Unitatea de învățare nr. 6

METODA BRANCH AND BOUND

Cuprins

Obiectivele unității de învățare nr. 6

După ce veți parcurge această unitate de învățare, veți reuşi să:

- stăpâniți o metodă complexă de programare, ce îmbină parcurgerea în adâncime a unui arbore cu parcurgerea sa pe lățime;
- aplicați metoda *Branch and Bound* problemelor de optim.

Indicații metodice pentru unitatea de învățare nr. 6

Materialul trebuie parcurs în ordinea sa firească, prezentată în continuare. Se recomandă conspectarea şi notarea ideilor principale, repcum şi consultarea bibliografiei pentru detalii şi informații suplimentare.

Timpul minim pe care trebuie să-l acordați acestei unități de învățare este de **8 ore**.

6.1. Prezentare generală

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

Deosebim două tipuri de probleme:

1) Se caută un anumit vârf, numit *vârf rezultat*, care evident este final (nu are descendenţi).

2) Există mai multe vârfuri finale, care reprezintă soluţii posibile, dintre care căutăm de exemplu pe cel care optimizează o anumită funcţie.

Exemplul 1:

Jocul 15 (Perspico).

Un număr de 15 plăcuţe pătrate sunt incorporate într-un cadru 4×4, o poziţie fiind liberă. Fiecare plăcuţă este etichetată cu unul dintre numerele 1,2,...,15. Prin *configuraţie* înţelegem o plasare oarecare a plăcuţelor în cadru. Orice plăcuţă adiacentă cu locul liber poate fi mutată pe acest loc liber. Dându-se o configuraţie iniţială şi una finală, se cere o succesiune de mutări prin care să ajungem din configuraţia iniţială în cea finală. Configuraţiile iniţială şi 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 conţinând plăcuţa imaginară cu eticheta 16.

Prezentăm întâi o condiţie de existenţă a unei succesiuni de mutări prin care se poate trece de la configuraţia iniţială în cea finală.

Cele 16 locaşuri sunt considerate ca fiind ordonate de la stânga la dreapta şi de jos în sus. Pentru plăcuţa etichetată cu `i` definim valoarea `n(i)` ca fiind numărul locaşurilor care urmează celei pe care se află plăcuţa şi care conţin o plăcuţă a cărei etichetă este mai mică decât `i`. De exemplu pentru configuraţia iniţială de mai sus avem:
`n(8)=1`; `n(4)=0`; `n(16)=10`; `n(15)=1` etc.

Fie `l` şi `c` linia şi coloana pe care apare locul liber. Fie `x∈{0,1}` definit astfel: `x=0` dacă şi 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* ⇔ `n(1)+n(2)+...+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 ieşi 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 şi corespund configurației finale.

Exemplul 2:

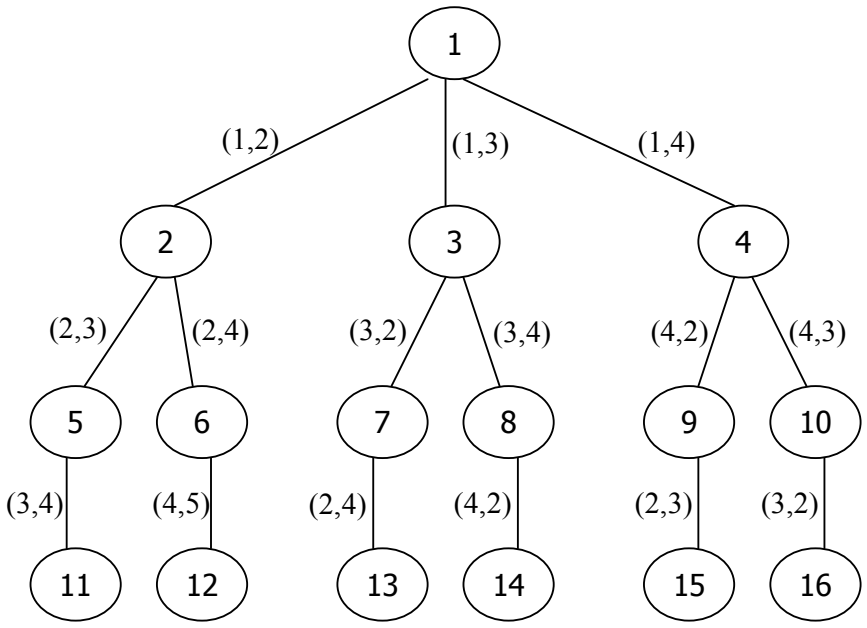
Circuitul hamiltonian de cost minim

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 vârfurile corespund vârfurilor din graf, iar muchiile reprezintă arce din graf, este următorul:



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

Revenim la descrierea metodei Branch and Bound.

Ştim că şi 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 aceeaşi 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 incluşi 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 şi parcurgerea în adâncime ar fi ineficientă: 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 ineficientă dacă vârfurile au mulți fii.

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

Observație: Totdeauna costul unui vârf va fi mai mic decât cel al descendenților (fiilor).

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

depăşeşte `lim`, atunci vârful curent este ignorat: nu este luat în considerare şi deci este eliminat întregul subarbore pentru care este rădăcină. Dacă nu cunoaştem 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)` | `y` 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 totuşi am cunoaşte această funcție, soluția poate fi determinată imediat: plecăm din rădăcină şi coborâm mereu spre un vârf cu acelaşi 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 informaților din drumul de la rădăcină la `x` ;
3) este indicat ca $\hat{c} \le c$ pentru a ne asigura că dacă $\hat{c}(x) > lim$, atunci şi $c(x) > lim$, deci `x` nu va mai fi dezvoltat.

O primă modalitate de a asigura compromisul între parcurgerile în adâncime şi 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 `nx` şi orice vârf situat pe un nivel $ny \ge nx+k$, 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 $ny \ge nx+k$ dacă în `L` apare un vârf situat pe nivelul `nx`, adică nu putem merge "prea mult" în adâncime. Dacă această condiție este îndeplinită, este valabilă următoarea propoziție:

Propoziție: *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` şi numărul de plăcuțe care nu sunt la locul lor (aici k=15).

6.2. Algoritmul Branch & Bound pentru probleme de optim

Să presupunem că dorim să determinăm vârful final de cost minim şi 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;
calculez ĉ(rad); tata(i) ← 0
while L ≠ ∅
   i ⇐ L {este scos vârful i cu ĉ(i) minim din min-ansamblul L}
   for toți j fii ai lui i
      calculez ĉ(j); calcule locale asupra lui j; tata(j) ← i
      if j este vârf final
      then if ĉ(j)<min
           then min ← ĉ(j); i_final ← j
                  elimină din L vârfurile k cu ĉ(k)≥min
       (*)
      else if ĉ(j)<min
           then j ⇒ L
if min=+lim then write('Nu există soluție')
            else writeln(min); i ← i_final
                 while i ≠ 0
                    write(i); i ← tata(i)
```

Observație: La (*) am ținut cont de faptul că dacă `j` este descendent al lui `i`, atunci `ĉ(i)<ĉ(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)` | `y` fiu al lui `x`} altfel.

Fiecărui vârf `x` îi vom ataşa o *matrice de costuri* M_x (numai dacă nu este frunză) şi valoarea `ĉ(x)`.

Observație: Dacă micşorăm toate elementele unei linii sau coloane cu α, orice circuit hamiltonian va avea costul micşorat cu α, deoarece în orice circuit hamiltonian din orice vârf pleacă exact un arc şi în orice vârf soseşte 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 `rad`=1 plecăm de la matricea de costuri `C`. Matricea ataşată va fi matricea redusă obținută din `C`, iar `ĉ(1)` = cantitatea cu care s-a redus `C`.
În general, pentru un vârf `y` oarecare al cărui tată este `x` şi muchia `(x,y)` este etichetată cu `(i,j)`:

- dacă `y` este vârf terminal, `ĉ(x)` va fi chiar `c(y)`, adică costul real al circuitului;
- în caz contrar, plecând de la M_x şi `ĉ(x)` procedăm astfel:

- 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 şi obținem M_y; fie `r` cantitatea cu care s-a redus M_x. Vom lua `ĉ(y)` ← `ĉ(x)+r+`M_x`(i,j)`.

Concret, pentru exemplul dat, calculele se desfăşoară astfel:

- Pentru rădăcină:

- reduc liniile în ordine cu 2, 1, 3, 2;
- reduc prima coloană cu 1;
- în acest mod obțin `ĉ(1)=9`, iar matricea M_1 este:

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

- Acum `min`←`9`; `L={1}`. Este extras vârful 1 şi sunt considerați fiii săi.

- Pentru vârful 2:

- plec de la M_1 şi pun ∞ pe linia 1 şi coloana 2;
- reduc linia 3 cu 3;
- în acest mod obțin `ĉ(2)=3+9+1=13`, iar matricea M_2 este:

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

- Pentru vârful 3:

- plec de la M_1 şi pun ∞ pe linia 1 şi coloana 3;
- reduc linia 2 cu 3;
- în acest mod obțin `ĉ(3)=3+9+5=17`, iar matricea M_3 este:

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

- Pentru vârful 4:

- plec de la M_1 şi pun ∞ pe linia 1 şi coloana 4;
- nu este necesară vreo reducere;
- în acest mod obțin $\hat{c}(4)=0+9+0=9$, iar matricea M_4 este

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

♦ 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:

- plec de la M_4 şi pun ∞ pe linia 4 şi coloana 2;
- nu este necesară vreo reducere;
- în acest mod obțin $\hat{c}(9)=0+9+0=9$, iar matricea M_9 este:

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

- Pentru vârful 10:

- plec de la M_4 şi pun ∞ pe linia 4 şi coloana 3;
- reduc linia 2 cu 3, iar linia 3 cu 5;
- în acest mod obțin $\hat{c}(10)=8+9+4=21$, iar matricea M_{10} este:

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

♦ 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 ieşire circuitul căutat (1,4,2,3,1) şi algoritmul se opreşte.

Răspunsurile la test se vor da în spațiul liber din chenar, în continuarea enunțurilor

Testul de autoevaluare nr. 1

1. Care este clasa de probleme pentru care se poate aplica metoda *Branch and Bound*?

2. Cum se alege funcția de cost şi care rolul ei?

3. Care este criteriul prin care din mulțimea vârfurilor active este ales vârful curent ?

4. Care este relația între o funcție de cost oarecare şi funcția de cost ideală?

Răspunsurile la acest test se găsesc pe pagina următoare.

6.3. Comentarii şi răspunsuri la testele de autoevaluare

Testul 1.

1. Metoda *Branch and Bound* este aplicabilă problemelor care se pot reprezenta pe un arbore (finit sa infinit) de căutare.

2. Orice funcție de cost c trebuie să îndeplinească condițiile:
   - dacă y este fiu al lui x avem c(x)<c(y);
   - c(x) să poată fi calculată doar pe baza informațiilor din drumul de la rădăcină la x .
   Rolul ei este de a aproxima funcția de cost ideală.

3. Pentru problemele de minim, vârful curent este ales din mulțimea vârfurilor active şi anume este ales ca fiind cel de cost minim.

4. Varianta recursivă are o formă mai simplă şi mai concisă. În schimb varianta iterativă aduce un plus de eficiență, ce se regăseşte în timpul de executare.

5. În general, pentru problemele de minim, funcția de cost aleasă trebuie să aibă valori mai mici decât funcția de cost ideală.

6.4. Lucrare de verificare pentru studenți

Scrieți câte un program ce foloseşte metoda Branch and Bound pentru rezolvarea celor două probleme de mai jos:

1. **Jocul Perspico.** Enunțul este prezentat mai sus.

2. **Jocul Quad.** QUAD se joacă pe o tablă având nouă câmpuri: patru colțuri, patru mijloace, şi un centru, fiecare câmp putând fi alb sau negru. Un exemplu de configurație este următoarea:

colț	mijloc	colț
mijloc	**centru**	mijloc
colț	mijloc	**colț**

în care câmpurile negre apar întărite.
Regulile de joc sunt următoarele:
– acționarea unui câmp negru are efect nul;
– acționarea unui colț alb schimbă culorile lui, ale mijloacelor vecine şi a centrului;
– acționarea unui mijloc alb schimbă culorile lui şi ale colțurilor vecine;
– acționarea centrului alb schimbă culorile lui şi ale mijloacelor vecine.
Scopul jocului este de a trece dintr-o configurație inițială într-o configurație finală, ambele date.

Rezolvările, dar şi dificultățile întâmpinate, vor fi expediate prin mail tutorelui.

6.5. Bibliografie

[1] Georgescu Horia, *Tehnici de programare*, Editura Universității Bucureşti, 2005
[2] Cormen T. H. et al. *Introducere în algoritmi*, Computer Libris Agora, 2000
[3] Parberry Ian, Gasarch William, *Problems on Algorithms* (2nd Edition), 2002
[4] Cormen T. H. et al. *Introducere în algoritmi*, Computer Libris Agora, 2000
[5] Skiena Steven, *The Algoritm Design Manual*, Springer 1998