

## Kapitola 3

# Metódy založené na prehl'adávaní stavového priestoru

Uvažujme taký typ úloh, ktoré možno charakterizovať nasledovným spôsobom. Výstupom úlohy je  $n$ -tica (množina  $n$ -tíc, prípadne postupnosť), ktorá spĺňa nejaké ohraničenia. Pritom požadované riešenie možno napísať v tvare  $(x_1, x_2, \dots, x_n)$ , kde  $x_i \in S_i$ , kde  $S_i$  je **konečná** množina.

**Príklad 3.1 Problém  $n$ -dám.** Na šachovnicu  $n \times n$  chceme umiestniť  $n$  dám tak,  $n$  dám na šachovnici aby v každom riadku a v každom stĺpci stála dáma a aby sa navzájom neohrozovali. Riešenie hľadáme vo forme vektora dĺžky  $n$ , kde na  $s$ -tej pozícii je číslo riadku, v ktorom je v  $s$ -tom stĺpci umiestnená dáma.

Iným, nevhodným problémom, je aj problém triedenia, ktorý možno naformulovať aj takto: keď triedime  $n$ -prvkovú množinu, výsledkom môže byť  $n$ -tica indexov  $i_1, i_2, \dots, i_n$ , kde  $i_j$  je index  $j$ -teho najmenšieho prvku zo vstupu.

Všetky potenciálne  $n$ -tice tvoria potenciálny stavový priestor úlohy/priestor riešení. Vďaka konečnosti jednotlivých množín  $S_i$  je tento priestor konečný. Keďže však stavový priestor úlohy obsahuje minimálne všetky potenciálne riešenia, je jeho veľkosť aspoň  $|S_1| \cdot |S_2| \cdot \dots \cdot |S_m|$ .

Konečnosť priestoru riešení umožňuje hľadať požadované riešenie napríklad

**metódou hrubej sily** – vygenerujeme všetky prípustné riešenia a z nich vyberieme tie, ktoré spĺňajú nami požadované požiadavky. Zložitosť je určite aspoň veľkosť stavového priestoru, čo je veľa.

Budeme sa snažiť vektor budovať postupne, pričom pre každý vygenerovaný začiatok vektora zistíme, či sa dá predĺžiť na riešenie. Má to tú výhodu, že ak pre nejaký začiatok  $x_1, x_2, \dots, x_k$  zistíme, že sa nedá predĺžiť, potom minimálne

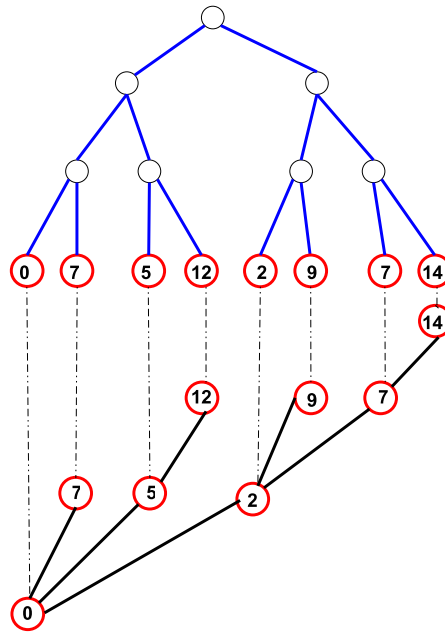
$$|S_{k+1}| \cdot \dots \cdot |S_n|$$

prípadov nemusíme generovať.

**Podmienky**, ktoré má riešenie spĺňať, sú väčšinou dvojakého charakteru

– *explicitné*, ktoré hovoria o  $x_i$  ako takom. Špecifikujú hodnoty, ktoré môže  $x_i$  nadobúdať ( $x_i \in S_i, x_i \leq 20, \dots$ ). Tieto podmienky definujú stavový priestor problému





Keď máme predstavu o tom, ako by vyzeral strom stavového priestoru problému, musíme systematicky "generovať" vrcholy tohto stromu. K lepšiemu popisu budeme uvažovať 3 kategórie vrcholov:

**E(expand)** – vrchol, ktorý sa práve vybral na spracovanie

*kategórie vrcholov*

**L(live)** – živý vrchol, teda taký, ktorý sme už vygenerovali, ale ešte sa doň budeme vracieť

**D(dead)** – mŕtvy vrchol, ktorý už má spracované všetky deti, resp. sme zistili, že sa z neho nevieme dostať do prípustného riešenia.

"Notoricky známe" sú dva systematické postupy prehľadávania stromov a grafov. Z nich vychádzajú aj dve základné metódy:

Branch & Bound  
 $\Downarrow$   
 do šírky + orezanie

Backtracking  
 $\Downarrow$   
 do hĺbky + orezanie

### 3.1 Backtracking-prehľadávanie s návratom

Spomedzi "priestor prehľadávajúcich metód" začneme s metódou prehľadávania s návratom – tzv. backtrackingom.

Nech  $T(X(1), \dots, X(k-1))$  je množina potenciálnych hodnôt pre  $X(k)$

*metóda*

$B_k(X(1), \dots, X(k))$  je booleovská funkcia, ktorá dáva hodnotu true práve vtedy keď  $X(1), \dots, X(k)$  je alebo môže byť predĺžené na riešenie.

Potom metódu možno schématicky znázorniť algoritmom 7. Efektívnosť tejto metódy zrejme závisí od

- výpočtu  $T(\cdot)$ , mohutnosti  $T(\cdot)$
- výpočtu  $B_k(\cdot)$ , počtu prvkov, spĺňajúcich  $B_k(\cdot)$ .

**Algoritmus 7** BACKTRACK

---

```

1:  $k \leftarrow 1$ 
2: while  $k > 0$  do
3:   if  $\exists X(k); X(k) \in T(X(1), \dots, X(k-1))$  a  $B_k(X(1), \dots, X(k))$  then
4:     if  $X(1), \dots, X(k)$  je cesta do odpovedového vrchola then
5:       return  $X(1), \dots, X(k)$ 
6:      $k \leftarrow k + 1$ 
7:   else  $k \leftarrow k - 1$ 

```

---

Zamyslime sa nad tým, či a ako môžeme ovplyvniť počet vrcholov stromu, ktoré pri prehľadávaní stavového priestoru vygenerujeme.

*skúšame zmenšiť veľkosť prezretého priestoru*

- ak nezáleží na poradí dopĺňaných položiek výsledného vektora, zdá sa rozumnejšie generovať položky v poradí, ktoré zodpovedá neklesajúcemu usporiadaniu mohutností príslušných  $S_i$ . Intuícia za tým – ak zistíme, že sa prvých  $k$  položiek *nedá* doplniť na riešenie, potom  $S(k, m) = |S_{k+1}| \cdot \dots \cdot |S_m|$  prípadov nemusíme generovať. Chceli by sme, aby  $S(k, m)$  bolo čo možno najväčšie číslo.
- pre zvolenú postupnosť  $i_1, i_2, \dots, i_n$  skúsime odhadnúť, koľko vrcholov bude treba vygenerovať. Ako? Nech pre vrchol v hĺbke 1 vygenerujeme  $m_1$  synov spĺňajúcich  $B_1$ . Náhodne sa presuneme do jedného z nich a zistíme, koľko má synov, spĺňajúcich  $B_2$ . Nech je to  $m_2 \dots$ . Nech  $m_i$  je počet synov vrchola v hĺbke  $i$ , ktorí spĺňajú  $B_i$ . Potom počet vygenerovaných vrcholov odhadneme ako

$$m_1 + m_1 m_2 + m_1 m_2 m_3 + \dots + m_1 m_2 \dots m_{n-1}$$

Zopakovaním pre niekoľko zvolených permutácií môžeme vybrať tú permutáciu, pre ktorú nám odhad vychádza najlepšie.

*opäť problém súčtov* Vráťme sa k problému čiastočných súčtov. Predpokladáme, že

$$w_1 \leq w_2 \leq \dots \leq w_n, M \in \mathbb{N}$$

Budeme aplikovať statický prístup. Hľadáme teda vektor dĺžky  $n$ , ktorého  $i$ -ta položka hovorí o prítomnosti, resp. neprítomnosti  $w_i$  v súčte. Ako bude vyzerať ohraničujúca funkcia?

Predpokladajme, že  $\sum_{i=1}^{k-1} x_i w_i < M$

- Kedy môžeme uvažovať, že  $\mathbf{x}_k = \mathbf{1}$ ?
  - Ak  $\sum_{i=1}^k w_i x_i \leq M$  a  $\sum_{i=1}^k w_i x_i + \sum_{i=k+1}^n w_i \geq M$ , tak áno
  - Ak  $\sum_{i=1}^k w_i x_i < M$  a  $\sum_{i=1}^k w_i x_i + w_{k+1} > M$ , vtedy možnosť  $x_k = 1$  nemôže viesť k riešeniu a preto ju neaplikujeme.
- Kedy môžeme uvažovať, že  $\mathbf{x}_k = \mathbf{0}$ ?
  - Vtedy, ak  $\sum_{i=1}^{k-1} w_i x_i + \sum_{i=k+1}^n w_i \geq M$

Označme

$$s = \sum_{i=1}^{k-1} w_i x_i \qquad r = \sum_{i=k+1}^n w_i$$

**Algoritmus 8** SumSub(s,k,r)

---

```

 $x(k) \leftarrow 1;$ 
if  $s+w(k) = M$  then return  $(x(1), \dots, x(k))$ 
else if  $s + w(k) + w(k+1) \leq M$  then
    SumSub( $s+w(k)$ ,  $k+1$ ,  $r-w(k)$ )
if  $s+r-w(k) \geq M$  a  $s+w(k+1) \leq M$  then
     $X(k) \leftarrow 0$ 
    Sum-of-sub( $s,k+1,r-w(k)$ )

```

---

**3.2 LC Branch and Bound**

Prehľadávanie do hĺbky i do šírky sú vlastne "slepé" metódy – postup prehľadávania je daný dopredu a príliš nezohľadňuje kvalitu získavaného riešenia. Metóda LC Branch and Bound (LC-B&B) sa snaží túto "sleposť" odstrániť. Rieši sa to určením inteligentnej funkcie  $c(x)$ , ktorou ohodnotíme živé vrcholy. Hodnotu tejto funkcie potom využijeme pri voľbe nového E-vrchola.

*ohodnocujeme  
vrcholy*

Čo by táto funkcia mala odrážať? Ideálne by bolo, keby hovorila o úsilí, ktoré ešte treba vynaložiť na to, aby sme sa z daného vrchola dostali k odpovedi (resp. o hodnote účelovej funkcie, ktorú z daného vrchola môžeme dosiahnuť).

- počet vrcholov, ktoré ešte treba vygenerovať
- počet úrovní, ktoré nás delia od odpovede

Keď chceme presnú hodnotu, znamenalo by to prezretie celého stavového priestoru, preto budeme používať radšej len odhad.

$\hat{c}(x) = f(h(x)) + \hat{g}(x)$ , kde  
 $h(x)$  je cena dosiahnutia  $x$  z koreňa  
 $f()$  je nejaká neklesajúca funkcia  
 $\hat{g}(x)$  je odhad úsilia do najbližšieho odpovedového listu

Ak zvolíme  $f(x) = 0$ , tak vlastne vôbec neuvažujeme úsilie vynaložené na príchod do vrchola. Zodpovedá to tomu, že celkom prirodzene predpokladáme, že  $\hat{g}(x) \leq \hat{g}(y)$  ak  $x$  je synom  $y$ . Pritom sa snažíme ísť stále hlbšie a hlbšie a do vedľajšieho podstromu sa už nikdy nedostaneme. Keby  $\hat{g}(x)$  bola reálna hodnota a nie len odhad, bol by to dobrý prístup. Preto  $f(x) \neq 0$  dáva možnosť prechodu do vedľajšieho podstromu.

Vyhľadávanie, ktoré zo živých vrcholov vyberá podľa minimálnej hodnoty  $\hat{c}(x)$  sa volá **LC-SEARCH**.

BFS  $\hat{g}(x) = 0$ ,  $f(h(x)) =$  úroveň  $x$   
 DFS  $f(h(x)) = 0$ ,  $\hat{g}(x) < \hat{g}(y)$  pre  $x$  dieťa  $y$

**Vlastnosti LC Branch and Bound**

Často riešime optimalizačné úlohy. Vieme pomocou metódy LC-B&B dosiahnuť minimum (optimum)?

*vlastnosti metódy  
LC-B & B*

Uvažujme nasledovnú funkciu

$$c(\mathbf{x}) = \begin{cases} \text{cena cesty z koreňa do } x, & \text{ak } x \text{ je list s odpoveďou} \\ \infty, & \text{ak } x \text{ je list bez odpovede} \\ \min\{c(y) \mid y \text{ je list stromu s koreňom } x\}, & \text{ak } x \text{ je vnútorný vrchol} \end{cases}$$

Ľahko vidno, že ak by LC-B&B používalo pri rozhodovaní funkciu  $c(x)$ , dosiahli by sme optimálne riešenie. Keďže (z hľadiska zložitosti) efektívny výpočet  $c()$  pri riešení ťažkých problémov nemáme, používame odhad  $\hat{c}(x)$ . Použitie  $\hat{c}(x)$  namiesto  $c(x)$  vo všeobecnosti optimálne riešenie nedáva. Platí ale nasledujúca veta.

**Veta 3.1**  $\hat{c}(x)$  nech je odhad  $c(x)$  taký, že

$$\hat{c}(x) < \hat{c}(z) \Leftrightarrow c(x) < c(z)$$

Potom algoritmus LC používajúci  $\hat{c}(x)$  namiesto  $c(x)$  nájde riešenie minimálnej ceny a skončí.

Ohraničovanie

ohraničovanie

Ak používame odhad  $\hat{c}(x)$  taký, že  $\hat{c}(x) \leq c(x)$ , tak  $\hat{c}(x)$  vlastne dáva dolný odhad. Ak by sme mali aj horný odhad  $U$ , tak všetky živé vrcholy s  $\hat{c}(x) > U$  možno vylúčiť. Ako získame  $U$ ?

- heuristika
- začínajúc s  $\infty$  vylepšujeme podľa toho, čo dosahujeme

Ukážeme použitie pre optimalizačnú úlohu. Ako  $c$  budeme používať účelovú funkciu, resp. ako  $\hat{c}(x)$  jej odhad.

### 3.3 Problém obchodného cestujúceho.

Použitie metódy LC-B & B ilustrujeme na probléme obchodného cestujúceho. O tomto probléme je známe, že je ťažký<sup>2</sup>. Daný je ohodnotený orientovaný graf reprezentujúci mestá so vzdialenosťami. Treba prejsť cez všetky mestá, v každom, s výnimkou štartovacieho, treba byť práve raz. Zo štartovacieho mesta vyjdeme a potom sa doň vrátíme. Treba minimalizovať dĺžku prejdenej cesty, pričom pod dĺžkou cesty myslíme súčet ohodnotení hrán, ktoré tvoria cestu. Bez ujmy na všeobecnosti môžeme predpokladať, že cesta začína a končí v meste č.1.

Aké hodnoty pre  $\hat{c}, c, U$ ?

$\mathbf{c(A)}$

- ak  $A$  je list, potom  $c(A)$  je dĺžka cesty definovanej cestou z koreňa do listu  $A$
- ak  $A$  nie je list, tak  $c(A)$  je cena minimálnej ceny listu v podstrome s koreňom  $A$

Jednoduchý odhad  $\hat{c}(x)$  je založený na tzv. redukovanej matici. Hovoríme, že

- riadok(stĺpec) matice je redukovaný, ak obsahuje aspoň jednu 0 a ostatné prvky sú nezáporné;
- **matice je redukovaná**, ak každý riadok a stĺpec, ktorý obsahuje aspoň jeden prvok rôzny od  $\infty$ , je redukovaný.

Prečo nás zaujíma redukovaná matice? Nech  $A$  je matice cien grafu. Uvedomme si, že každá cesta obchodného cestujúceho (OC) obsahuje práve jednu hranu  $A(i, j)$  pre  $i = k$  a práve jednu hranu  $A(i, j)$  pre  $j = k$ . Preto keď odpočítame konštantu  $t$  od každého prvku v jednom riadku (resp. stĺpci) matice cien  $A$ , každá cesta OC sa skrúti práve o  $t$ . Nemení sa ale relatívny vzťah ciest. Minimálna cesta ostáva minimálnou.

Každému bodu stavového priestoru priradíme redukovanú maticu a cenu nasledovným spôsobom:

<sup>2</sup>Tento problém je NP-úplný, čo znamená, že máme dosť dôvodov sa domnievať, že preň neexistuje polynomiálny algoritmus.

**Algoritmus 9** LC-B&B - Redukcia

Nech  $A$  je (aktuálna) matica cien grafu,  $r$  označuje cenu redukcie

- 1: nájdi minimum  $r_i$  v riadku  $i = 1, \dots, n$ . Nech je to prvok  $A(i, j)$  na pozícii  $(i, j)$
- 2:  $r \leftarrow r + r_i$
- 3: odpočítaj  $r_i$  od každého prvku v riadku  $i$ . Tým na mieste  $A(i, j)$  vznikne hodnota 0
- 4: ak nevznikla redukovaná matica, postupuj analogicky ďalej po tých stĺpcoch, ktoré nie sú redukované, až kým nezískaš redukovanú maticu

- Koreňu priradíme maticu  $red(A)$ , ktorá vznikla z matice cien vyššie popísaným Algoritmom 9. Cena tohto vrchola je cena redukcie, ktorou sme vyrobili redukovanú maticu v koreni. Uvedomme si, že uvedená hodnota odpovedá tomu, že sa snažíme z každého vrchola odísť po najkratšej hrane.
- Nech  $O$  je otec a  $S$  jeho syn taký, že odpovedá zaradeniu hrany  $(i, j)$  do cesty OC. Zaradenie hrany  $(i, j)$  do cesty OC znamená, že
  - nesmieme viac použiť hranu odchádzajúcu z  $i$ . Preto každý prvok v  $i$ -tom riadku nastavíme na  $\infty$
  - nesmieme viac použiť hranu prichádzajúcu do  $j$ . Preto každý prvok v stĺpci  $j$  nastavíme na  $\infty$
  - ak ešte nechceme ísť do 1, nastavíme  $A(j, 1) = \infty$

Takto vznikla nová matica cien  $A_{i,j}$ , ktorá nemusí byť redukovaná. Zredukujeme ju, čím získame maticu  $red(A_{i,j})$ , ktorú priradíme vrcholu  $S$ . Nech  $r(i, j)$  je cena redukcie matice  $A_{i,j}$  na  $red(A_{i,j})$  (redukujeme riadky a stĺpce okrem tých, ktoré obsahujú samé  $\infty$ ).

Potom  $\widehat{c}(S) = \widehat{c}(O) + A(i, j) + r(i, j)$

Postupujeme teda tak, že vypočítame redukovanú maticu a cenu pre každého syna (branch) a pohneme sa do toho syna, ktorý má minimálnu cenu (LC).

**Poznámka:** Namiesto písania hodnôt  $\infty$  môžeme odpovedajúci riadok a stĺpec z matice odstrániť. V tomto prípade bude matica odpovedať už len podgrafu indukovanému doteraz nezaradených vrcholov.

Analogicky môžeme zvážiť rozhodnutie o nezaradení hrany  $(i, j)$  do cesty OC.

- ak  $X$  je otec a  $P$  jeho syn taký, že odpovedá nezaradeniu hrany  $(i, j)$  do cesty OC, tak maticu  $A$  musíme modifikovať nasledovne
  - keďže hranu  $(i, j)$  viac nemáme použiť, nastavíme  $A(i, j) = \infty$  redukcia.

Takto vznikla nová matica  $B$ , ktorú musíme zredukovať. Uvedomme si, že budeme redukovať len v tom prípade, ak pôvodná hodnota  $A(i, j) = 0$ . Redukovanú maticu  $B'$  priradíme vrcholu  $P$  a cenou bude  $\widehat{c}(P) = \widehat{c}(X) + r(i, j)$ , kde  $r(i, j)$  je cena redukcie potrebná na získanie redukovanej matice  $B'$ .

Teraz už môžeme uvažovať aj iné stratégie

- vyber hranu, ktorej nezaradenie vedie k maximálnej cene
- vyber hranu, pri ktorej je maximálny rozdiel medzi cenou zaradenia  $\widehat{c}(L)$  a cenou  $\widehat{c}(P)$  nezaradenia danej hrany do cesty OC

### 3.4 0/1 Plnenie batoha

Vysvetlili sme, ako metódu LC-B&B používame na riešenie minimalizačných problémov. Je možné ju použiť aj na riešenie problém 0/1 plnenia batoha, ktorý je maximalizačný? Ľahko vidno, že áno. Maximalizovať  $\sum_{i=1}^n p_i x_i$  je ekvivalentné minimalizovaniu  $-\sum_{i=1}^n p_i x_i$ .

Použijeme statickú reprezentáciu stavového priestoru (zaradenie  $x_i$  odpovedá 1-hrane, nezaradenie  $x_i$  0-hrane). Zrejme

$$c(\mathbf{x}) = \begin{cases} -\sum_{i=1}^n x_i p_i, & \text{ak list } x \text{ reprezentuje prípustné riešenie} \\ \infty, & \text{ak list } x \text{ neodpovedá prípustnému riešeniu} \\ \min\{c(0syn(x)), c(1syn(x))\}, & \text{ak } x \text{ je vnútorný vrchol} \end{cases}$$

Budeme používať dva odhady -  $\hat{c}(x)$  a  $u(x)$  - také, že

$$\hat{c}(x) \leq c(x) \leq u(x)$$

Ako tieto odhady získame? Ak  $x$  je vrchol na úrovni  $j$ , tak cesta z koreňa doň určuje nastavenie pre  $x_i, 1 \leq i < j$ . Zrejme

$$c(x) \leq -\sum_{i=1}^{j-1} x_i p_i$$

a preto ako horný odhad určite môžeme použiť  $u(x) = -\sum_{i=1}^{j-1} x_i p_i$ . Vylepšený horný odhad získame algoritmom 10; pre  $q = -\sum_{1 \leq i < j} p_i x_i$  použijeme

$$u(x) = UBOUND(q, \sum_{1 \leq i < j} w_i x_i, j-1, M)$$

---

#### Algoritmus 10 UBOUND(p,w,k,M)

---

```

P ← p; W ← w
for i ← k + 1 to n do
    if W + w_i ≤ M then W ← W + w_i; P ← P - p_i
return(P)

```

---

K určeniu dolného odhadu  $\hat{c}(x)$  si všimnime Algoritmus 11. Horný odhad na profit je získaný simulovaním greedy prístupu k riešeniu (maximalizačnej verzii) problému za predpokladu, že  $\frac{p_i}{w_i} \geq \frac{p_{i+1}}{w_{i+1}}$ . Ľahko vidno, že

$$-BOUND(-q, \sum_{1 \leq i < j} w_i x_i, j-1, M) \leq c(x)$$

preto ako dolný odhad použijeme

$$\hat{c}(x) = -BOUND(-q, \sum_{1 \leq i < j} w_i x_i, j-1, M)$$

---

**Algoritmus 11** BOUND( $p, w, k, M$ )

---

 $p$  aktuálny profit $w$  aktuálny súčet váh $k$  index posledne odstráneného prvku $M$  veľkosť batoha

výsledkom je nový profit

 $P \leftarrow p; W \leftarrow w$ **for**  $i \leftarrow k$  to  $n$  **do** $W \leftarrow W + w_i$ **if**  $W < M$  **then**  $P \leftarrow P + p_i$ **else** return  $\left( P + \left( 1 - \frac{W-M}{w_i} \right) \times p_i \right)$ return( $P$ )

---