

8 METODA BRANCH AND BOUND

8.1. Prezentare generală

Metoda *Branch and Bound* se aplică problemelor care pot fi reprezentate pe un arbore: se începe prin a lua una dintre mai multe decizii posibile, după care suntem puși în situația de a alege din nou dintre mai multe decizii; vom alege una dintre ele etc. Vârfurile arborelui corespund stărilor posibile în dezvoltarea soluției.

Deosebim două tipuri de probleme:

- 1) Se caută un anumit vârf, numit *vârf rezultat*, care evident este final (nu are descendenți).
- 2) Există mai multe vârfuri finale, care reprezintă soluții posibile, dintre care căutăm de exemplu pe cel care *minimizează* o anumită funcție.

Exemplul 1. Jocul 15 (Perspico).

Un număr de 15 plăcuțe pătrate sunt incorporate într-un cadru 4×4, o poziție fiind liberă. Fiecare plăcuță este etichetată cu unul dintre numerele 1,2,...,15. Prin *configurație* înțelegem o plasare oarecare a plăcuțelor în cadru. Orice plăcuță adiacentă cu locul liber poate fi mutată pe acest loc liber. Dându-se o configurație inițială și una finală, se cere să determinăm o succesiune de mutări prin care să ajungem din configurația inițială în cea finală. Configurațiile inițială și finală pot fi de exemplu:

1	2	3	4
5		7	8
9	6	10	11
13	14	15	12

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

unde locul liber mai poate fi considerat drept conținând plăcuța imaginară cu eticheta 16.

Observație. Mutarea unei plăcuțe adiacente locului liber pe acel loc poate fi gândită și ca mutarea locului liber pe o poziție adiacentă.

Prezentăm întâi o condiție de existență a unei succesiuni de mutări prin care se poate trece de la configurația inițială în cea finală.

Cele 16 locașuri sunt considerate ca fiind ordonate de la stânga la dreapta și de jos în sus. Pentru plăcuța etichetată cu i definim valoarea $n(i)$ ca fiind numărul locașurilor care urmează celei pe care se află plăcuța și care conțin o plăcuță a cărei etichetă este mai mică decât i . De exemplu pentru configurația inițială de mai sus avem:

$n(8)=1$; $n(4)=0$; $n(16)=10$; $n(15)=1$ etc.

Fie l și c linia și coloana pe care apare locul liber. Fie $x \in \{0, 1\}$ definit astfel: $x=0$ dacă și numai dacă $l+c$ este par. Se poate demonstra următorul rezultat:

Propoziție. Fiind dată o configurație inițială, putem trece din ea la configurația finală de mai sus $\Leftrightarrow n(1)+n(2)+\dots+n(16)+x$ este par.

În continuare vom presupune că putem trece de la configurația inițială la cea finală.

Cum locul liber poate fi mutat spre N, S, E, V (fără a ieși însă din cadru), rezultă că fiecare configurație (stare) are cel mult 4 descendenți. Se observă că arborele astfel construit este infinit. Stările finale sunt stări rezultat și corespund configurației finale.

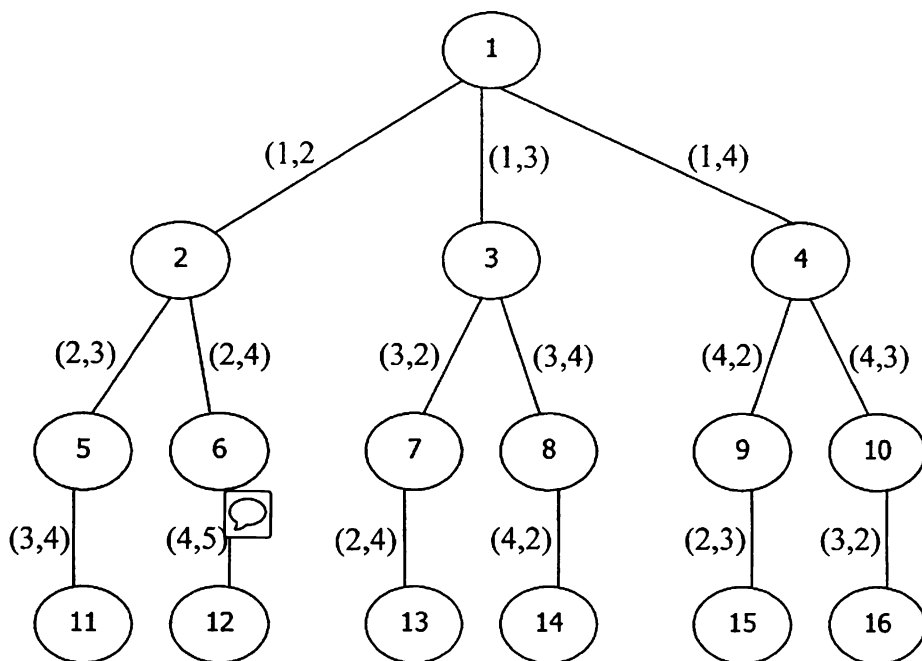
Exemplul 2. Circuitul hamiltonian de cost minim.

Se consideră un graf orientat cu arcele etichetate cu costuri pozitive. Inexistența unui arc între două vârfuri este identificată prin "prezența" sa cu costul $+\infty$. Presupunem că graful este dat prin matricea C a costurilor sale. Se cere să se determine, dacă există, un circuit hamiltonian de cost minim.

Să considerăm, de exemplu, graful dat de matricea de costuri:

$$C = \begin{pmatrix} \infty & 3 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$

Arborele spațiului de stări, în care muchiile corespund arcelor din graf, este următorul:



subînțelegându-se că se pleacă din vârful 1 și că din frunze se revine la acest vârf.

Revenim la descrierea metodei Branch and Bound.

Știm că și metoda backtracking este aplicabilă problemelor reprezentabile pe arbori. Există însă multe deosebiri, dintre care menționăm următoarele:

- ordinea de parcurgere a arborelui;
- modul în care sunt eliminați subarborii care nu pot conduce la o soluție;
- faptul că arborele poate fi infinit (prin natura sa sau prin faptul că mai multe vârfuri pot corespunde la o aceeași stare).

În general arborele de stări este construit dinamic.

Este folosită o listă L de *vârfuri active*, adică de stări care sunt susceptibile de a fi dezvoltate pentru a ajunge la soluție/soluții. Inițial, lista L conține rădăcina arborelui, care este *vârful curent*. La fiecare pas, din L alegem un vârf (care nu este neapărat un fiu al vârfului curent!), care devine noul vârf curent.

Când un vârf activ devine vârf curent, sunt generați toți fiii săi, care devin vârfuri active (sunt incluși în L). Apoi din nou este selectat un vârf curent.

Legat de modul prin care alegem un vârf activ drept vârf curent, deci implicit legat de modul de parcurgere a arborelui, facem următoarele remarci:

- parcurgerea DF nu este adecvată, deoarece pe de o parte arborele poate fi infinit, iar pe de altă parte soluția căutată poate fi de exemplu un fiu al rădăcinii

diferit de primul fiu și parcurgerea în adâncime ar fi inefficientă: se parcurg inutil stări, în loc de a avansa direct spre soluție;

– parcurgerea pe lățime conduce totdeauna la soluție (dacă aceasta există), dar poate fi inefficientă dacă vârfurile au mulți fii.

Metoda Branch and Bound încearcă un "compromis" între cele două parcurgeri menționate mai sus, atașând vârfurilor active câte un *cost* pozitiv, ce intenționează să fie o măsură a gradului de "aproiere" a vârfului de o soluție. Alegerea acestui cost este decisivă pentru a obține un timp de executare cât mai bun și depinde de problema concretă, dar și de abilitatea programatorului.

Observație. Costul unui vârf va fi totdeauna mai mic decât cel al descendenților (fiilor) săi.

De fiecare dată drept vârf curent este ales cel de cost minim (cel considerat ca fiind cel mai "aproape" de soluție). De aceea L va fi în general un min-ansamblu: costul fiecărui vârf este mai mic decât costul descendenților.

Din analiza teoretică a problemei deducem o valoare lim care este o aproximație prin adaos a minimului căutat: atunci când costul unui vârf depășește lim , vârful curent este ignorat: nu este luat în considerare și deci este eliminat întregul subarbore pentru care este rădăcină. Dacă nu cunoaștem o astfel de valoare lim , o inițializăm cu $+\infty$.

Se poate defini o *funcție de cost ideală*, pentru care $c(x)$ este dat de:

- nivelul pe care se află vârful x dacă x este vârf rezultat;
- $+\infty$ dacă x este vârf final, diferit de vârf rezultat;
- $\min \{c(y) \mid y \text{ fiu al lui } x\}$ dacă x nu este vârf final.

Această funcție este ideală din două puncte de vedere:

- nu poate fi calculată dacă arborele este infinit; în plus, chiar dacă arborele este finit, el trebuie parcurs în întregime, ceea ce este exact ce dorim să evităm;
- dacă totuși am cunoaște această funcție, soluția poate fi determinată imediat: plecăm din rădăcină și coborâm mereu spre un vârf cu același cost, până ajungem în vârful rezultat.

Neputând lucra cu funcția ideală de mai sus, vom alege o aproximație \hat{c} a lui c , care trebuie să satisfacă condițiile:

- 1) în continuare, dacă y este fiu al lui x avem $\hat{c}(x) < \hat{c}(y)$;
- 2) $\hat{c}(x)$ să poată fi calculată doar pe baza informațiilor din drumul de la rădăcină la x ;

- 3) este indicat ca $\hat{c} \leq c$ pentru a ne asigura că dacă $\hat{c}(x) > \lim$, atunci și $c(x) > \lim$, deci x nu va mai fi dezvoltat.

O primă modalitate de a asigura compromisul între parcurgerile în adâncime și pe lățime este de a alege funcția \hat{c} astfel încât, pentru o valoare naturală k , să fie îndeplinită condiția: pentru orice vârf x situat pe un nivel n_x și orice vârf situat pe un nivel $n_y \geq n_x + k$, să avem $\hat{c}(x) > \hat{c}(y)$, indiferent dacă y este sau nu descendent al lui x .

Condiția de mai sus spune că niciodată nu poate deveni activ un vârf aflat pe un nivel $n_y \geq n_x + k$ dacă în L apare un vârf situat pe nivelul n_x , adică nu putem merge "prea mult" în adâncime. Dacă această condiție este îndeplinită, este valabilă următoarea propoziție:

Propoziție. În ipoteza că este îndeplinită condiția de mai sus și dacă există soluție, ea va fi atinsă într-un timp finit, chiar dacă arborele este infinit.

Putem aplica cele de mai sus pentru jocul Perspico, alegând:

$\hat{c}(x)$ = suma dintre lungimea drumului de la rădăcină la x și numărul de plăcuțe care nu sunt la locul lor (aici $k=15$).

8.2. Algoritmul Branch & Bound pentru probleme de optim

Să presupunem că dorim să determinăm vârful final de cost minim și drumul de la rădăcină la el. Fie \lim aproximarea prin adaos considerată mai sus.

Algoritmul este următorul (rad este rădăcina arborelui, iar i_{final} este vârful rezultat):

```

i ← rad; L ← {i}; min ← lim;
calculăm  $\hat{c}(\text{rad})$ ; tata(i) ← 0
while L ≠ ∅
  i ← L {este scos vârful i cu  $\hat{c}(i)$  minim din min-ansamblul L}
  for toți j fii ai lui i
    calculăm  $\hat{c}(j)$ ; calcule locale asupra lui j; tata(j) ← i
    if j este vârf final
      then if  $\hat{c}(j) < \min$ 
        then min ←  $\hat{c}(j)$ ;  $i_{\text{final}} \leftarrow j$ 
        elimină din L vârfurile k cu  $\hat{c}(k) \geq \min$  (*)
      else if  $\hat{c}(j) < \min$ 
        then j ⇒ L

```

```

if min=lim then write('Nu există soluție')
else writeln(min); i ← ifinal
while i ≠ 0
    write(i); i ← tata(i)

```

Observație. La (*) am ținut cont de faptul că dacă j este descendent al lui i , atunci $\hat{c}(i) < \hat{c}(j)$.

Vom aplica algoritmul de mai sus pentru problema circuitului hamiltonian de cost minim, pe exemplul considerat mai sus.

Pentru orice vârf x din arborele de stări, valoarea $c(x)$ dată de funcția de cost ideală este:

- lungimea circuitului corespunzător lui x dacă x este frunză
- $\min \{c(y) \mid y \text{ fiu al lui } x\}$ altfel.

Fiecărui vârf x îi vom atașa o *matrice de costuri* M_x (numai dacă nu este frunză) și o valoare $\hat{c}(x)$.

Observație. Dacă micșorăm toate elementele unei linii sau coloane cu α , orice circuit hamiltonian va avea costul micșorat cu α , deoarece în orice circuit hamiltonian din orice vârf pleacă exact un arc și în orice vârf sosește exact un arc.

Conform acestei observații, vom lucra cu *matrici de costuri reduse* (în care pe orice linie sau coloană apare cel puțin un zero, exceptând cazul când linia sau coloana conține numai ∞).

Pentru rădăcina $\text{rad}=1$ plecăm de la matricea de costuri C . Matricea atașată va fi matricea redusă obținută din C , iar $\hat{c}(1)$ = cantitatea cu care s-a redus matricea C .

În general, pentru un vârf y oarecare al cărui tată este x și muchia (x, y) este etichetată cu (i, j) :

- dacă y este vârf terminal, $\hat{c}(x)$ va fi chiar $c(y)$, adică costul real al circuitului;
- în caz contrar, plecând de la M_x și $\hat{c}(x)$ procedăm astfel:

$$\begin{array}{c} x \\ \left| \right. \\ (i, j) \\ \left| \right. \\ y \end{array}$$

 - elementele liniei i devin ∞ , deoarece mergem sigur către vârful j din graf;
 - elementele coloanei j devin ∞ , deoarece am ajuns sigur în vârful j din graf;
 - $M_x(j, 1) \leftarrow \infty$, pentru a nu reveni prematur în rădăcina 1;
 - reducem noua matrice M_x și obținem M_y ; fie r cantitatea cu care s-a redus M_x . Vom lua $\hat{c}(y) \leftarrow \hat{c}(x) + r + M_x(i, j)$.

Concret, pentru exemplul dat, calculele se desfășoară astfel:

- Pentru rădăcină:
 - reducem liniile în ordine cu 2, 1, 3, 2;
 - reducem prima coloană cu 1;
 - în acest mod obținem $\hat{c}(1) = 9$

$$M_1 = \begin{pmatrix} \infty & 1 & 5 & 0 \\ 3 & \infty & 0 & 8 \\ 0 & 5 & \infty & 0 \\ 3 & 0 & 4 & \infty \end{pmatrix}$$

♦ Acum $\min \leftarrow 9$; $L = \{1\}$. Este extras vârful 1 și sunt considerați fiii săi.

- Pentru vârful 2:
 - plecăm de la M_1 și punem ∞ pe linia 1 și coloana 2;
 - elementul de pe linia 2 și coloana 1 devine ∞ ;
 - reducem linia 3 cu 3;
 - în acest mod obținem $\hat{c}(2) = 9 + 3 + 1 = 13$

$$M_2 = \begin{pmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 8 \\ 0 & \infty & \infty & 0 \\ 0 & \infty & 1 & \infty \end{pmatrix}$$

- Pentru vârful 3:
 - plecăm de la M_1 și punem ∞ pe linia 1 și coloana 3;
 - elementul de pe linia 3 și coloana 1 devine ∞ ;
 - reducem linia 2 cu 3;
 - în acest mod obținem $\hat{c}(3) = 9 + 3 + 5 = 17$

$$M_3 = \begin{pmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & 5 \\ \infty & 5 & \infty & 0 \\ 3 & 0 & \infty & \infty \end{pmatrix}$$

- Pentru vârful 4:
 - plecăm de la M_1 și punem ∞ pe linia 1 și coloana 4;
 - elementul de pe linia 4 și coloana 1 devine ∞ ;
 - nu este necesară vreo reducere;
 - în acest mod obținem $\hat{c}(4) = 9 + 0 + 0 = 9$

$$M_4 = \begin{pmatrix} \infty & \infty & \infty & \infty \\ 3 & \infty & 0 & \infty \\ 0 & 5 & \infty & \infty \\ \infty & 0 & 4 & \infty \end{pmatrix}$$

♦ Acum $L = \{2, 3, 4\}$ cu $\hat{c}(2) = 13$, $\hat{c}(3) = 17$, $\hat{c}(4) = 9$. Devine activ vârful 4.

- Pentru vârful 9:
 - plecăm de la M_4 și punem ∞ pe linia 4 și coloana 2;
 - elementul de pe linia 2 și coloana 1 devine ∞ ;
 - nu este necesară vreo reducere;
 - în acest mod obținem $\hat{c}(9) = 9 + 0 + 0 = 9$

$$M_9 = \begin{pmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty \\ 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \end{pmatrix}$$

- Pentru vârful 10:
 - plecăm de la M_4 și punem ∞ pe linia 4 și coloana 3;
 - elementul de pe linia 3 și coloana 1 devine ∞ ;
 - reducem linia 2 cu 3, iar linia 3 cu 5;
 - în acest mod obținem $\hat{c}(10) = 9 + 8 + 4 = 21$

$$M_{10} = \begin{pmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty \end{pmatrix}$$

- ♦ Acum $L=\{2, 3, 9, 10\}$ cu $\hat{c}(2)=13$, $\hat{c}(3)=17$, $\hat{c}(9)=9$, $\hat{c}(10)=21$. Devine activ vârful 9. Singurul său descendent este 15, care este frunză. $\hat{c}(15)=c(15)=9$ (costul real al circuitului). Sunt eliminate din L vârfurile cu costurile mai mari decât 9, deci L devine vidă. min rămâne egal cu 9, va fi produs la ieșire circuitul căutat (1,4,2,3,1) și algoritmul se oprește.