pentru a nu irosi spaţiul, în loc de a memora în altă parte matricea modificată, vom memora in vectorii vr si vc suma valorilor care au fost adăugate pe o anumită linie, respectiv scăzute pe o anumită coloană. Vectorii l şi r vor reţine cu cine este cuplată o anumită linie, respectiv coloană, sau 0 dacă acea linie / coloană nu a fost cuplată încă. $p_i = j$ are semnificaţia că (i, j) este tăiat, iar vectorii cr şi cc sunt folosiţi pentru a afla rapid dacă o linie / coloană este acoperită. Fără alte comentarii, voi trece direct la prezentarea codului sursă care implementează pas cu pas cele descrise de mai sus.

```
#include <stdio.h>
#include <string.h>

int N, G[256][256], l[256], r[256], p[256], cr[256],
cc[256], vr[256], vc[256];

void find_zero () {
int i, j, min, t;
for (min = 1<<30, i = 1; i <= N; ++ i) if (!cr[i])
for (j = 1; j <= N; ++ j) if (!cc[j])
min <?= G[i][j] + vr[i] - vc[j];
for (i = 1; i <= N; ++ i) if ( cr[i]) vr[i] += min;
for (j = 1; j <= N; ++ j) if (!cc[j]) vc[j] += min;
for (i = 1; i <= N; ++ i) if (!cr[i])
for (j = 1; j <= N; ++ j) if (!cc[j] && G[i][j] + vr[i] == vc[j])
if (r[i]) {
p[i] = j, cr[i] = 1, cc[ r[i] ] = 0;
break;
} else {
do t = l[j], r[i] = j, l[j] = i, i = t, j = p[i]; while (t);
return;
}
```

```
find_zero ();
}

int main () {
int i, j, min, cnt;
scanf ("%d", &N);
for (i = 1; i <= N; ++ i)
for (j = 1; j <= N; ++ j) scanf ("%d", &G[i][j]);
memset (vr, 0, sizeof (vr)), memset (vc, 0, sizeof (vc));
memset (l, 0, sizeof (l)), memset (r, 0, sizeof (r));
for (cnt = 0; cnt < N; ++ cnt) {
memset (cr, 0, sizeof (cr)), memset (p, 0, sizeof (p));
memcpy (cc, l, sizeof (cc));
find_zero ();
}
return 0;
}
```

5 Optimizări

Codul de mai sus a fost destul de compactat si optimizat încât să nu mai aibă nevoie decât de următorul pas pentru a stoarce şi ultima picătură de performanţă din el: Nu este nevoie să pornim cu cuplajul de cardinal 0. La început scădem valoarea minimă din fiecare linie, apoi scădem valoarea minimă din fiecare coloană. În acest moment putem să parcurgem matricea pas cu pas şi, când găsim un element nul a cărui linie şi coloană nu au fost cuplate îl adăugăm la cuplajul iniţial. Astfel, nu vom mai fi nevoiţi să facem n paşi, iar timpul de execuţie va scădea simţitor.

6 Extinderi

După cum am promis, algoritmul lui Kuhn va putea fi aplicat şi pe matrice ne-pătrate. Dacă $m \neq n$ vom transforma matricea costurilor astfel:

1. pentru $m < n$, se adaugă matricei costurilor iniţiale $n-m$ linii care conţin numai 0.
2. pentru $m > n$, se adaugă matricei costurilor iniţiale $m-n$ coloane care conţin numai 0.

În cazul unor probleme de afectare pentru care se caută o afectare saturantă care maximizează suma $\sum_{i,j} x_{ij} v_{ij}$, vom defini o nouă matrice a costurilor, ale cărei elemente sunt: $v'_{ij} = v_0 - v_{ij}, i = \overline{1,m}, j = \overline{1,n}, v_0 = max_{i,j} v_{ij}$. Observăm că dacă matricea costurilor iniţială conţine un element egal cu ∞ această problemă este banală, deci presupunem $v_0 < \infty$. Soluţia de cost minim a problemei cu matricea costurilor modificată va coincide cu soluţia de cost

maxim a problemei de afectare cu matricea costurilor iniţială. Demonstraţia este imediată, şi rămâne că temă cititorului.

References

[1] Caius Iacob, Matematici clasice şi moderne, vol. I, Bucureşti: Editura Tehnică, 1978