Řešení úloh

(Teorie a algoritmy hledání řešení úloh)

18. – 25. 2. 2015

Řešení úloh

První základní otázka: Co je to úloha ?
Jak lze úlohu definovat ?

Odpověď:

Mějme dány dvě množiny stavů:

$\mathfrak{X} = \{ x_1, x_2, \ldots x_K \}$ … množina výchozích stavů

$\mathfrak{Y} = \{ y_1, y_2, \ldots y_M \}$ … množina cílových stavů

Řešením úlohy pak rozumíme postup (posloupnost operací), kterým (kterými) převedeme úlohu z některého výchozího stavu x_i do definovaného cílového stavu y_j.

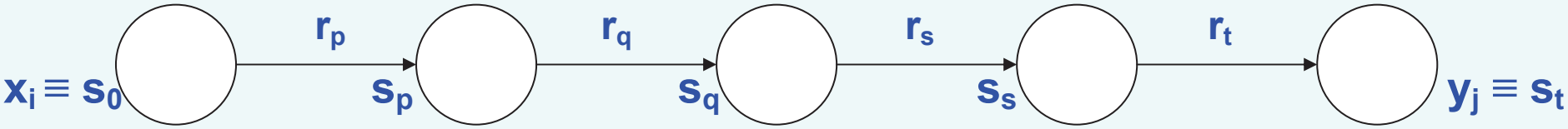
Obecně úlohu definujeme jako zobrazení

$$\mathfrak{X} \rightarrow \mathfrak{Y} \ .$$

Máme-li definován jen jeden konkrétní výchozí a jeden cílový stav úlohy, nabývá uvedené zobrazení tvar

$$x_i \rightarrow y_j \ .$$

Grafem reprezentujeme úlohu jako posloupnost stavů



s_p, s_q, s_s jsou vnitřní stavy (mezistavy) úlohy,

r_p, r_q, r_s,... jsou operátory popisující elementární operace.

Definujme množinu elementárních operátorů

$$RULES = \{ \mathbf{r}_1, \mathbf{r}_2, \ldots \mathbf{r}_{L-1}, \mathbf{r}_L \} .$$

Pak naši úlohu z předchozího obrázku lze vyjádřit

$$x_i \xrightarrow{R_{Kij}} y_j ,$$

kde $R_{Kij} = \mathbf{r}_p\, \mathbf{r}_q\, \mathbf{r}_s\, \mathbf{r}_t$ je kompoziční operátor,

$\mathbf{r}_p, \mathbf{r}_q, \mathbf{r}_s, \mathbf{r}_t \in RULES$, p,q,s,t $\in \{ 1, \ldots, L \}$.

Je-li více výchozích a cílových stavů, píšeme

$$\mathfrak{X} \xrightarrow{\mathfrak{R}} \mathfrak{Y} ,$$

$\mathfrak{R} = \{ R_{Kij} \}$ … množina všech kompozičních operátorů.

Definice: Úlohou potom nazveme trojici

$$(\mathfrak{X} , \mathfrak{Y} , \mathfrak{R}) ,$$

v níž vždy známe dvě složky a třetí určujeme.

Podle neznámé složky rozlišujeme úlohy:

- $(\mathfrak{X} , ? , \mathfrak{R})$ … deduktivní
- $(? , \mathfrak{Y} , \mathfrak{R})$ … abduktivní
- $(\mathfrak{X} , \mathfrak{Y} , ?)$ … induktivní

Pozn.: Abduktivní úloha poskytuje exaktní řešení pouze tehdy, je-li R_{Kij} bijektivní zobrazení (pro všechna $R_{Kij} \in \mathfrak{R}$).

Hledání řešení úlohy – hledání („sestavení“) takového kompozičního operátoru R_{Kij}, který vyhovuje zadaným množinám stavů $\mathfrak{X}$ a $\mathfrak{Y}$ a je v nějakém (obvykle daném) smyslu optimální.

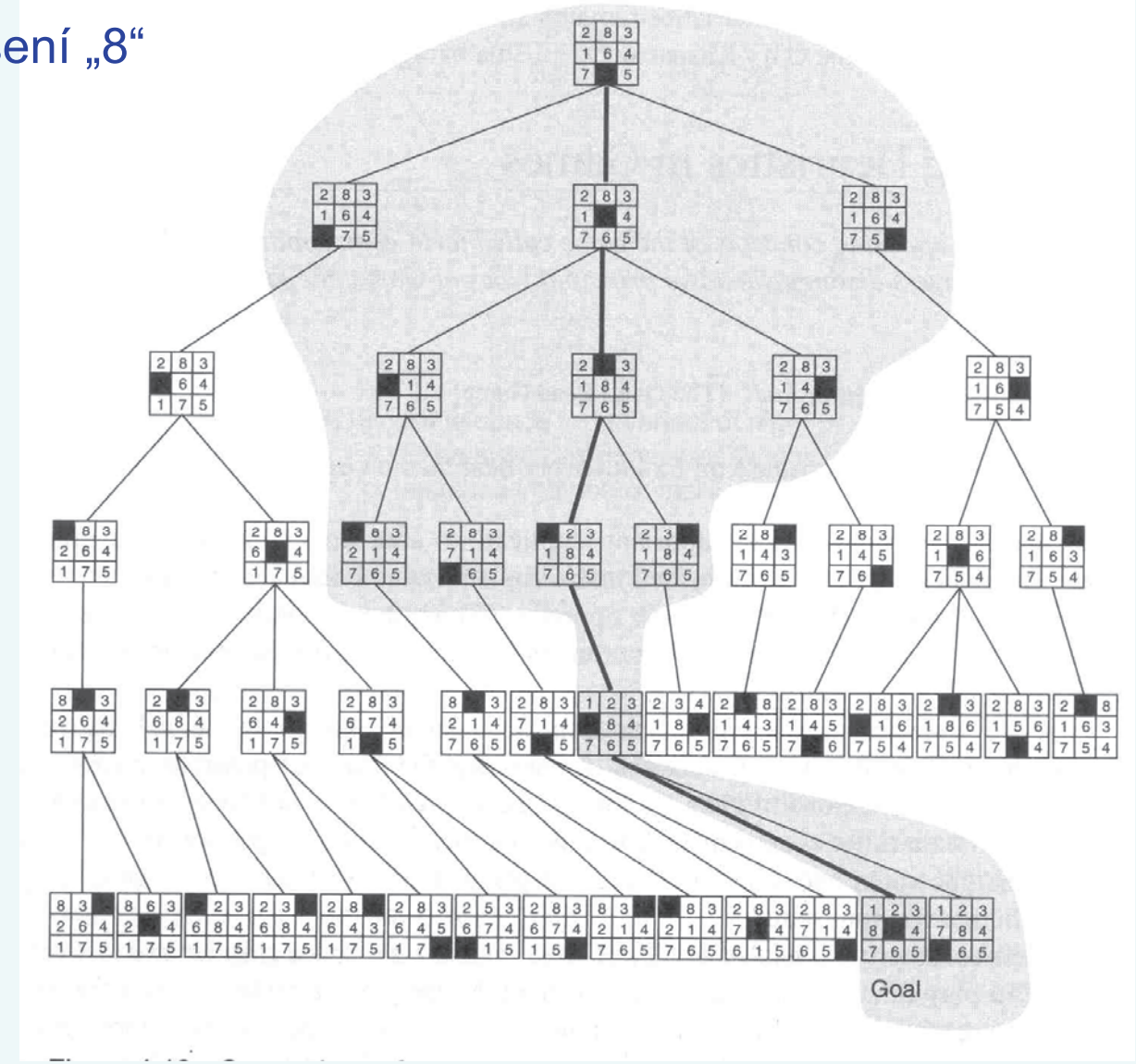
Postup:

Metodou pokusů a omylů vytváříme strom řešení úlohy a současným jeho prohledáváním hledáme takový kompoziční operátor R_{Kij}, který vyhovuje množinám stavů $\mathfrak{X}$ a $\mathfrak{Y}$.

Procházení (prohledávání) stromu řešení:

- deterministické
- náhodné
- heuristické

Příklad: Strom řešení „8“



Příklad: Strom řešení omezených „piškvorek“ – tic-tac-toe (první tři úrovně)

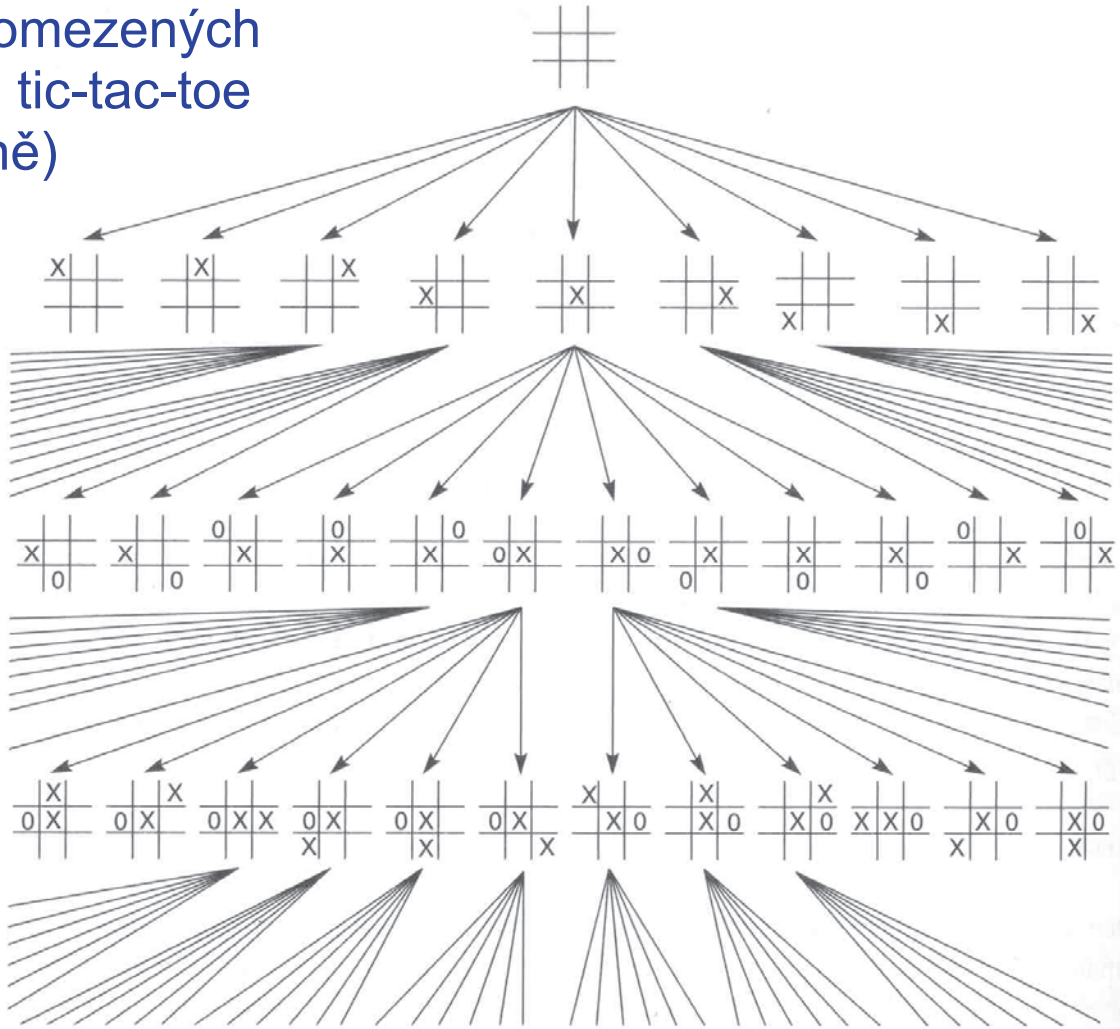


Figure II.5 Portion of the state space for tic-tac-toe.

Metoda reprezentace úlohy ve stavovém prostoru

Předpoklady:

1. Existuje konečná množina stavů, v nichž se úloha může nacházet:
   $S = \{ s_i \}, \quad i = 0, 1, \ldots, N$.
2. Existuje alespoň jeden výchozí (počáteční) stav úlohy $s_0 \in S$.
3. Existuje konečná množina cílových (požadovaných) stavů úlohy
   $G = \{ g_j \}, \quad j = 1, 2, \ldots, M$ taková, že $G \subseteq S$, $M \leq N$.
4. Existuje konečná množina elementárních operátorů
   $RULES = \{ r_l \}, \quad l = 1, 2, \ldots, L$,
   které převádějí úlohu ze stavu s_p do stavu s_q , $p, q = 0, 1, \ldots, N$.

Potom:

Stavový prostor úlohy je definován dvojicí

$$(\mathbf{S} , \mathit{RULES}) .$$

Konkrétní řešení úlohy je definováno trojicí

$$(\mathbf{s}_0 , \mathbf{s}_j , R_{Koj}) \text{ na } \mathbf{S} ,$$

kde R_{Koj} je kompoziční operátor, který převede úlohu z počátečního stavu $\mathbf{s}_0$ do stavu cílového $\mathbf{s}_j = \mathbf{g}_k$, čili

$$\mathbf{s}_0 \xrightarrow{R_{Koj}} \mathbf{s}_j = \mathbf{g}_k \in \mathbf{G} , \quad j = 0, 1, \ldots, N , \quad k = 1, \ldots, M .$$

Poznámka: Nachází-li se úloha na počátku řešení ve stavu, který je žádaným cílovým stavem, lze řešení úlohy formálně zapsat

$$\mathbf{s}_0 \xrightarrow{R_{Koo}} \mathbf{s}_0 = \mathbf{g}_1$$

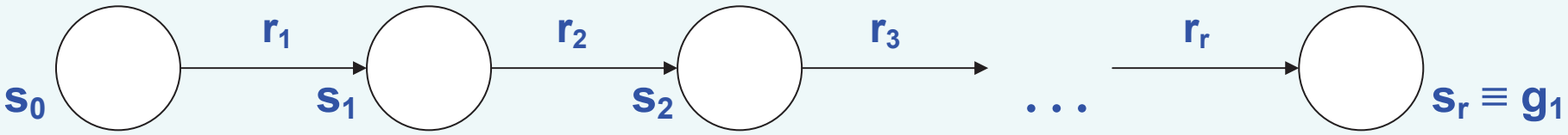
a R_{Koo} budiž prázdný kompoziční operátor, čili $R_{Koo} = \mathbf{r}_\varepsilon$, kde $\mathbf{r}_\varepsilon$ je prázdná elementární operace.

Řešením úlohy ve stavovém prostoru pak budiž kompoziční operátor

$$R_{Koj} = \mathbf{r}_1\,\mathbf{r}_2\,\mathbf{r}_3 \ldots \mathbf{r}_{r-1}\,\mathbf{r}_r$$

takový, že

$$\mathbf{s}_1 \leftarrow \mathbf{r}_1(\mathbf{s}_0),$$
$$\mathbf{s}_2 \leftarrow \mathbf{r}_2(\mathbf{s}_1) = \mathbf{r}_2(\mathbf{r}_1(\mathbf{s}_0)), \quad \text{(zapíšeme } \mathbf{r}_2\,\mathbf{r}_1(\mathbf{s}_0)\text{)}$$
$$\ldots,$$
$$\mathbf{s}_r \leftarrow \mathbf{r}_r(\mathbf{s}_{r-1}) = \mathbf{r}_r(\mathbf{r}_{r-1}(\ldots\mathbf{r}_1(\mathbf{s}_0))).$$



Hledání řešení úlohy

Př.:

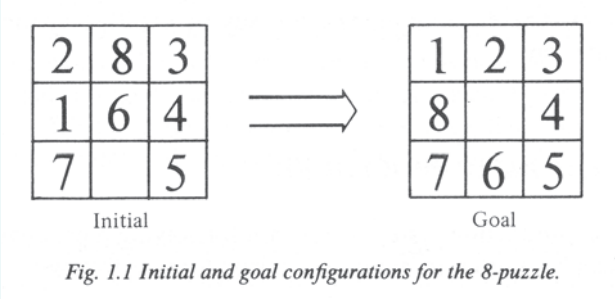
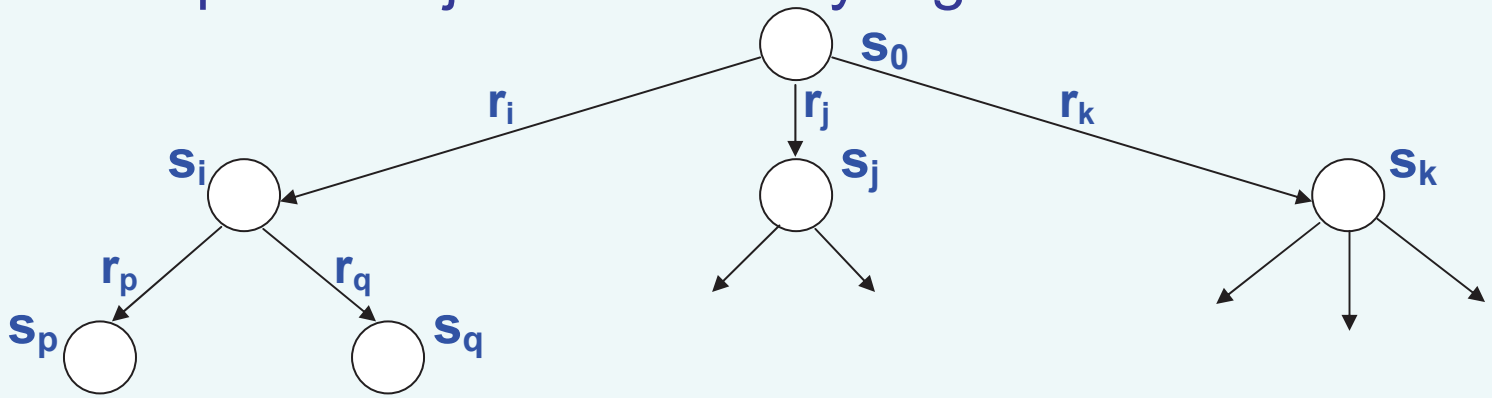


Fig. 1.1 Initial and goal configurations for the 8-puzzle.

Úlohu reprezentujeme stromovým grafem



kde uzly grafu reprezentují stavy úlohy,
hrany grafu reprezentují přechody mezi stavy (aplikace elementárních operátorů)

Definice: Uzel, do něhož vede bezprostřední hrana, je nazýván bezprostředním následovníkem – uzly s_i , s_j , s_k jsou bezprostředními následovníky uzlu s_0 , uzly s_p , s_q jsou bezprostředními následovníky uzlu s_i atd.

Definice: Nalezení všech bezprostředních následovníků uzlu s_k se nazývá expanzí uzlu s_k .

Definice: Počet hran vedoucích z uzlu s_0 do uzlu s_p definuje hloubku uzlu s_p .

STROM ŘEŠENÍ ÚLOHY

Definice:

Strom řešení úlohy je orientovaný graf definovaný takto:

1. Graf má jediný uzel bez bezprostředního předchůdce – **kořen**. Tento uzel reprezentuje **výchozí stav** úlohy.
2. U každého uzlu definujeme rekurzivně **hloubku** uzlu:
   - kořen má hloubku **0** ,
   - má-li uzel hloubku **d** , má každý jeho bezprostřední následovník hloubku **d+1** .
3. Uzly, které nemají žádného bezprostředního následovníka (listy), reprezentují
   - **cílový stav** úlohy,
   - stavy, na něž nelze aplikovat žádný z operátorů,
   - stavy, o nichž jsme usoudili, že jsou z nějakých důvodů neperspektivní (jejich další rozvíjení nemá smysl).

4. Orientovaná hrana reprezentuje přechod úlohy z daného (aktuálního) stavu do stavu nového – **aplikaci elementárního operátoru**.

Aplikaci všech možných elementárních operátorů na stav úlohy, jíž získáme všechny možné následující stavy úlohy (bezprostřední následovníky), nazveme **expanzí uzlu**.

Nalezení řešení úlohy = nalezení cesty spojující počáteční uzel grafu řešení úlohy (výchozí stav, kořen grafu) s listem reprezentujícím cílový (žádaný) stav úlohy.

Ale: Jak (čím) najdeme, resp. určíme, řešení úlohy ?

PRODUKČNÍ SYSTÉM

Produkční systém se skládá z:

- **databáze** úlohy obsahující **fakta**
- **báze znalostí** obsahující **produkční pravidla** ve tvaru

  podmínka ⟶ akce

- **řídicího mechanismu** – jehož úkolem je:
  - provést volbu, které aplikovatelné pravidlo bude použito,
  - vybrat fakta z databáze, která budou dosazena do podmínky zvoleného produkčního pravidla,
  - ukončit řešení (výpočet), je-li splněna cílová podmínka
- **množiny cílů**, které mají být splněny

Cílová podmínka produkčního systému:

a) explicitní, odvozená z množiny cílů,

b) implicitní, nejde-li na daný obsah databáze aplikovat žádné další produkční pravidlo.

Pracovní režimy řídicího mechanismu:

- přímochody
- zpětnochody

Přímochodý režim:

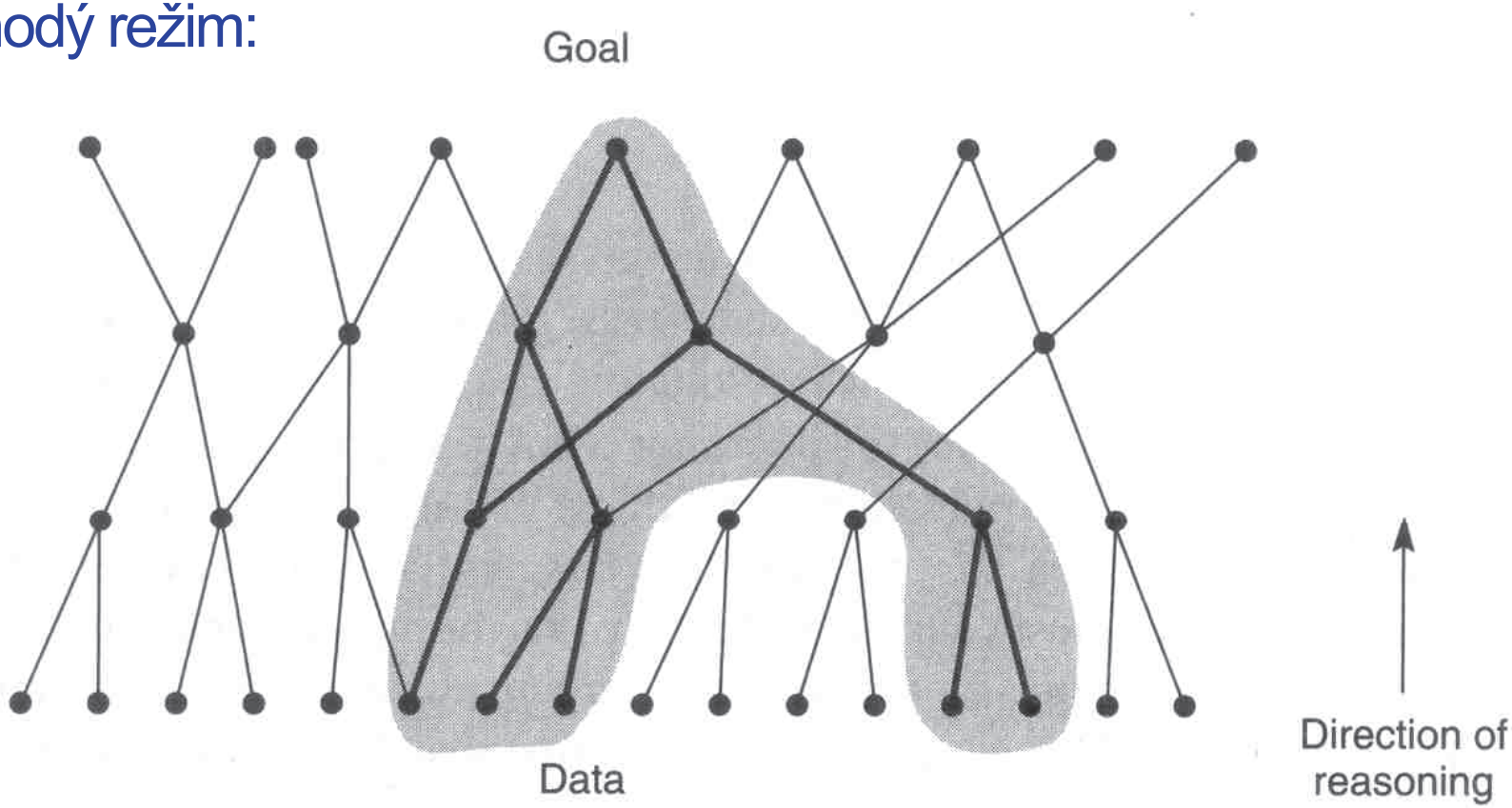


Figure 3.13 State space in which data-directed search prunes irrelevant data and their consequents and determines one of a number of possible goals.

Zpětnochodý režim:

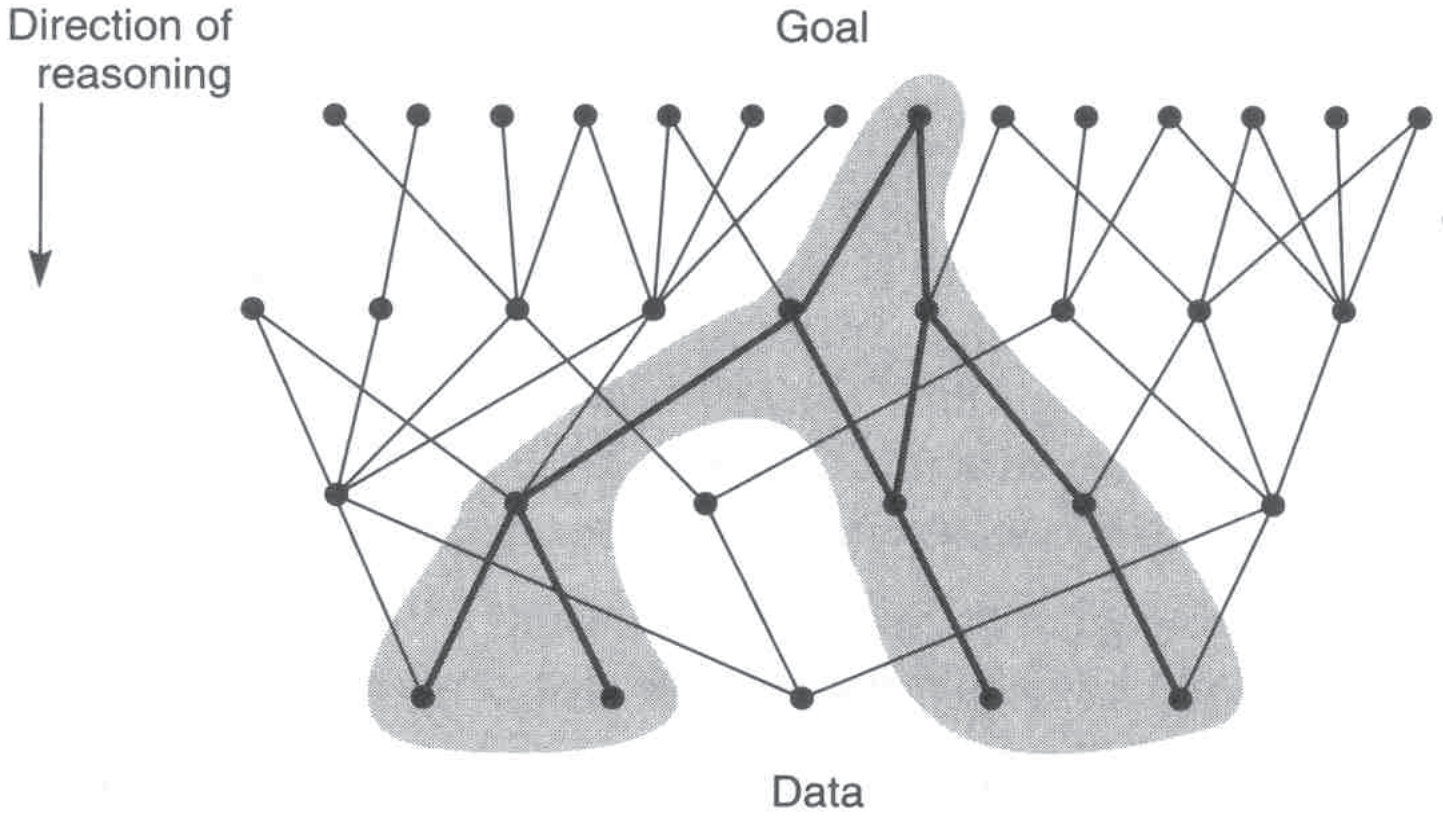


Figure 3.12 State space in which goal-directed search effectively prunes extraneous search paths.

ZÁKLADNÍ PROCEDURA PRODUKČNÍHO SYSTÉMU

$DATA \longleftarrow$ výchozí stav ; { poč. stav databáze }

repeat

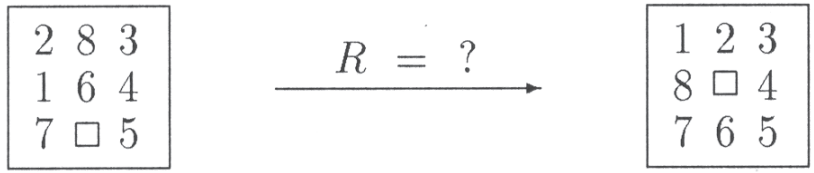
  select (nějaké) pravidlo $r_i \in RULES$, které je aplikovatelné na $DATA$;

  $DATA \longleftarrow r_i\,(DATA)$; { aplikace r_i na $DATA$ }

until $DATA$ splňují cílovou podmínku ;

INDETERMINISMUS

Př.: Hlavolam „8“



Pravidla hry můžeme obecně zapsat jako

if podmínka **then** akce

nebo ve tvaru

podmínka → akce .

Pravidla pro přesouvání kamenů (prázdné políčko nazvěme „blank“):

"blank" není v horním řádku	⟶	posuň "blank" nahoru
"blank" není v dolním řádku	⟶	posuň "blank" dolů
"blank" není v levém sloupci	⟶	posuň "blank" doleva
"blank" není v pravém sloupci	⟶	posuň "blank" doprava

ŘÍDICÍ MECHANISMY (řídicí strategie)

- neodvolatelné (irrevocable)
- pokusné (tentative)

Př.: Výběr pravidel gradientní metodou (hill climbing algorithm):

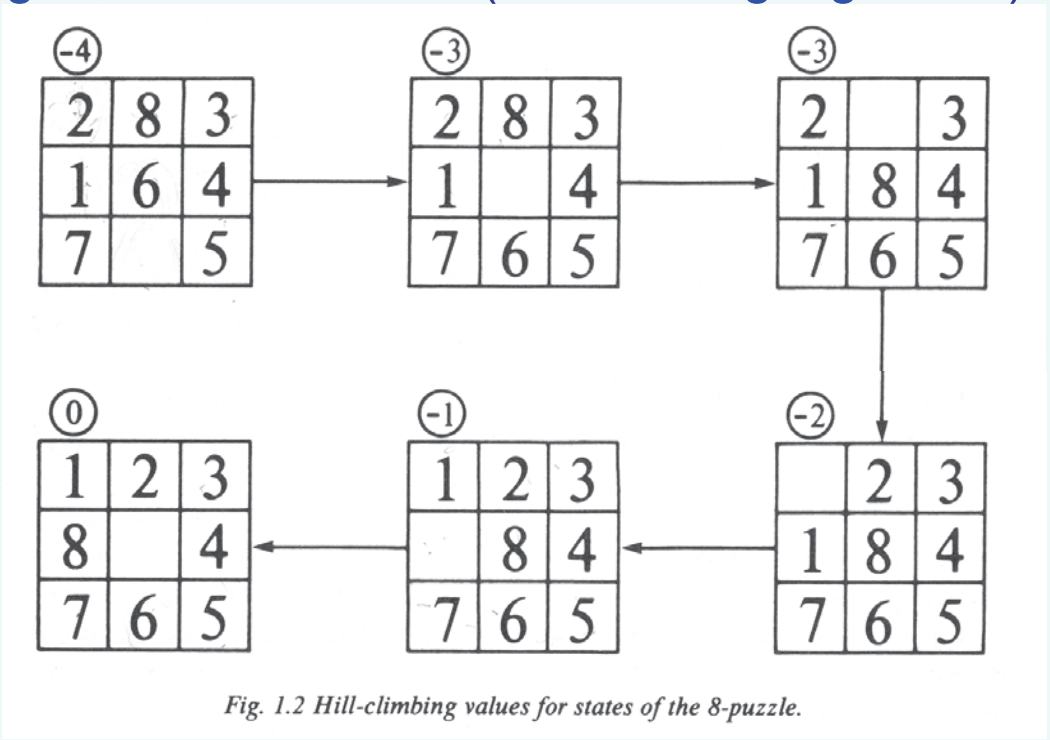


Fig. 1.2 Hill-climbing values for states of the 8-puzzle.

Pokusné řídicí strategie

- metoda (algoritmus) navracení (backtracking)
- metody hledání v grafu řešení úlohy
  - slepé strategie (neinformované)
  - informované strategie (zpravidla heuristické)

ALGORITMUS NAVRACENÍ (BACKTRACKING)

deterministický – prod. pravidla se aplikují ve stanoveném pořadí

- backtracking stavový
- backtracking operátorový (Floydův)

2. Řešení úloh

Recursive procedure **BACKTRACK** (*DATA*)
{*DATA* reprezentuje databázi příslušného stavu řešení}

1. **if TERM** (*DATA*) **then return** *NIL*;

   { **TERM** je logická funkce (predikát) nabývající pravdivostní hodnoty **true** v případě, že obsah databáze *DATA* splňuje cílovou podmínku implementovaného produkčního systému. Jako příznak úspěšného ukončení hledání řešení je procedurou vracen *NIL*, resp. prázdný seznam. }

2. **if DEADEND** (*DATA*) **then return** *FAIL*;

   { **DEADEND** je logická funkce nabývající hodnoty **true** v případě, že mechanismus se dostal do chybového (fail) stavu (podmínky viz výše) a nelze dále úspěšně pokračovat vpřed k hledanému cíli řešení. V takovém případě vrací procedura symbol *FAIL* jako příznak dosažení chybového stavu. }

3. *RULES* ← **APPRULES** (*DATA*);

   { **APPRULES** je funkce určující, která produkční pravidla a v jakém pořadí budou na daný obsah databáze *DATA* aplikována }

4. LOOP: **if NULL** (*RULES*) **then return** *FAIL*;

   { není-li žádné další aplikovatelné pravidlo, končí procedura chybovým (fail) stavem a vrací symbol *FAIL* }

5. $R \leftarrow$ **FIRST** (*RULES*);

   { aplikováno bude v pořadí první, resp. podle daného kritéria "nejlepší" pravidlo }

6. *RULES* $\leftarrow$ **TAIL** (*RULES*);

   { z posloupnosti aplikovatelných produkčních pravidel je vyjmuto zvolené pravidlo }

7. *NEW_DATA* $\leftarrow$ **R** (*DATA*);

   { na obsah databáze *DATA* je aplikováno produkční pravidlo **R** z posloupnosti *RULES* a je vygenerován nový obsah databáze *NEW_DATA* }

8. *PATH* $\leftarrow$ **BACKTRACK** (*NEW_DATA*);

   { procedura **BACKTRACK** je vyvolána rekursivně pro nový obsah databáze }

9. **if** *PATH* = *FAIL* **then** **goto** LOOP;

   { vrátilo-li vyvolání procedury **BACKTRACK** chybový příznak *FAIL*, přechází na použití dalšího produkčního pravidla (pokud takové existuje) }

10. **return** **CONS** (*R*, *PATH*);

    { v opačném (úspěšném) případě je přidáno nově aplikované produkční pravidlo na čelo seznamu (vrchol zásobníku) reprezentujícího vygenerovanou cestu ve stromu řešení z počátečního uzlu do uzlu aktuálního }

2. Řešení úloh

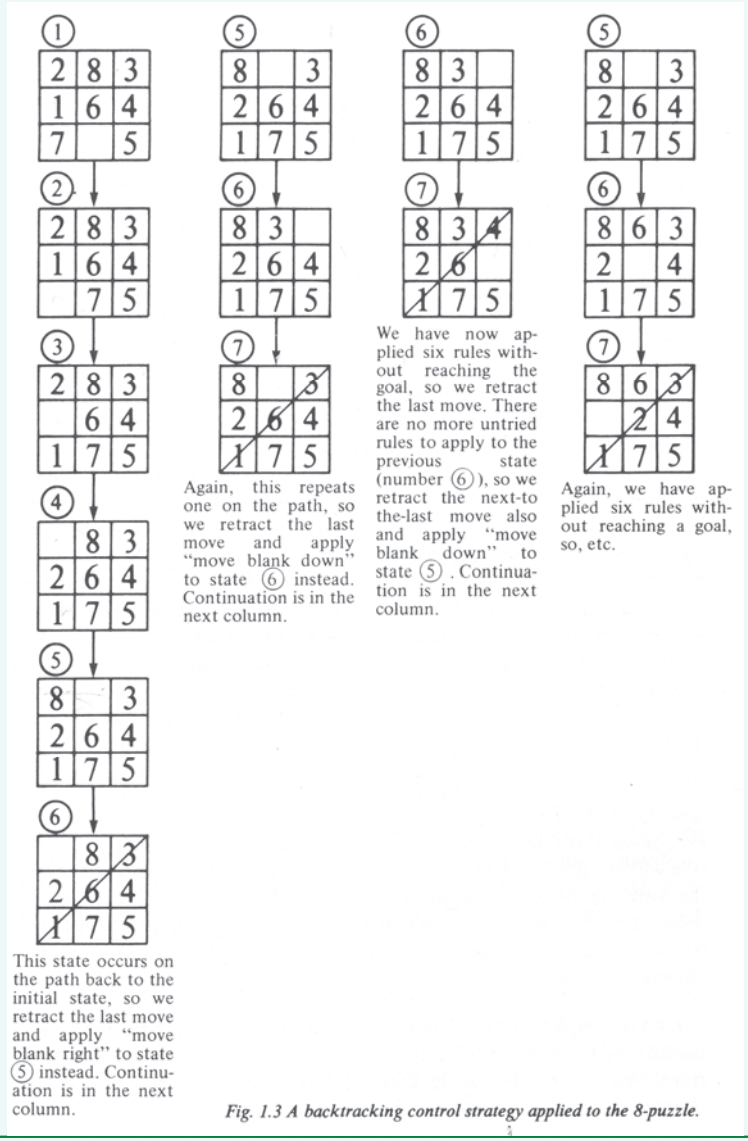


Fig. 1.3 A backtracking control strategy applied to the 8-puzzle.

2. Řešení úloh

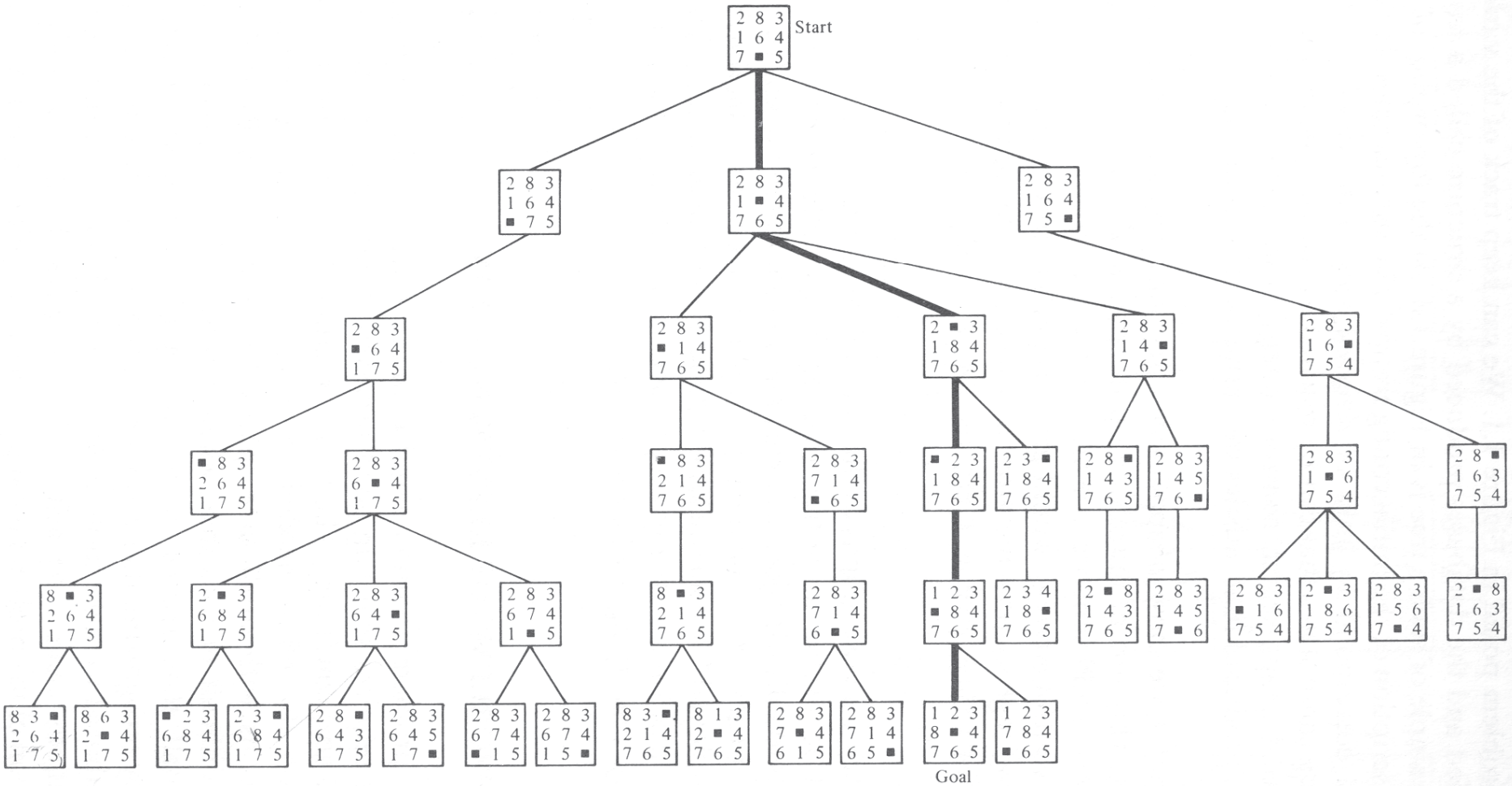


Fig. 1.4 A search tree for the 8-puzzle.

METODY HLEDÁNÍ V GRAFU (GRAPH SEARCH)

Zapamatovávají si celou dosud vygenerovanou část stromu řešení –

- jsou paměťově náročné,
- jsou použitelné účinné globální heuristiky.

Základem metod hledání v grafu jsou:

- expanze uzlu (aplikace všech možných produkčních pravidel na *DATA* příslušející danému uzlu grafu),
- ohodnocující funkce přiřazující uzlu ohodnocení – zpravidla heuristická *(globální heuristika)*.

Př.: **Problém obchodního cestujícího** – jak objet zákazníky v N místech s minimálními náklady, např.:

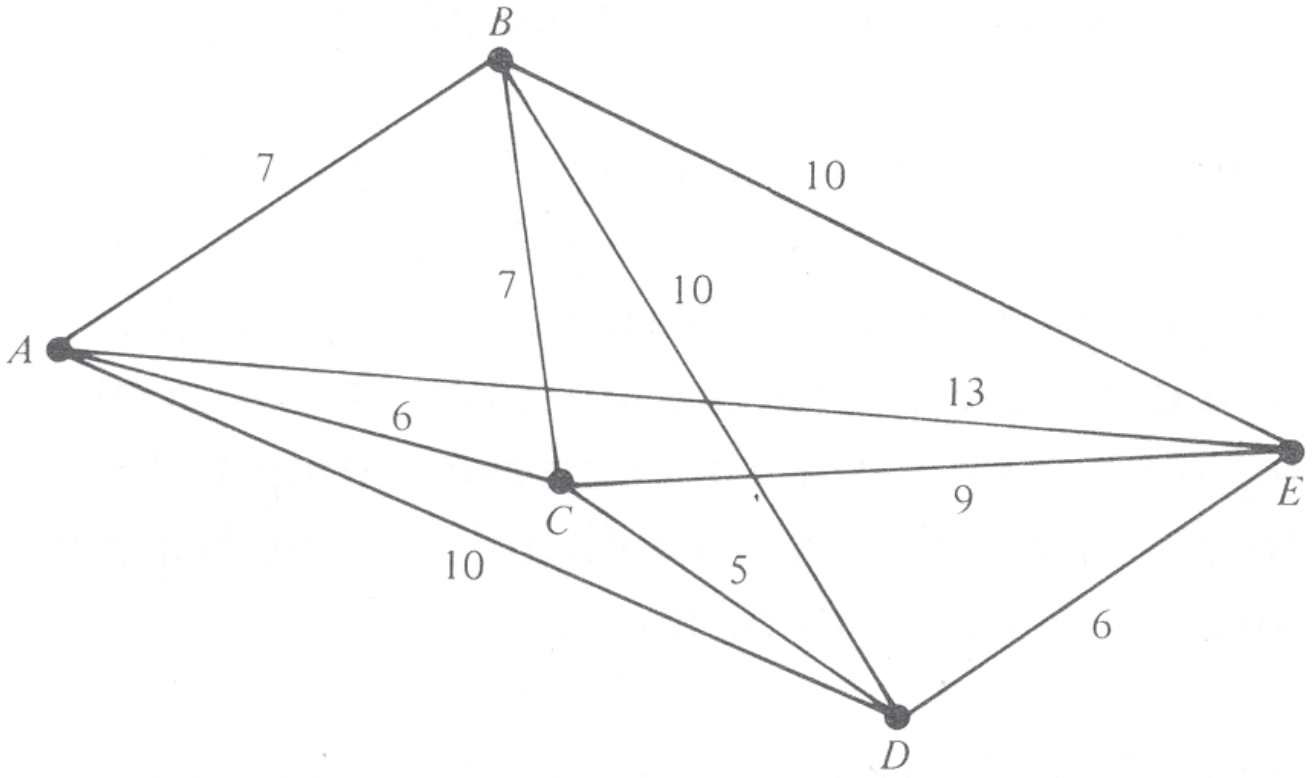


Fig. 1.5 A map for the traveling salesman problem.

Graf řešení úlohy obchodního cestujícího:

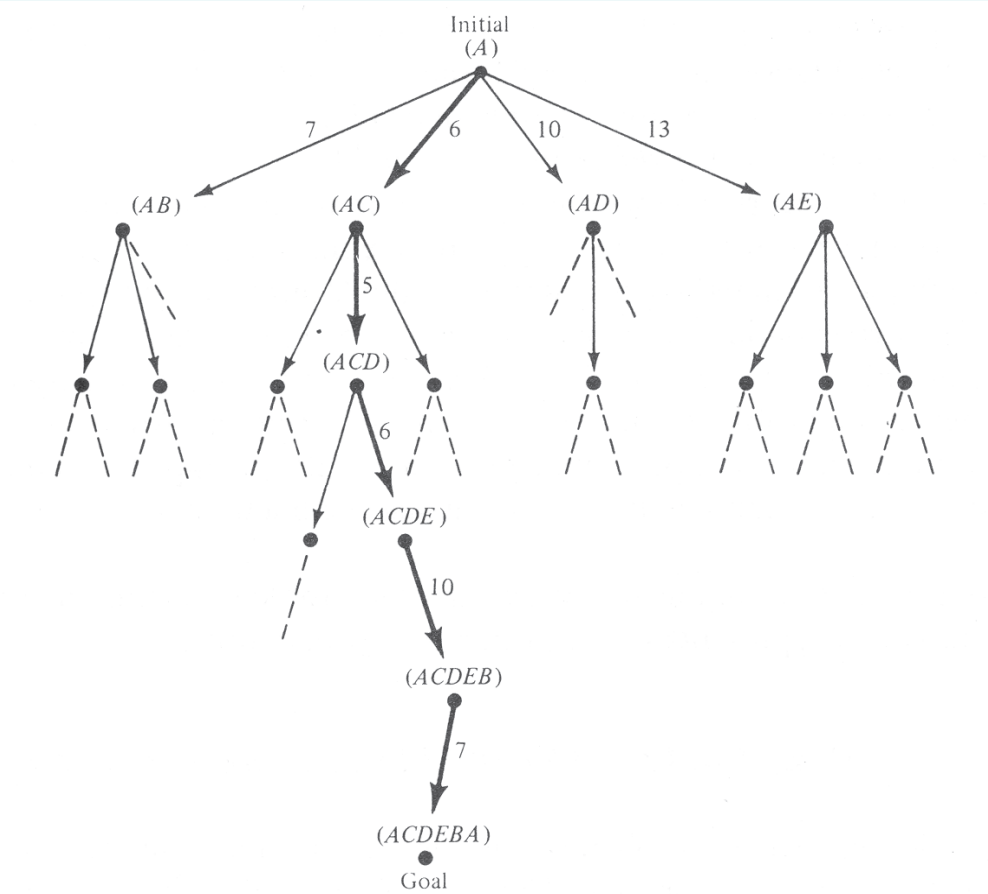


Fig. 1.6 A search tree for the traveling salesman problem.

SLEPÉ STRATEGIE HLEDÁNÍ ŘEŠENÍ ÚLOHY

a) do hloubky (depth first search)

Speciální případ základního algoritmu hledání v grafu, v němž platí:

- ohodnocení uzlu = hloubka uzlu,
- v kroku 3 ((***)) se bere maximum,
- v kroku 3 ((***)) je nutno přidat test, zda již bylo dosaženo zadané maximální hloubky; pokud ano, vracíme se ke kroku 2.

Př.: „8“

2. Řešení úloh

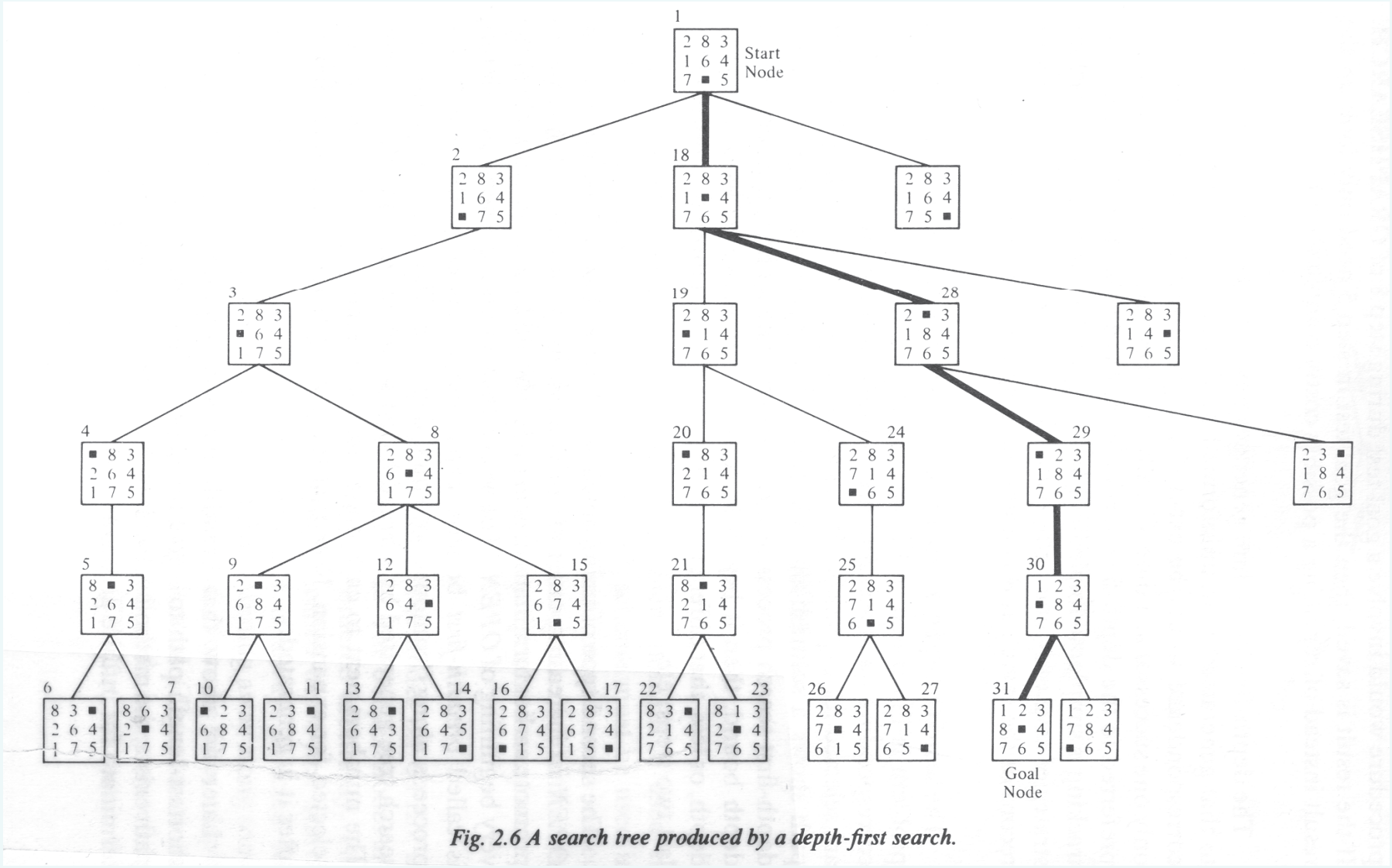


Fig. 2.6 A search tree produced by a depth-first search.

SLEPÉ STRATEGIE HLEDÁNÍ ŘEŠENÍ ÚLOHY

b) do šířky (breadth first search)

Speciální případ základního algoritmu hledání v grafu, v němž platí:

- ohodnocení uzlu = hloubka uzlu,
- v kroku 3 ((***)) se bere minimum.

Poznámka: Při prohledávání stromu řešení algoritmem prohledávání do šířky je vždy nalezena nejkratší cesta k cílovému uzlu, pokud tato cesta existuje.

Př.: „8“

2. Řešení úloh

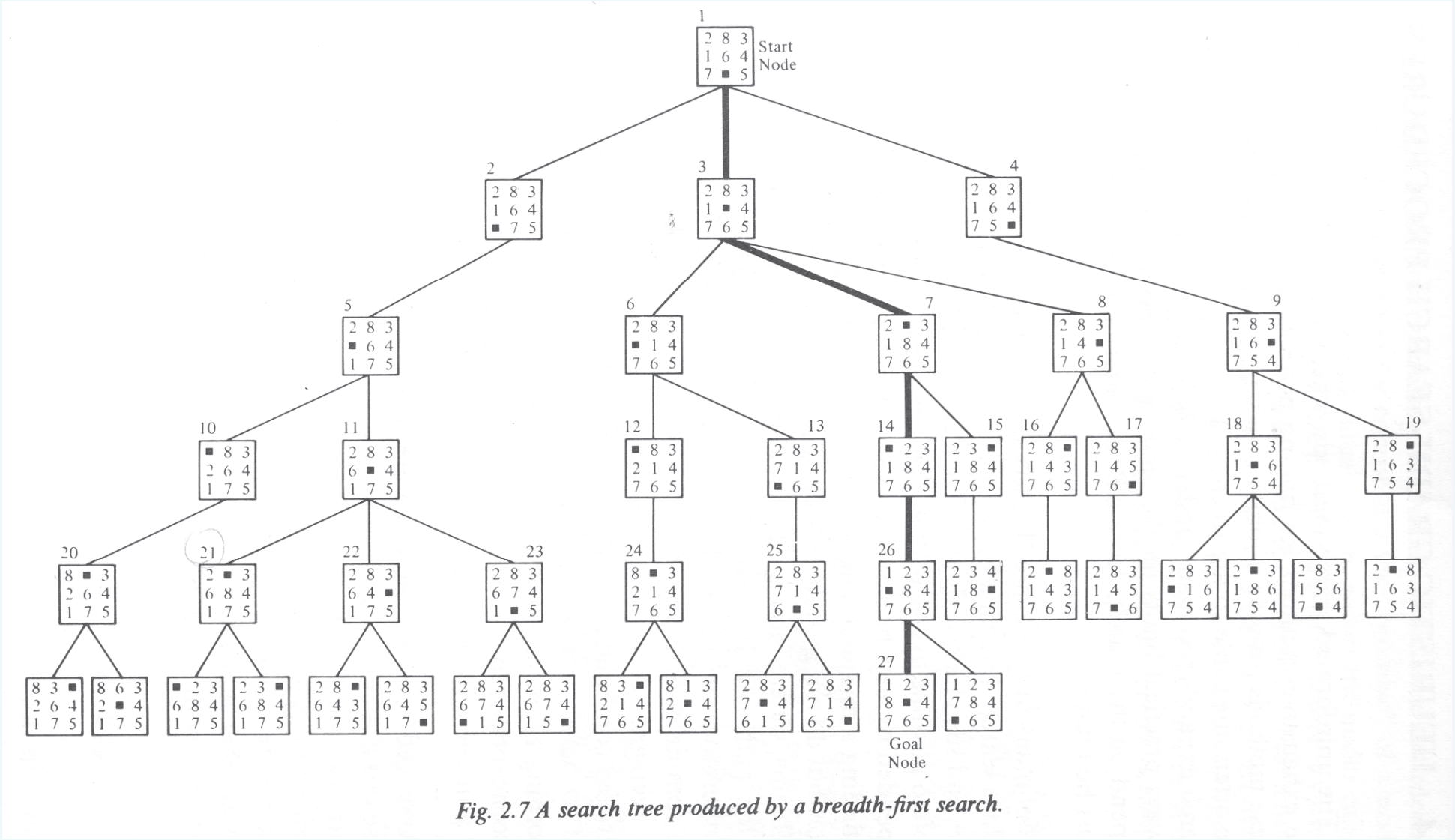


Fig. 2.7 A search tree produced by a breadth-first search.

ZDOKONALENÉ SLEPÉ STRATEGIE HLEDÁNÍ ŘEŠENÍ

Neinformované metody prohledávání nemají k dispozici žádné vhodné znalosti o stavovém prostoru, které by jim umožnily urychlit cestu k cíli. Jsou odsouzeny k systematickému procházení všech uzlů, dokud nenaleznou řešení. Jednotlivé algoritmy se od sebe liší jen způsobem, jakým toto systematické procházení provádějí, avšak lze je určitým způsobem zdokonalit. Ke zdokonaleným slepým strategiím patří:

- prohledávání do hloubky s omezením (depth-limited search)
- iterativní prohledávání do hloubky (iterative deepening search)
- obousměrné prohledávání (bidirectional search)
- prohledávání se stejnou (uniformní) cenou (uniform-cost search)
- prohledávání naslepo (blind search)
- prohledávání metodou rozděl a panuj (divide and conquer search)

a) Prohledávání do hloubky s omezením

Omezené prohledávání do hloubky se od klasického prohledávání do hloubky liší pouze v tom, že je zadána maximální hloubka, do které se prohledává. Uzly této hloubky se již dále nerozvíjejí. Tím lze eliminovat problémy s nekonečnou větví.

Tato jednoduchá strategie má dvě slabiny:

- Při výběru nevhodné cesty se může pouze po dlouhé době ukázat, že nevede k cíli.
- Toutéž slepou uličkou můžeme projít vícekrát, jestliže k ní vede více cest, protože si nepamatujeme prošlé cesty; nebrání tedy cyklům v posloupnosti stavů a není systematická.

Na druhé straně je výhodná pro použití např. u úlohy obchodního cestujícího, když je na mapě 20 měst, a my víme, že nejkratší cestu z města *A* do města *B* není nutné vést přes více než 18 měst.

b) Iterativní prohledávání do hloubky

Iterativní prohlubování je vlastně omezené prohledávání do hloubky, pro které se postupně zvětšuje maximální hloubka, do které se prohledává. Prohledává se tedy (do hloubky) nejprve stavový prostor hloubky 1, pak do hloubky 2, pak do hloubky 3 atd. Tato varianta dokáže odstranit i druhý problém klasického prohledávání do hloubky, neboť nalezne optimální řešení (pokud existuje).

Výhody algoritmu iterativního vyhledávání:

- Jedná se o kompromis mezi výhodami a nevýhodami vyhledávání do šířky a do hloubky.
- Je nejvhodnější neinformovanou metodou pro velké stavové prostory s neznámou hloubkou, ve které se nachází řešení.

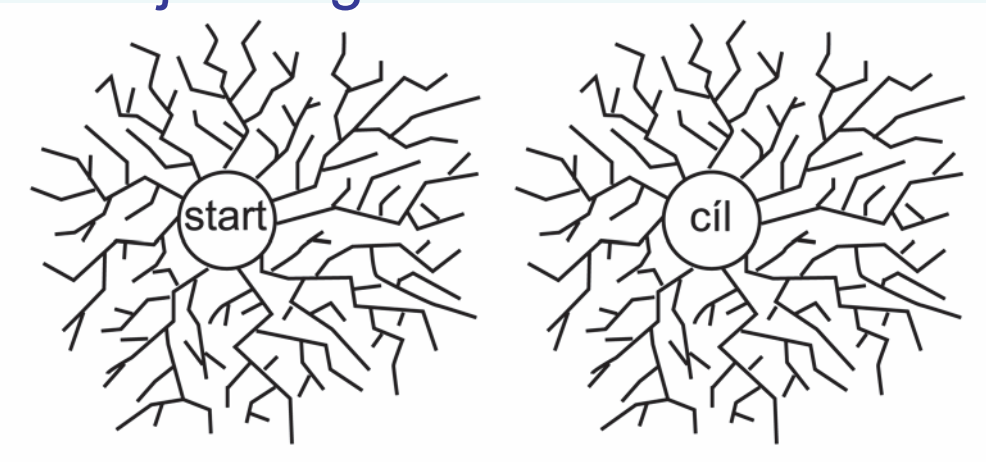
Nevýhody algoritmu iterativního vyhledávání:

- Nevíme, jaký určit limit pro hloubku vyhledávání, a proto musíme vyzkoušet všechny hloubky počínaje nulovou.

Příklad:

c) Obousměrné prohledávání

Je jednoduchým vylepšením algoritmu vyhledávání do šířky, které si všímá toho, že počet prozkoumaných políček roste kvadraticky s délkou cesty. Proto rozjíždí simultánně dvě vyhledávání do šířky. Jedno ze startu, druhé z cíle. Postupně provádí vždy jeden cyklus jedné, a jeden cyklus druhé šířky. Ve chvíli, kdy se šířky setkají, je možné zkonstruovat cestu ze startu do cíle. Pokud jedna z šířek dojde do stavu, kdy už nemá nová políčka na prozkoumání, pak cesta v mapě neexistuje a algoritmus může skončit.



Vlastnosti vyhledávání do šířky v obou směrech:

- Paměťová i časová náročnost je závislá vzhledem k počtu políček mapy.
- Hledáme současně z počátečního i cílového stavu.
- Hledání skončí, když na druhém konci najdeme uzel, který si pamatujeme z prvního konce.

Výhody vyhledávání do šířky v obou směrech:

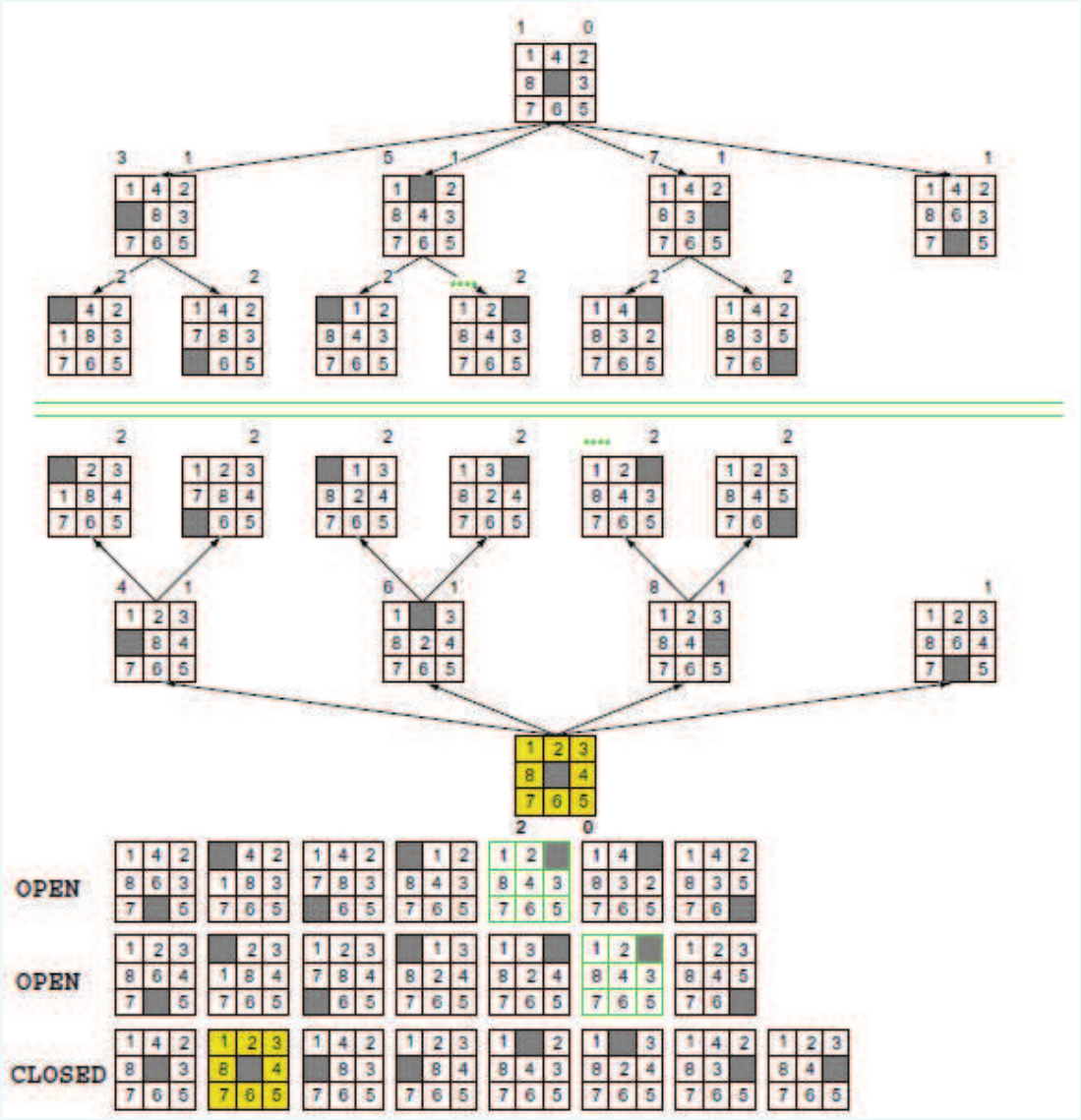
- Rychlý algoritmus, protože počet prozkoumaných políček roste kvadraticky s délkou cesty.

Nevýhody vyhledávání do šířky v obou směrech:

- Paměťové nároky jsou mnohem vyšší než u vyhledávání do šířky.

2. Řešení úloh

Příklad:



d) Prohledávání se stejnou (uniformní) cenou

V každém kroku se expanduje uzel, který má nejmenší akumulovanou cenu (cenu cesty z výchozího do tohoto uzlu). Jde o zobecnění slepého prohledávání do šířky (prohledávání do šířky je prohledávání s jednotnou cenou, kde cenou je hloubka uzlu).

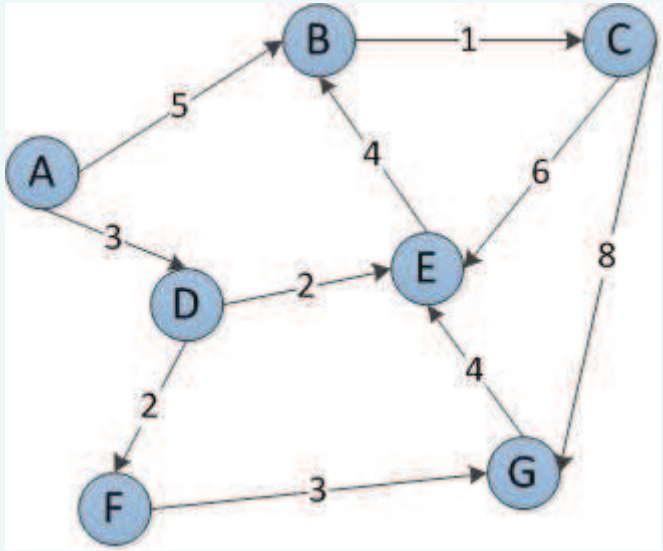
```
procedure UniformCostSearch(Graph, root, goal)
  node := root, cost = 0
  frontier := priority queue containing node only
  explored := empty set
  do
    if frontier is empty
      return failure
    node := frontier.pop()
    if node is goal
      return solution
    explored.add(node)
    for each of node's neighbors n
      if n is not in explored
```

```
if n is not in frontier
  frontier.add(n)
else if n is in frontier with higher cost
  replace existing node with n
```

Example

Expansion showing the explored set and the frontier (priority queue):
Start Node: A
Goal Node: G

Step	Frontier: a set of (node, its cost) objects	Expand[*]	Explored: a set of nodes
1	{(A,0)}	A	∅
2	{(D,3),(B,5)}	D	{A}
3	{(B,5),(E,5),(F,5)}	B	{A,D}
4	{(E,5),(F,5),(C,6)}	E	{A,D,B}
5	{(F,5),(C,6)}[**]	F	{A,D,B,E}
6	{(C,6),(G,8)}	C	{A,D,B,E,F}
7	{(G,8)}	G	{A,D,B,E,F,C}
8	∅		



[*] The node chosen to be expanded for the next step.
[**] B is not added to the frontier because it is found in the explored set.
Found the path: A to D to F to G.

Vlastnosti uniformního prohledávání:

- Jako hodnotící funkci použijeme nákladovou funkci: $f(i) = g(i)$.
- Vidíme, že slepé vyhledávání do šířky je speciálním případem prohledávání metodou stejných cen, a sice když náklady všech hran jsou jednotkové.

Výhody uniformního prohledávání:

- Vždy nalezne optimální řešení (za předpokladu, že řešení existuje).
- Vhodně se používá u úlohy obchodního cestujícího.

Nevýhody uniformního prohledávání:

- Je obecně málo efektivní, optimální řešení často nalezne za cenu expanze velkého počtu uzlů.