

2. Řešení úloh

Řešení úloh

(Teorie a algoritmy hledání řešení úloh)

8. 3. 2019

2. Řešení úloh

Řešení úloh

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

2. Řešení úloh

Řešení úloh

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

Odpověď:

Mějme dány dvě množiny stavů:

$\mathcal{X} = \{ x_1, x_2, \dots, x_K \}$... množina výchozích stavů

$\mathcal{Y} = \{ y_1, y_2, \dots, y_M \}$... množina cílových stavů

2. Řešení úloh

Řešení úloh

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

Odpověď:

Mějme dány dvě množiny stavů:

$\mathcal{X} = \{ x_1, x_2, \dots, x_K \}$... množina výchozích stavů

$\mathcal{Y} = \{ y_1, y_2, \dots, y_M \}$... 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 .

2. Řešení úloh

Obecně úlohu definujeme jako zobrazení

$$x \rightarrow y .$$

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

$$x_i \rightarrow y_j .$$

2. Řešení úloh

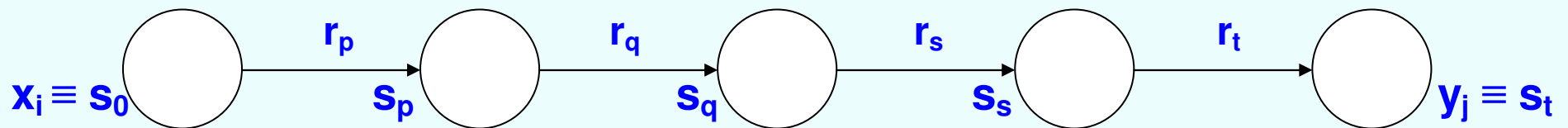
Obecně úlohu definujeme jako zobrazení

$$x \rightarrow y .$$

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, \dots jsou operátory popisující elementární operace.

2. Řešení úloh

Definujme množinu elementárních operátorů

$$RULES = \{ r_1, r_2, \dots, r_{L-1}, r_L \} .$$

Pak naši úlohu z předchozího obrázku lze vyjádřit

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

kde $R_{Kij} = r_p r_q r_s r_t$ je kompoziční operátor,

$$r_p, r_q, r_s, r_t \in RULES, \quad p, q, s, t \in \{ 1, \dots, L \} .$$

2. Řešení úloh

Definujme množinu elementárních operátorů

$$RULES = \{ r_1, r_2, \dots, r_{L-1}, r_L \} .$$

Pak naši úlohu z předchozího obrázku lze vyjádřit

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

kde $R_{Kij} = r_p r_q r_s r_t$ je kompoziční operátor,

$$r_p, r_q, r_s, r_t \in RULES, \quad p, q, s, t \in \{ 1, \dots, L \} .$$

Je-li více výchozích a cílových stavů, píšeme

$$\mathcal{X} \xrightarrow{\mathcal{R}} \mathcal{Y}, \quad \mathcal{R} = \{ R_{Kij} \} \dots \text{množina všech kompozičních operátorů.}$$

2. Řešení úloh

Definice: Úlohou potom nazveme trojici

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

v níž vždy známe dvě složky a třetí určujeme.

2. Řešení úloh

Definice: Úlohou potom nazveme trojici

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

v níž vždy známe dvě složky a třetí určujeme.

Podle neznámé složky rozlišujeme úlohy:

- $(\mathcal{X} , ? , \mathcal{R})$... deduktivní
- $(? , \mathcal{Y} , \mathcal{R})$... abduktivní
- $(\mathcal{X} , \mathcal{Y} , ?)$... induktivní

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

2. Řešení úloh

Umělá inteligence vyšetřuje skupinu induktivních úloh, u nichž člověk musí formulovanou hypotézu řešení zpětně deduktivně prověřit, zda vyhovuje zadaným množinám stavů x a y .

2. Řešení úloh

Umělá inteligence vyšetřuje skupinu induktivních úloh, u nichž člověk musí formulovanou hypotézu řešení zpětně deduktivně prověřit, zda **vyhovuje** zadaným množinám stavů x a y .

Dva způsoby výběru vhodné hypotézy z \mathcal{R} :

1. Apriorně zvolíme množinu operátorů, kterou **parametrizujeme** (nastavením vhodných parametrů vybereme takový kompoziční operátor, který **vyhovuje** zadaným množinám stavů x a y).

2. Řešení úloh

Umělá inteligence vyšetřuje skupinu induktivních úloh, u nichž člověk musí formulovanou hypotézu řešení zpětně deduktivně prověřit, zda vyhovuje zadaným množinám stavů \mathcal{X} a \mathcal{Y} .

Dva způsoby výběru vhodné hypotézy z \mathcal{R} :

1. Apriorně zvolíme množinu operátorů, kterou parametrizujeme (nastavením vhodných parametrů vybereme takový kompoziční operátor, který vyhovuje zadaným množinám stavů \mathcal{X} a \mathcal{Y}).
2. Apriorně zvolíme množinu elementárních operátorů, ze kterých se postupně snažíme „složit“ výsledný kompoziční operátor vyhovující zadaným množinám stavů \mathcal{X} a \mathcal{Y} .

Poznámka: Induktivní úlohy jsou zpravidla **nedeterministické** .

2. Řešení úloh

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

2. Řešení úloh

Hledání řešení úlohy – hledání („sestavení“) takového kompozičního operátoru R_{Kij} , který vyhovuje zadaným množinám stavů x a 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 x a y .

2. Řešení úloh

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

Postup:

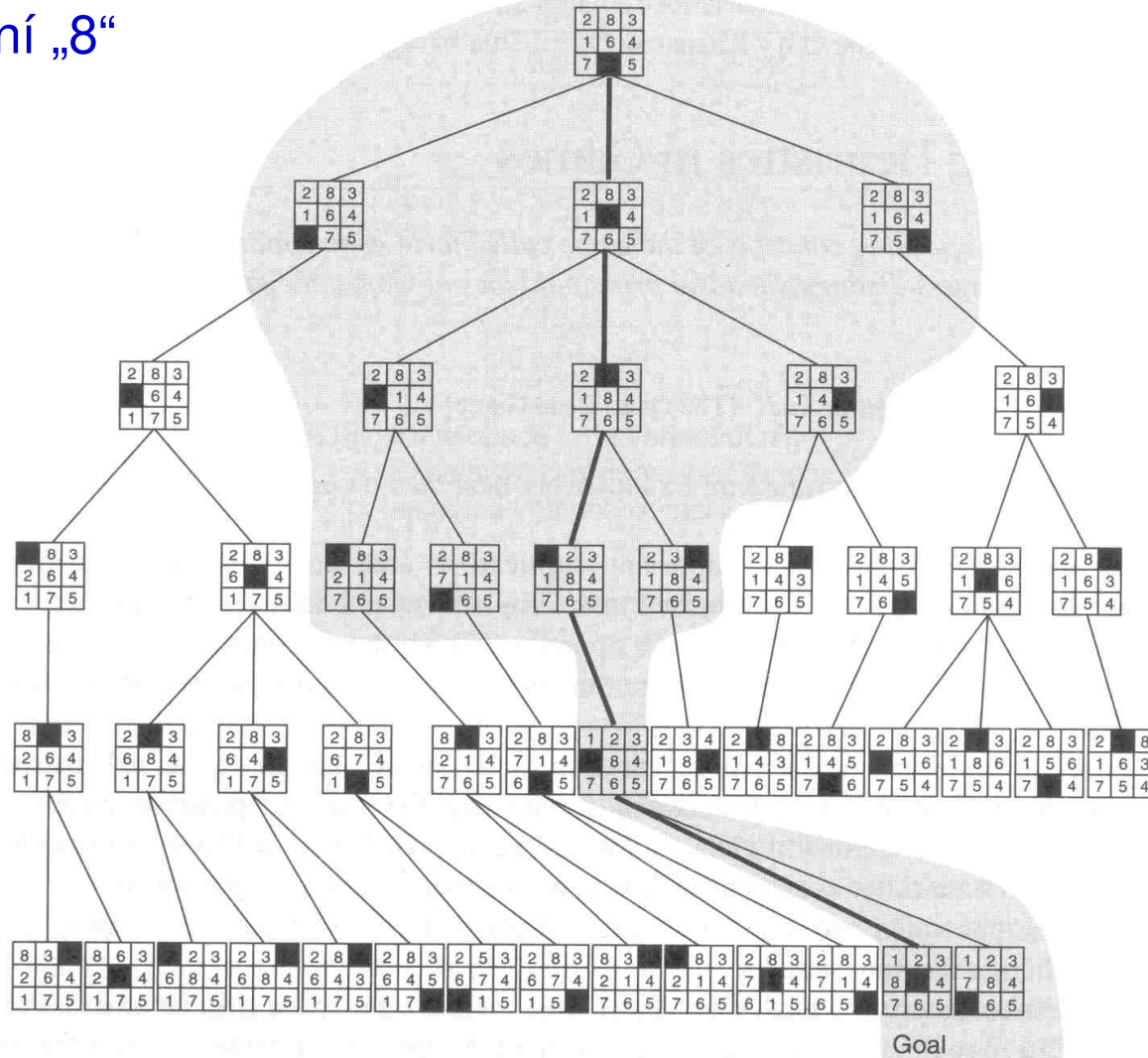
Metodou pokusů a omylů **vytváříme strom řešení** úlohy a současným jeho prohledáváním hledáme takový kompoziční operátor R_{Kij} , který vyhovuje množinám stavů x a y .

Procházení (prohledávání) stromu řešení:

- deterministické
- náhodné
- heuristické

2. Řešení úloh

Příklad: Strom řešení „8“



2. Řešení úloh

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

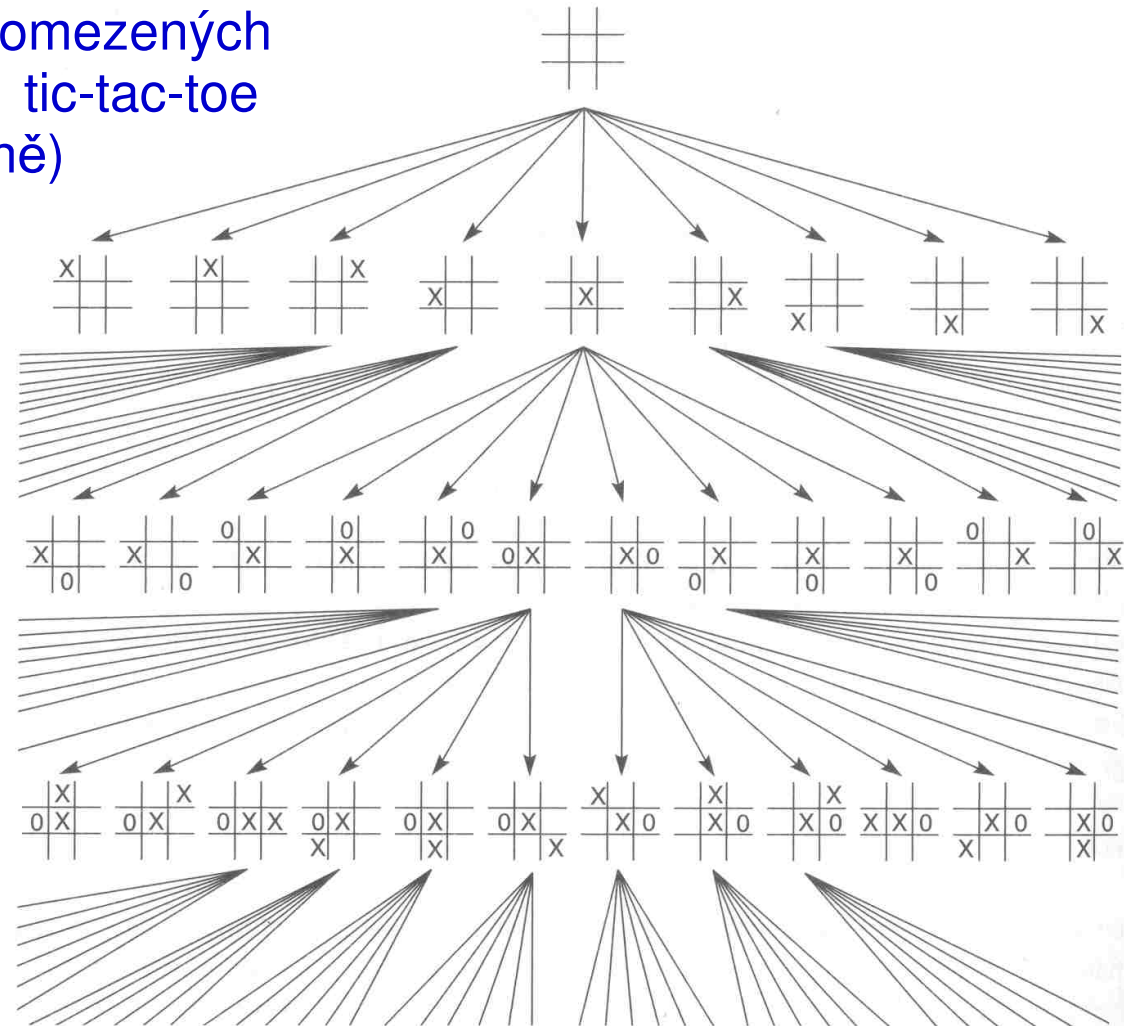


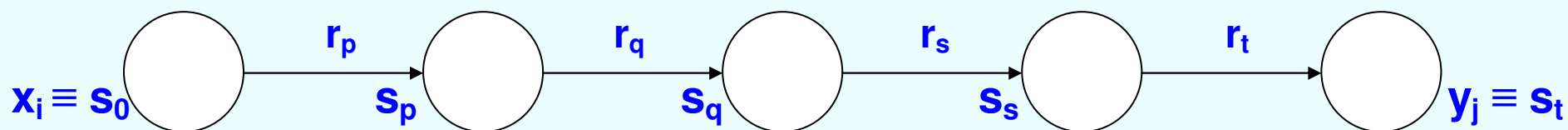
Figure II.5 Portion of the state space for tic-tac-toe.

2. Řešení úloh

Reprezentace úlohy

Úloha $(\mathcal{X}, \mathcal{Y}, ?)$ má zpravidla definovanou množinu výchozích stavů, v nichž se nachází, a množinu cílových stavů, jichž má být řešením úlohy dosaženo – úlohu budeme reprezentovat ve stavovém prostoru .

Řekli jsme, že grafem reprezentujeme úlohu jako posloupnost stavů,



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

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

Zobecněním dostaneme **metodu stavového prostoru** .

2. Řešení úloh

Metoda reprezentace úlohy ve stavovém prostoru

Předpoklady:

1. Existuje konečná množina stavů, v nichž se úloha může nacházet:

$$\mathbf{S} = \{ \mathbf{s}_i \}, \quad i = 0, 1, \dots, N.$$

2. Existuje alespoň jeden výchozí (počáteční) stav úlohy $\mathbf{s}_0 \in \mathbf{S}$.

3. Existuje konečná množina cílových (požadovaných) stavů úlohy

$$\mathbf{G} = \{ \mathbf{g}_j \}, \quad j = 1, 2, \dots, M \text{ taková, že } \mathbf{G} \subseteq \mathbf{S}, M \leq N.$$

4. Existuje konečná množina elementárních operátorů

$$\mathbf{RULES} = \{ \mathbf{r}_l \}, \quad l = 1, 2, \dots, L,$$

které převádějí úlohu ze stavu \mathbf{s}_p do stavu \mathbf{s}_q , $p, q = 0, 1, \dots, N$.

2. Řešení úloh

Potom:

Stavový prostor úlohy je definován dvojicí

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

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

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

Poznámka: Nachází-li se úloha na počátku řešení ve stavu, který je žadáným 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}_ε je prázdná elementární operace.

2. Řešení úloh

Řešením úlohy ve stavovém prostoru pak budiž kompoziční operátor

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

takový, že

$$\mathbf{s}_1 \leftarrow \mathbf{r}_1 (\mathbf{s}_0),$$

$$\mathbf{s}_2 \leftarrow \mathbf{r}_2 (\mathbf{s}_1) = \mathbf{r}_2 (\mathbf{r}_1 (\mathbf{s}_0)), \quad (\text{zapišeme } \mathbf{r}_2 \mathbf{r}_1 (\mathbf{s}_0))$$

\dots ,

$$\mathbf{s}_r \leftarrow \mathbf{r}_r (\mathbf{s}_{r-1}) = \mathbf{r}_r (\mathbf{r}_{r-1} (\dots \mathbf{r}_1 (\mathbf{s}_0))).$$

2. Řešení úloh

Řešením úlohy ve stavovém prostoru pak budiž kompoziční operátor

$$R_{Koj} = r_1 r_2 r_3 \dots r_{r-1} r_r$$

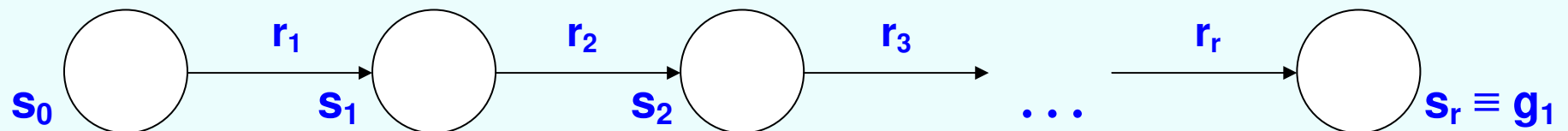
takový, že

$$\mathbf{s}_1 \leftarrow r_1 (\mathbf{s}_0),$$

$$\mathbf{s}_2 \leftarrow r_2 (\mathbf{s}_1) = r_2 (r_1 (\mathbf{s}_0)), \quad (\text{zapišeme } r_2 r_1 (\mathbf{s}_0))$$

\dots ,

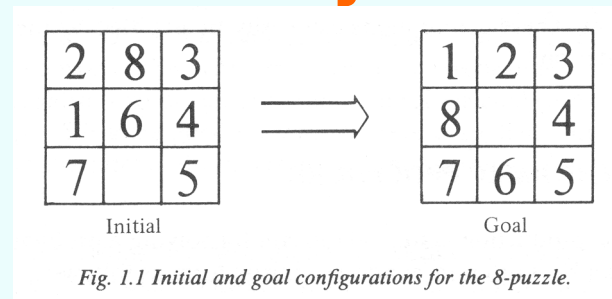
$$\mathbf{s}_r \leftarrow r_r (\mathbf{s}_{r-1}) = r_r (r_{r-1} (\dots r_1 (\mathbf{s}_0))).$$



2. Řešení úloh

Hledání řešení úlohy

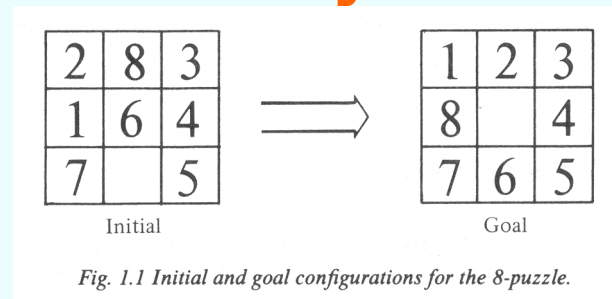
Př.:



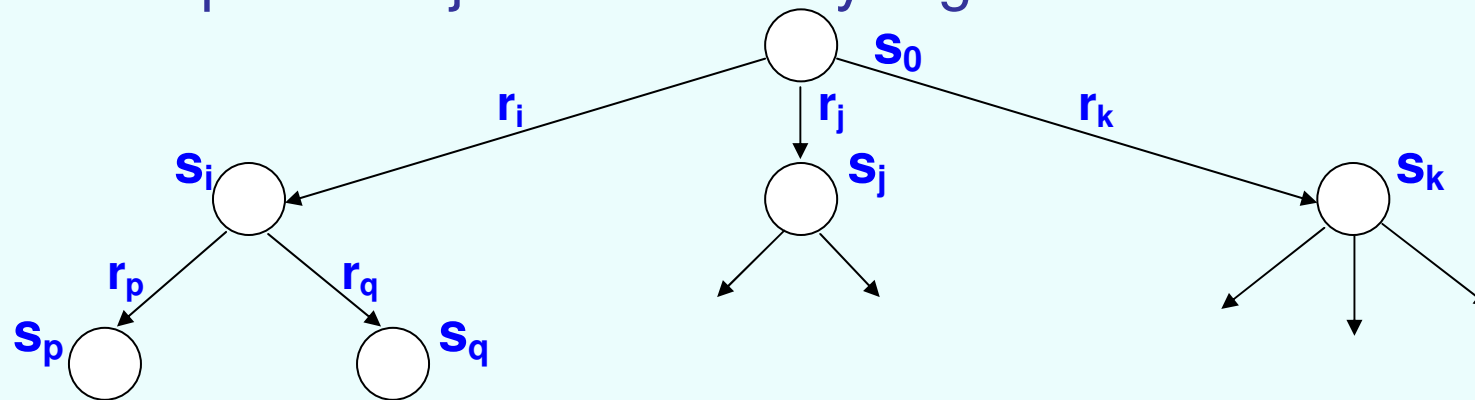
2. Řešení úloh

Hledání řešení úlohy

Př.:



Úlohu reprezentujeme stromovým grafem



kde **uzly grafu**
hrany grafu

reprezentují stavy úlohy,
reprezentují přechody mezi stavy (aplikace
elementárních operátorů)

2. Řešení úloh

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.

2. Řešení úloh

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** .

2. Řešení úloh

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).

2. Řešení úloh

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**.

2. Řešení úloh

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.

2. Řešení úloh

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 ?

2. Řešení úloh

PRODUKČNÍ SYSTÉM

Produkční systém se skládá z:

- **databáze** úlohy obsahující **fakta**
- **báze znalostí** obsahující **produkční pravidla** ve tvaru
$$\text{podmínka} \longrightarrow \text{akce}$$
- **řídícího 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

2. Řešení úloh

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:

- přímochodý
- zpětnochodý

2. Řešení úloh

Přímochodý režim:

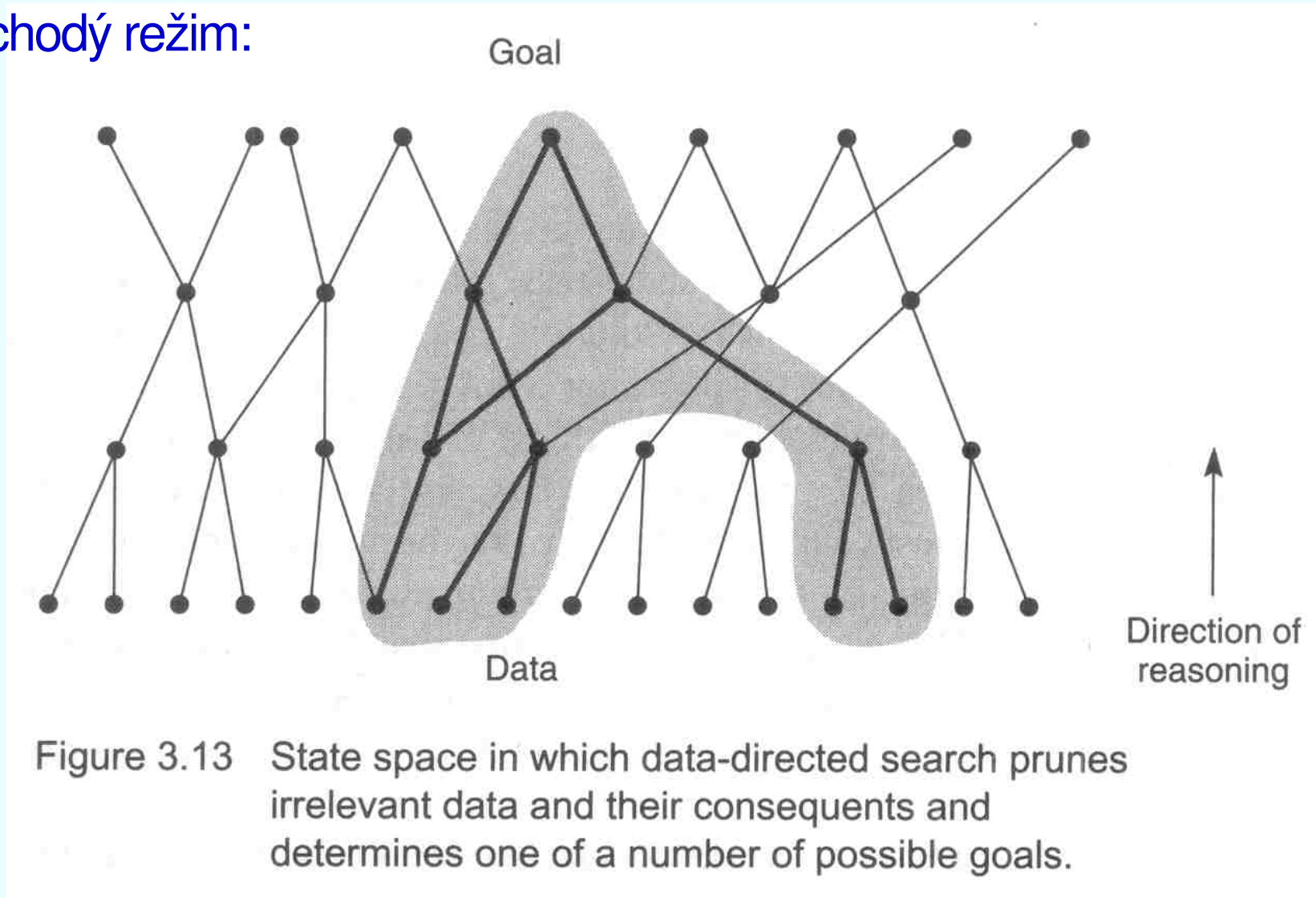


Figure 3.13 State space in which data-directed search prunes irrelevant data and their consequents and determines one of a number of possible goals.

2. Řešení úloh

Zpětnochodý režim:

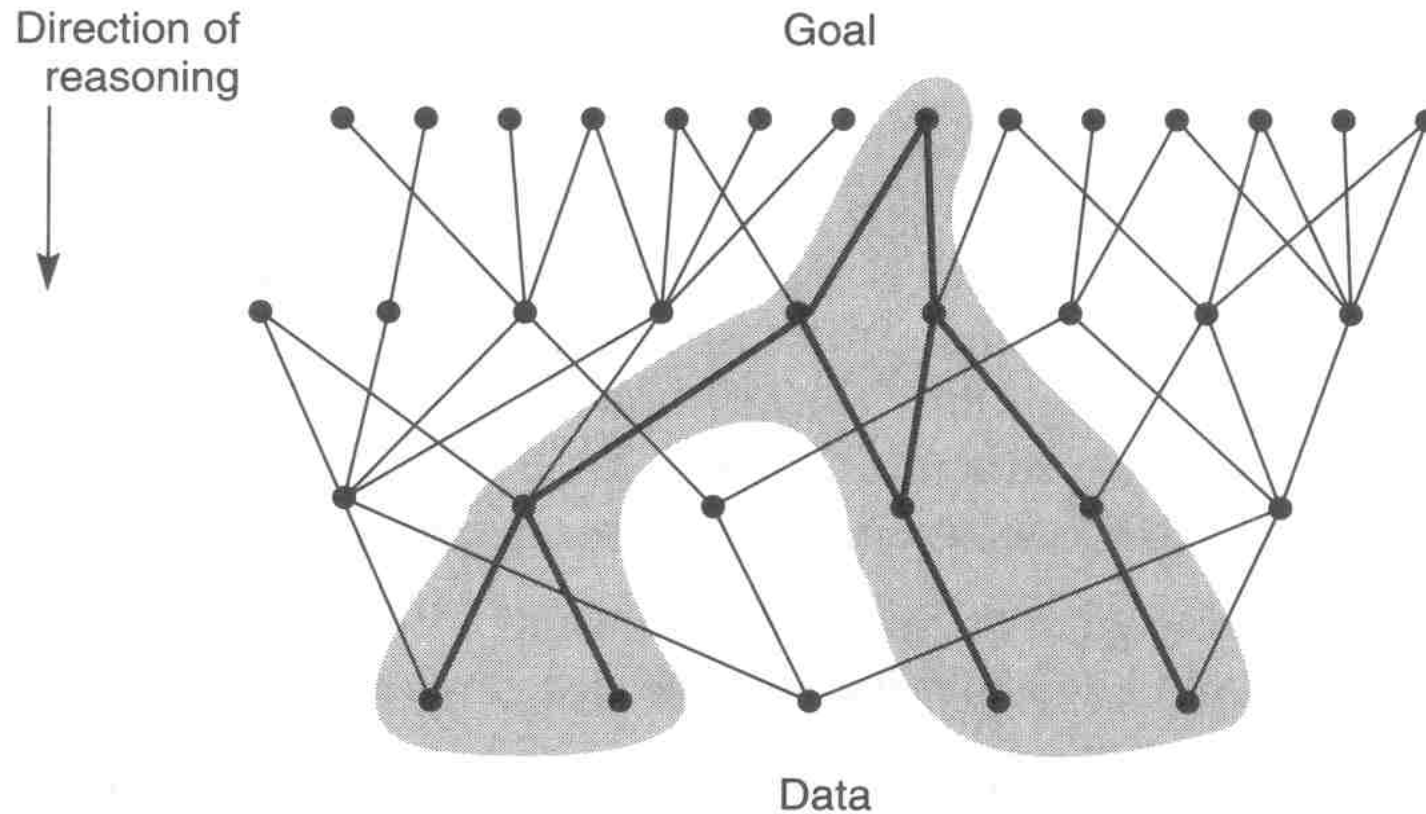


Figure 3.12 State space in which goal-directed search effectively prunes extraneous search paths.

2. Řešení úloh

ZÁKLADNÍ PROCEDURA PRODUKČNÍHO SYSTÉMU

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

repeat

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

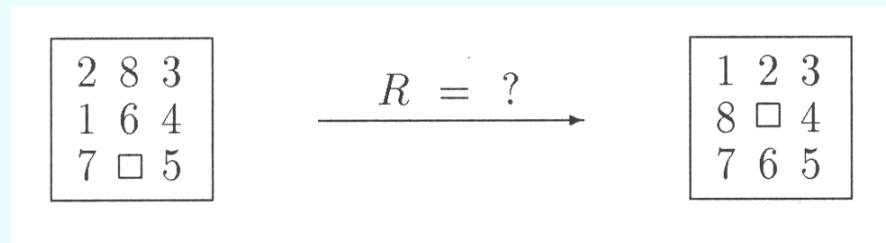
$DATA \leftarrow r_i (DATA)$; { aplikace r_i na $DATA$ }

until $DATA$ splňují cílovou podmínku;

2. Řešení úloh

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 |

2. Řešení úloh

ŘÍDICÍ MECHANISMY (řídící strategie)

- neodvolatelné (irrevocable)
- pokusné (tentative)

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

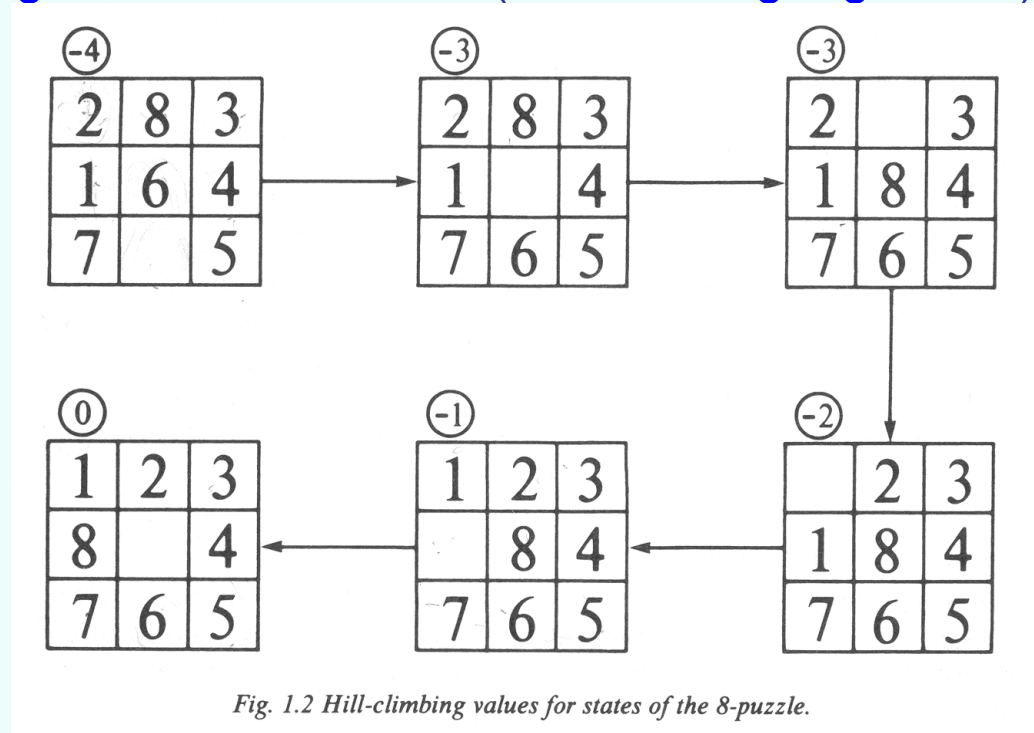


Fig. 1.2 Hill-climbing values for states of the 8-puzzle.

2. Řešení úloh

Pokusné řídicí strategie

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

2. Řešení úloh

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 vrácen *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* }

2. Řešení úloh

5. $R \leftarrow \mathbf{FIRST} (RULES);$
{ aplikováno bude v pořadí první, resp. podle daného kritéria "nejlepší" pravidlo }
6. $RULES \leftarrow \mathbf{TAIL} (RULES);$
{ z posloupnosti aplikovatelných produkčních pravidel je vyjmuto zvolené pravidlo }
7. $NEW_DATA \leftarrow \mathbf{R} (DATA);$
{ na obsah databáze $DATA$ je aplikováno produkční pravidlo \mathbf{R} z posloupnosti $RULES$ a je vygenerován nový obsah databáze NEW_DATA }
8. $PATH \leftarrow \mathbf{BACKTRACK} (NEW_DATA);$
{ procedura $\mathbf{BACKTRACK}$ je vyvolána rekursivně pro nový obsah databáze }
9. **if** $PATH = FAIL$ **then goto** LOOP;
{ vrátilo-li vyvolání procedury $\mathbf{BACKTRACK}$ chybový příznak $FAIL$, přechází na použití dalšího produkčního pravidla (pokud takové existuje) }
10. **return** $\mathbf{CONS} (R, PATH);$
{ 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

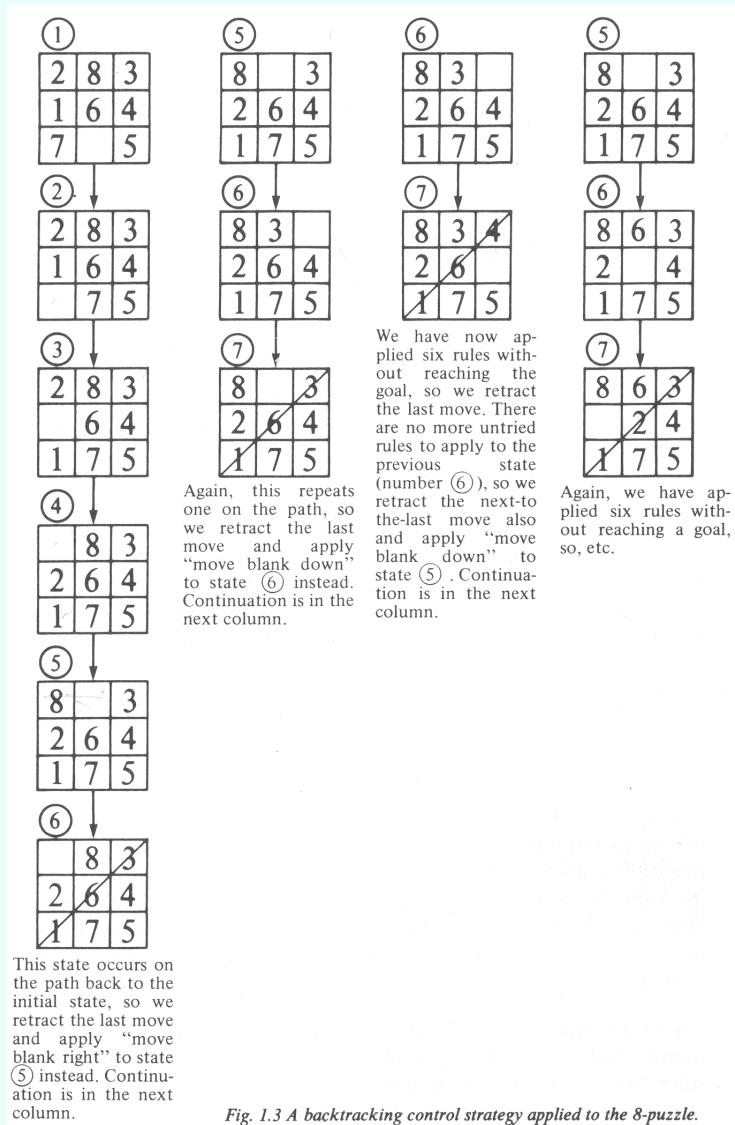


Fig. 1.3 A backtracking control strategy applied to the 8-puzzle.

2. Řešení úloh

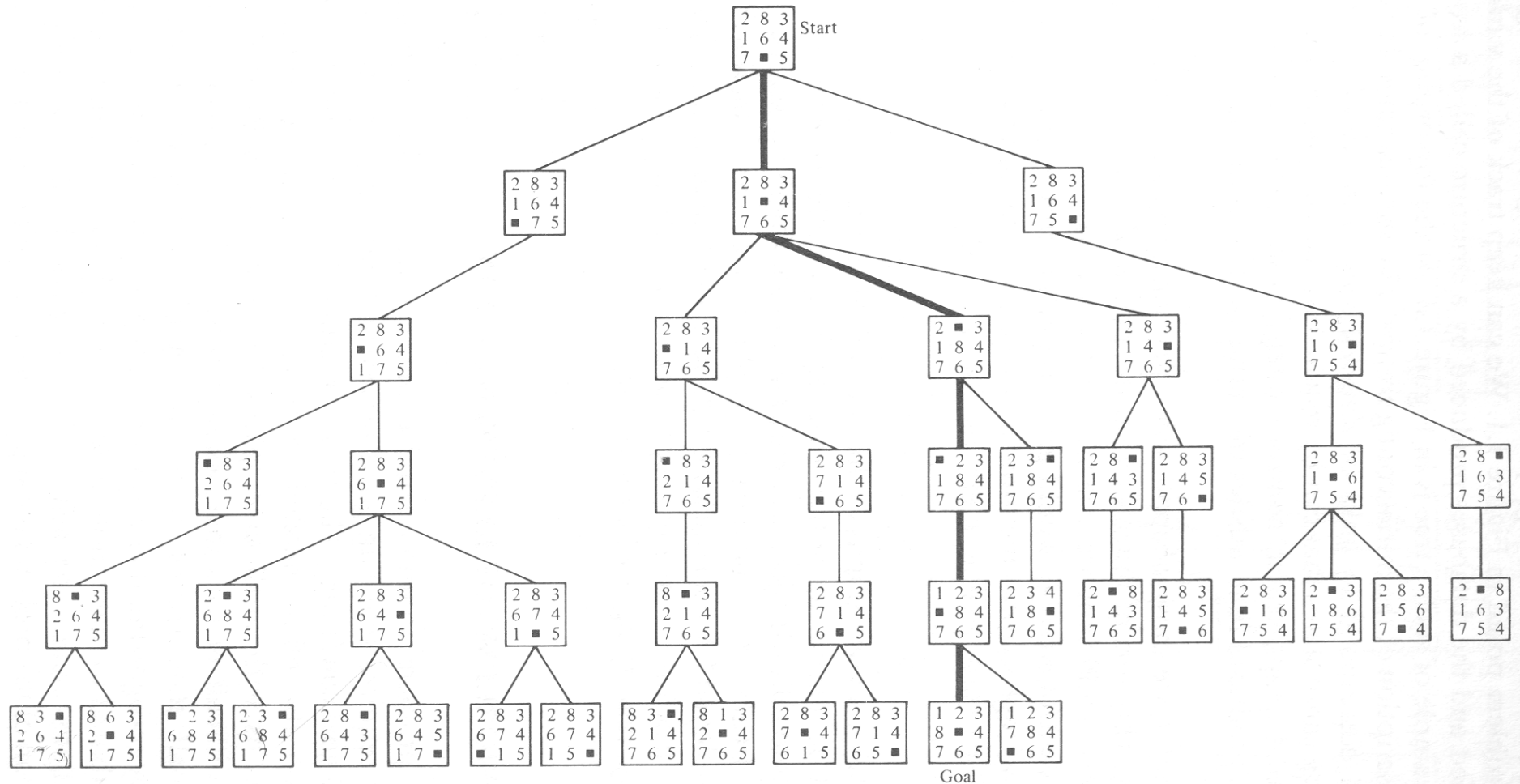


Fig. 1.4 A search tree for the 8-puzzle.

2. Řešení úloh

METODY HLEDÁNÍ V GRAFU (GRAPH SEARCH)

Zapamatovávají si celou dosud vygenerovanou část stromu řešení –

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

2. Řešení úloh

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*).

2. Řešení úloh

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

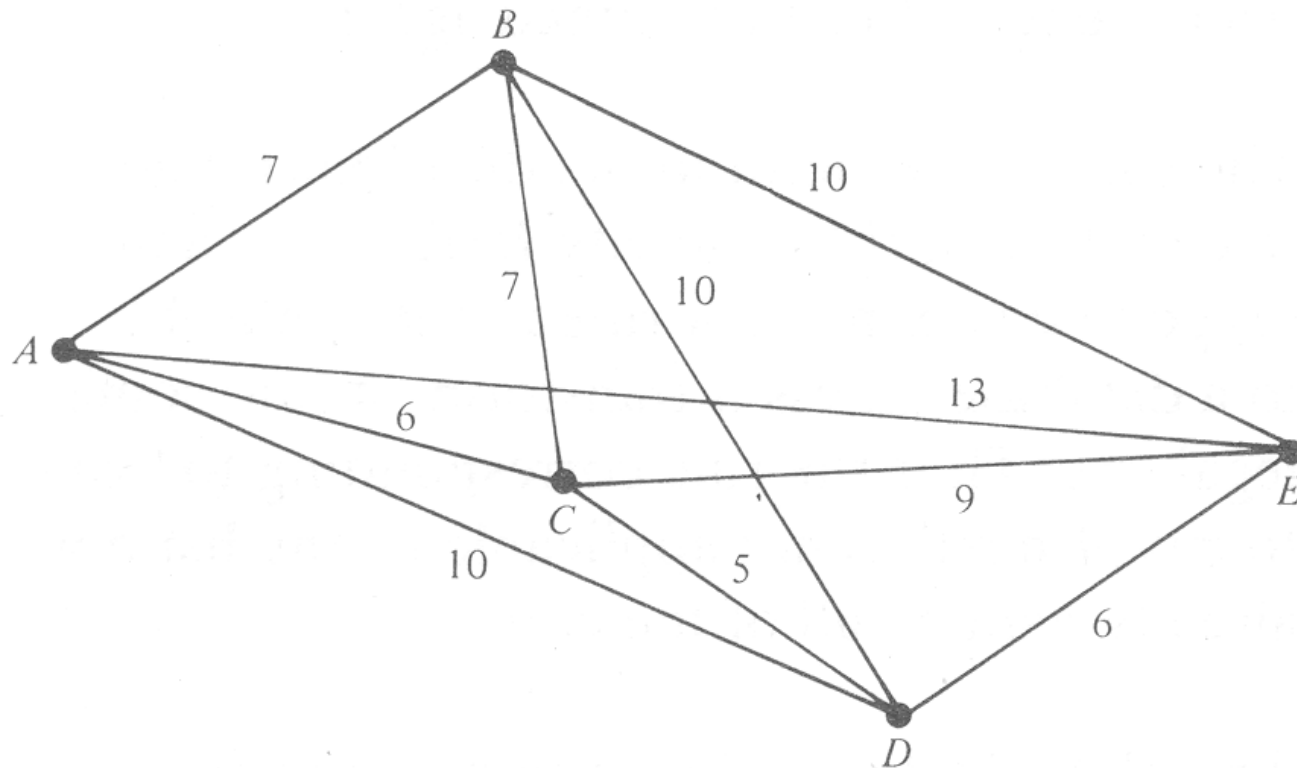
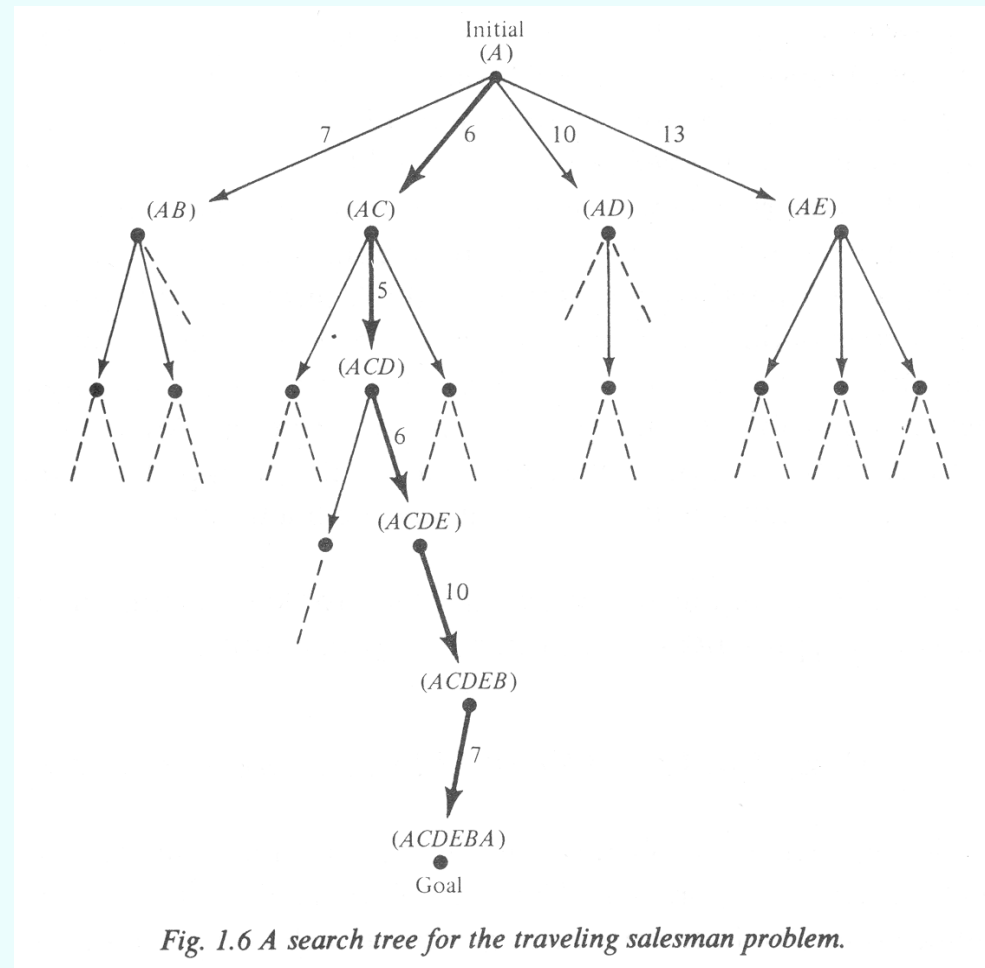


Fig. 1.5 A map for the traveling salesman problem.

2. Řešení úloh

Graf řešení úlohy obchodního cestujícího:



2. Řešení úloh

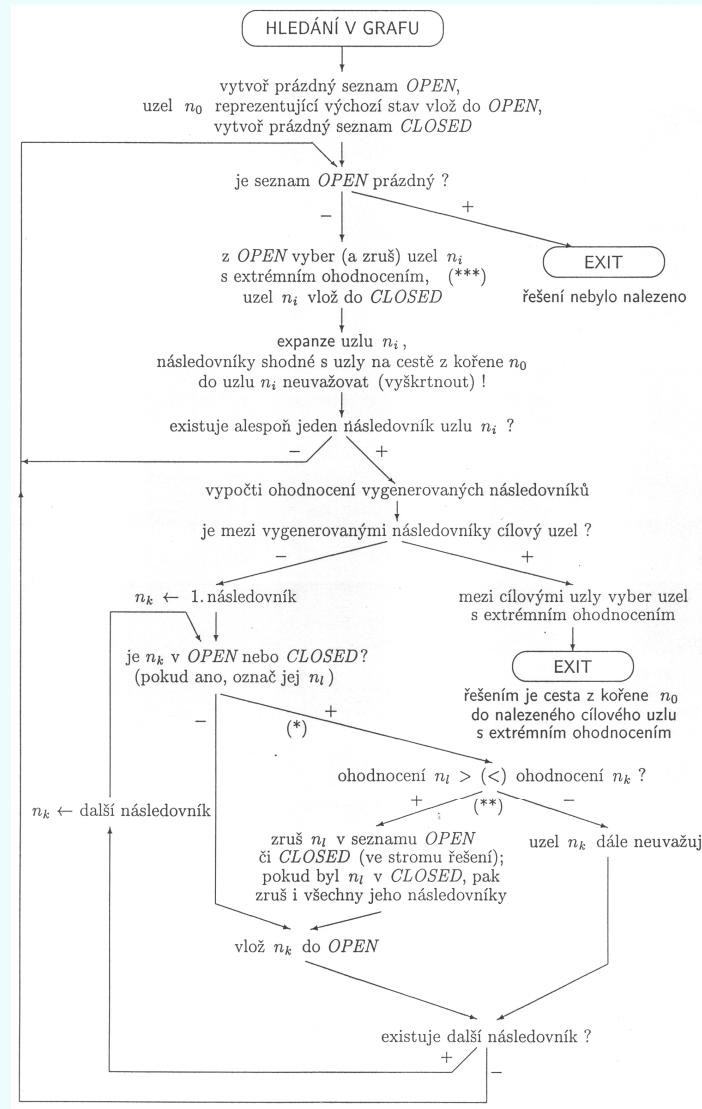
ALGORITMUS HLEDÁNÍ V GRAFU (GRAPH SEARCH)

1. Vytvoř seznam *OPEN*, vlož do něj uzel n_0 a urči jeho ohodnocení f_0 . Vytvoř prázdný seznam *CLOSED*.
2. Je-li seznam *OPEN* prázdný, řešení neexistuje, tj. ukonči řešení.
3. Ze seznamu *OPEN* vyber uzel n_i , který má *extrémní* (minimální nebo maximální) ohodnocení. Uzel n_i zruš v seznamu *OPEN* a umísti jej do seznamu *CLOSED*.
4. Proveď expanzi uzlu n_i , přičemž ihned vyškrtni všechny bezprostřední následovníky, kteří už jsou předchůdci uzlu n_i . Zbylé bezprostřední následovníky n_j zařaď do seznamu *OPEN* a urči (vypočti) jejich ohodnocení f_j . Neexistuje-li žádný bezprostřední následovník, přejdi na krok 2.
5. Jsou-li mezi vygenerovanými následovníky uzlu n_i uzly *cílové*, vyber cílový uzel s extrémním ohodnocením a ukonči prohledávání. Ve vygenerovaném stromu řešení najdi cestu od kořene stromu k cílovému uzlu a jako nalezené řešení poskytni posloupnost elementárních operátorů (produkčních pravidel) prováděných podél nalezené cesty. Ukonči řešení.
V opačném případě pokračuj krokem 6.

2. Řešení úloh

6. Pro každého bezprostředního následovníka uzlu n_i , kterého označíme n_j , a jeho ohodnocení f_j , proved':
- (a) Není-li uzel n_j s vypočteným ohodnocením f_j ani v seznamu *OPEN* ani v seznamu *CLOSED* (uzel se ve vygenerovaném stromu řešení dosud nevyskytuje), umístí jej do seznamu *OPEN*.
 - (b) Jestliže se uzel n_j s ohodnocením f_j v seznamu *OPEN* nebo *CLOSED* již vyskytuje, avšak s ohodnocením $f'_j \neq f_j$ pak
 - je-li $f'_j > (<) f_j$, vypust' uzel n_j s ohodnocením f'_j ze seznamu *OPEN* nebo *CLOSED* (ze stromu řešení) a umístí nově vygenerovaný uzel n_j s ohodnocením f_j do seznamu *OPEN*;
 - není-li $f'_j > (<) f_j$, zruš v seznamu *OPEN* nově vygenerovaný uzel n_j s ohodnocením f_j .
 - (c) Dojde-li ke zrušení nějakého uzlu n_j ze seznamu *CLOSED*, musejí být zrušeni rovněž všichni jeho následovníci; přitom je lhostejno, zda se nalézají v seznamu *OPEN* nebo *CLOSED* (musejí být zrušeni ve stromu řešení, tj. zlikviduje se celý podstrom s kořenem n_j).
7. Pokračuj krokem 2.

2. Řešení úloh



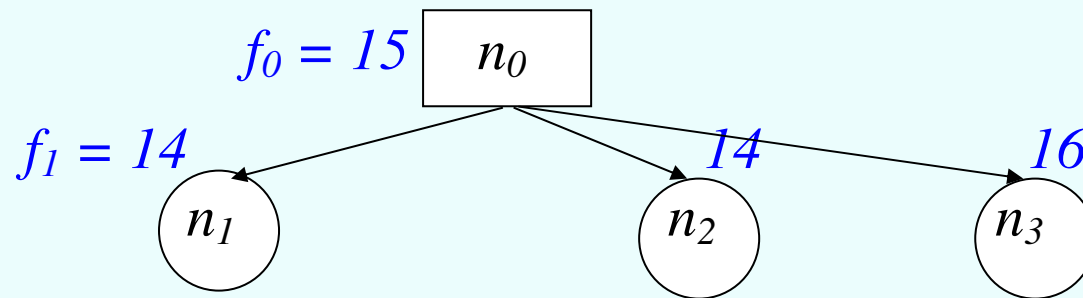
2. Řešení úloh

Princip vytváření stromu řešení úlohy:

$$f_0 = 15 \quad \textcircled{n_0}$$

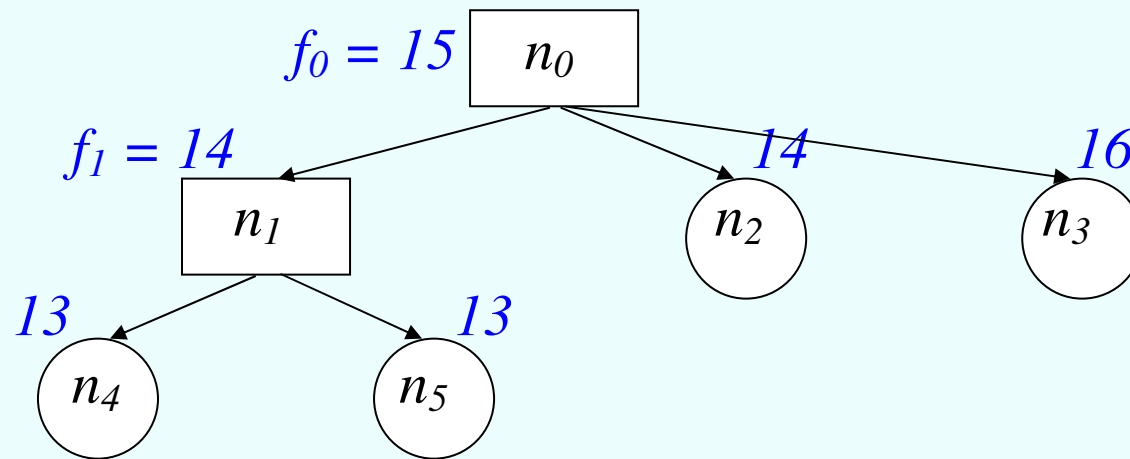
2. Řešení úloh

Princip vytváření stromu řešení úlohy:



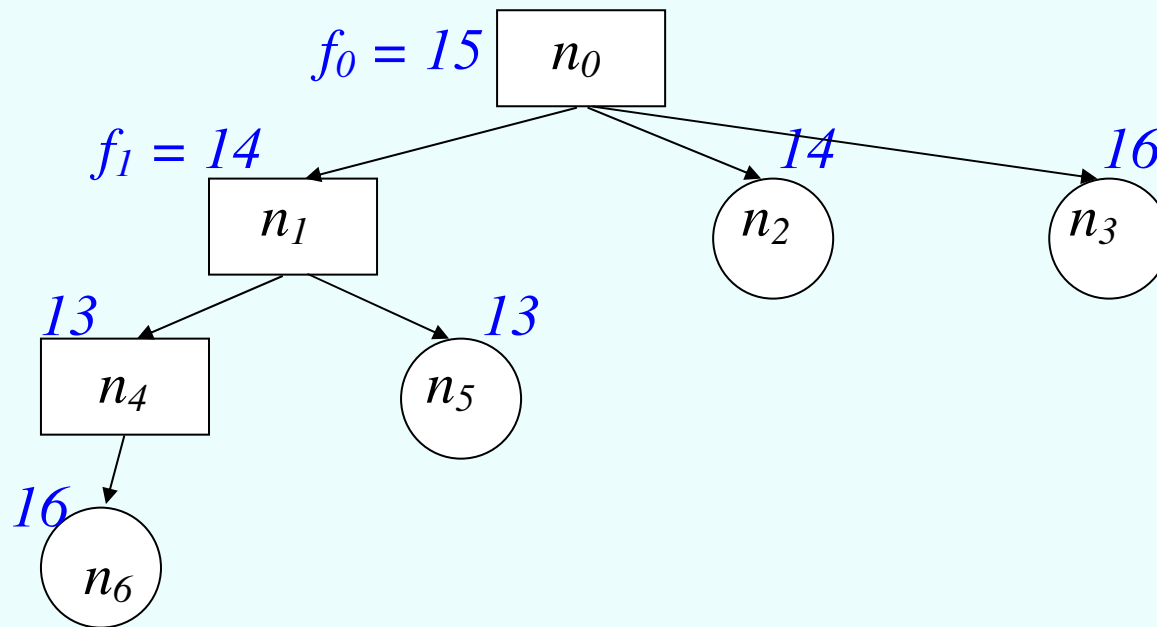
2. Řešení úloh

Princip vytváření stromu řešení úlohy:



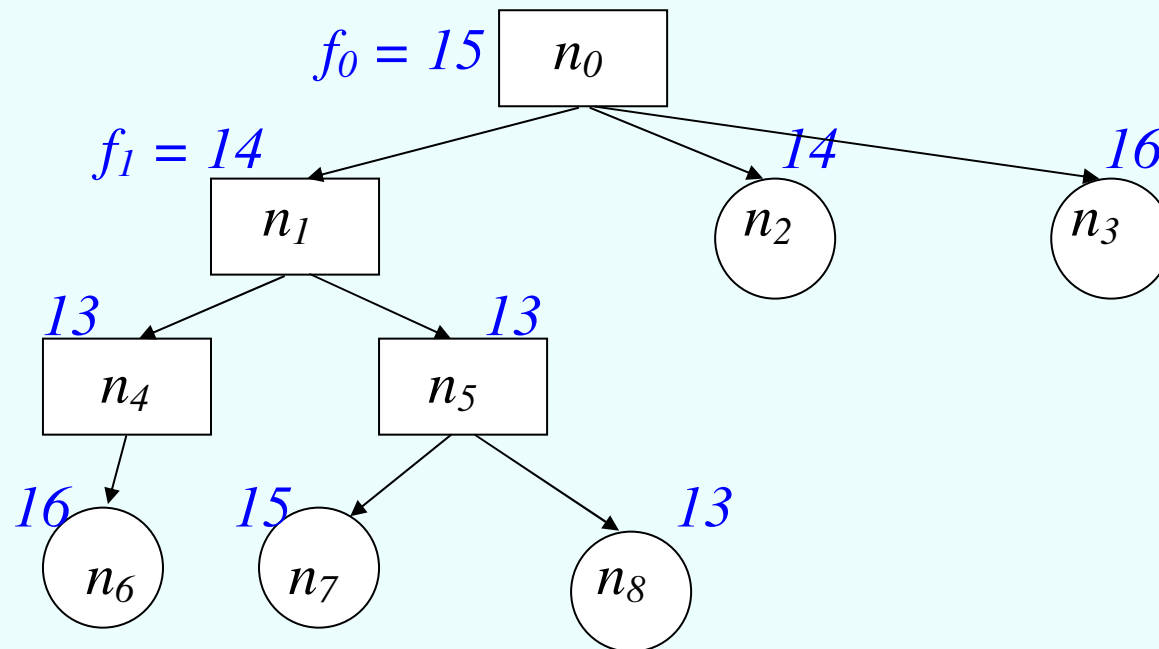
2. Řešení úloh

Princip vytváření stromu řešení úlohy:



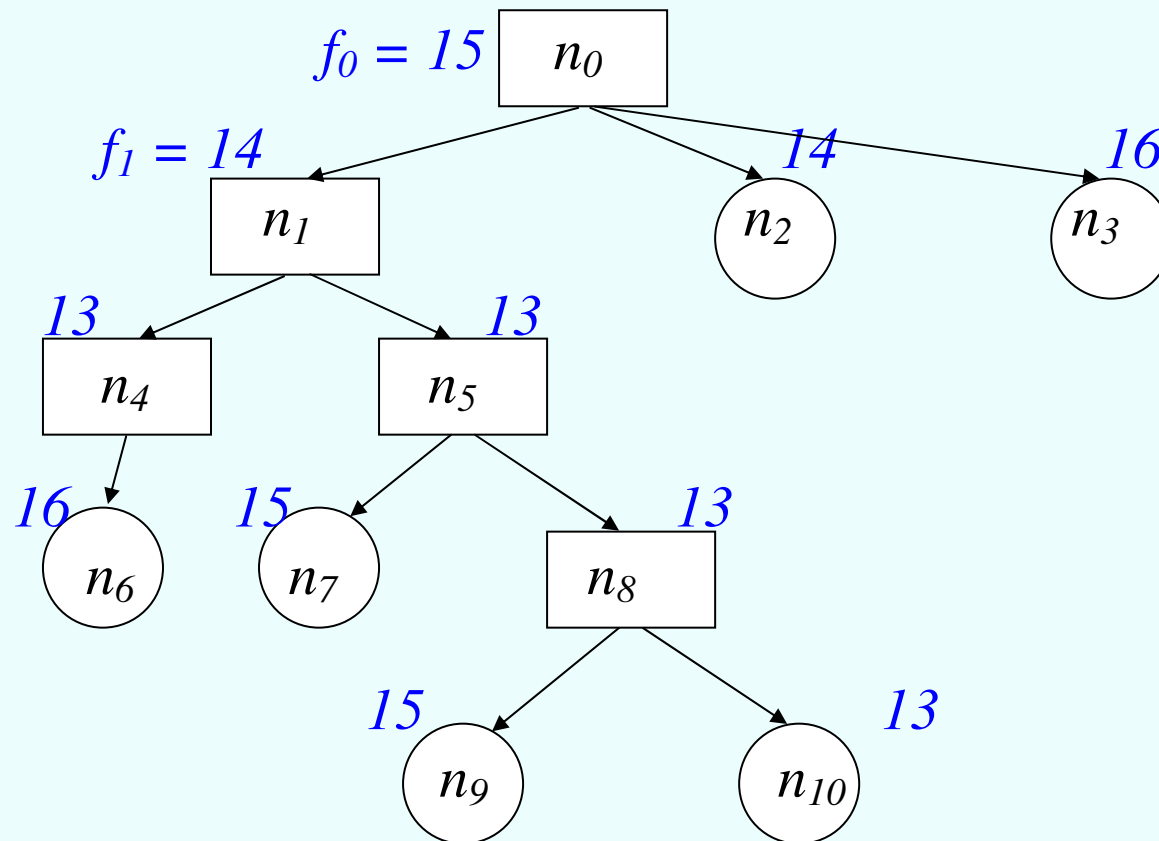
2. Řešení úloh

Princip vytváření stromu řešení úlohy:



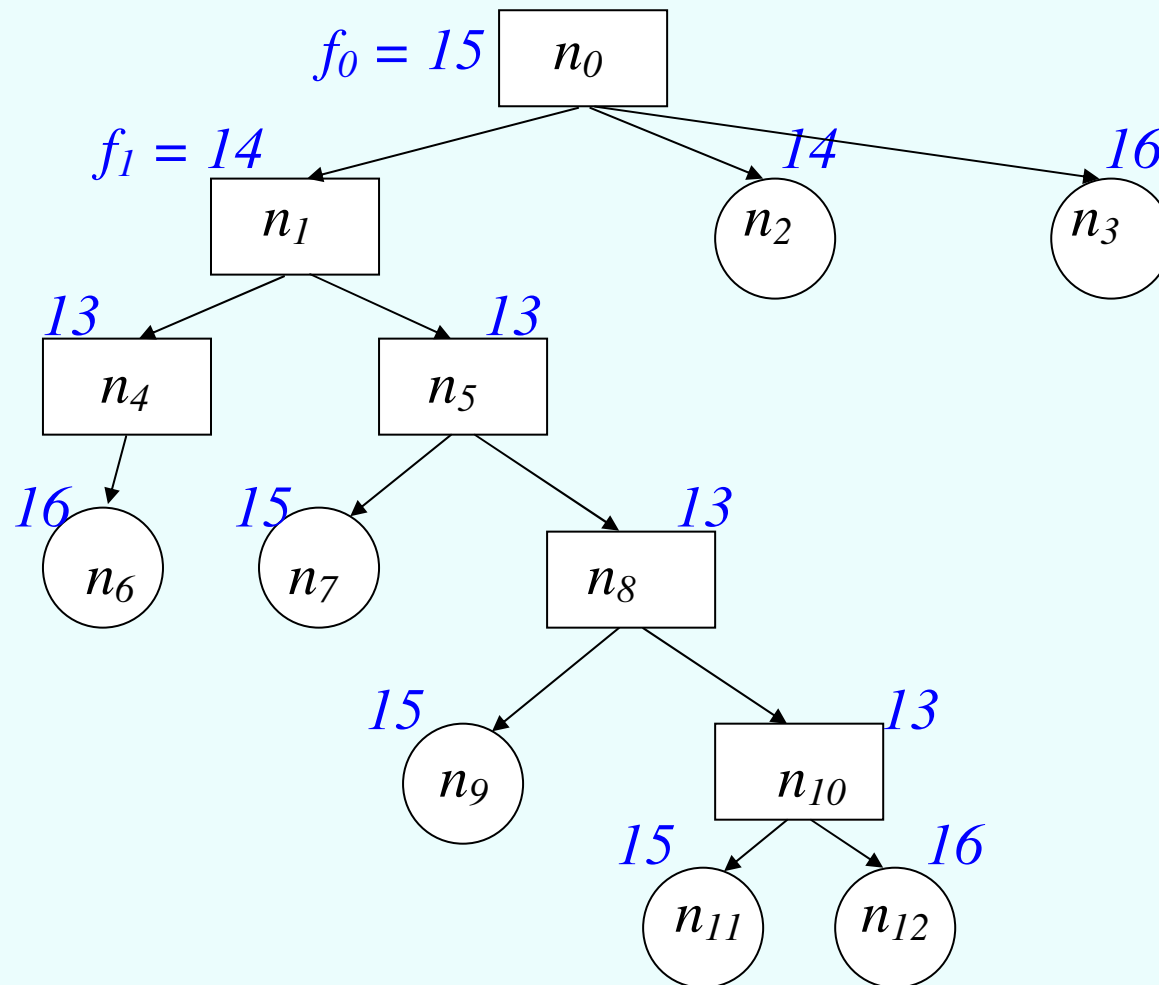
2. Řešení úloh

Princip vytváření stromu řešení úlohy:



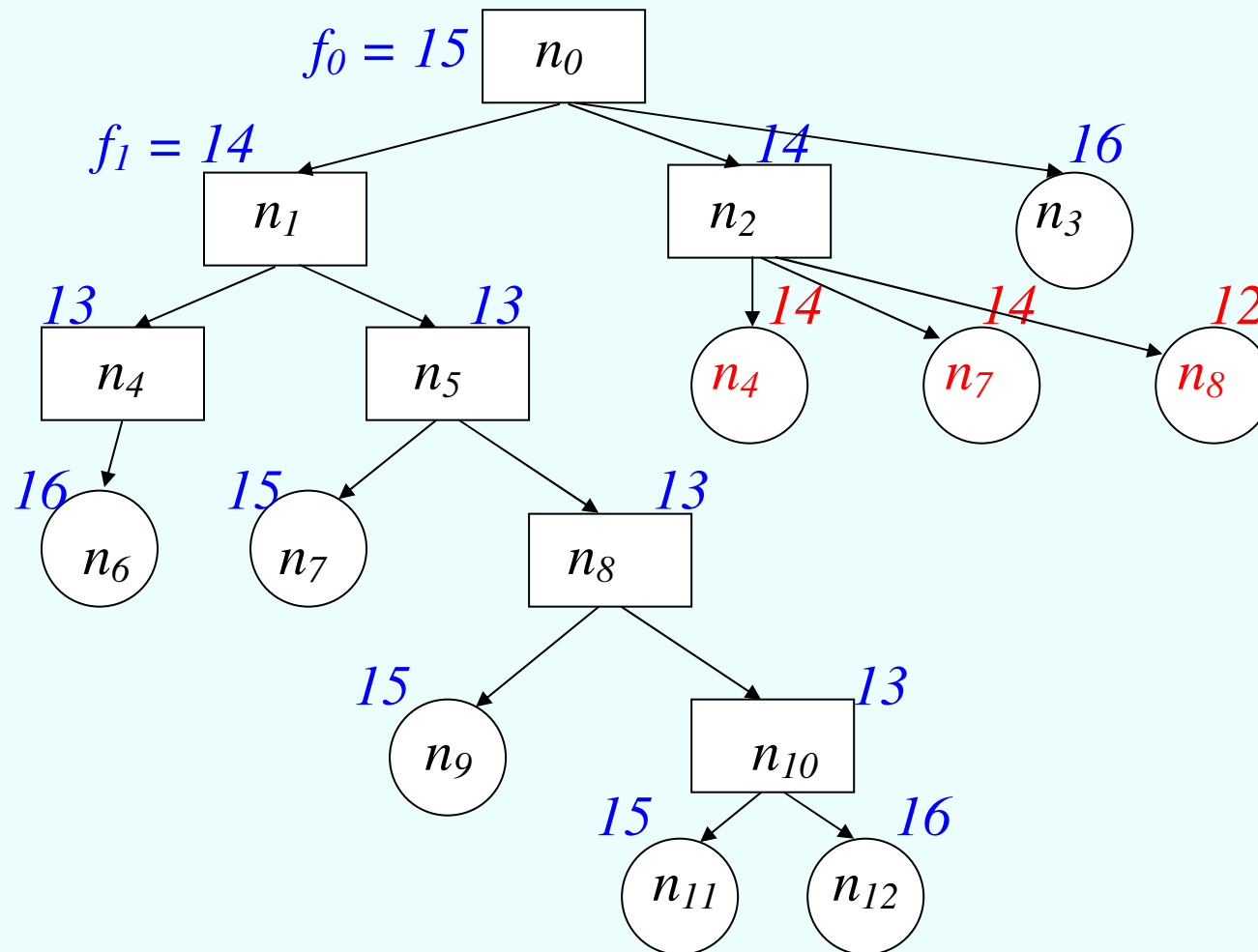
2. Řešení úloh

Princip vytváření stromu řešení úlohy:



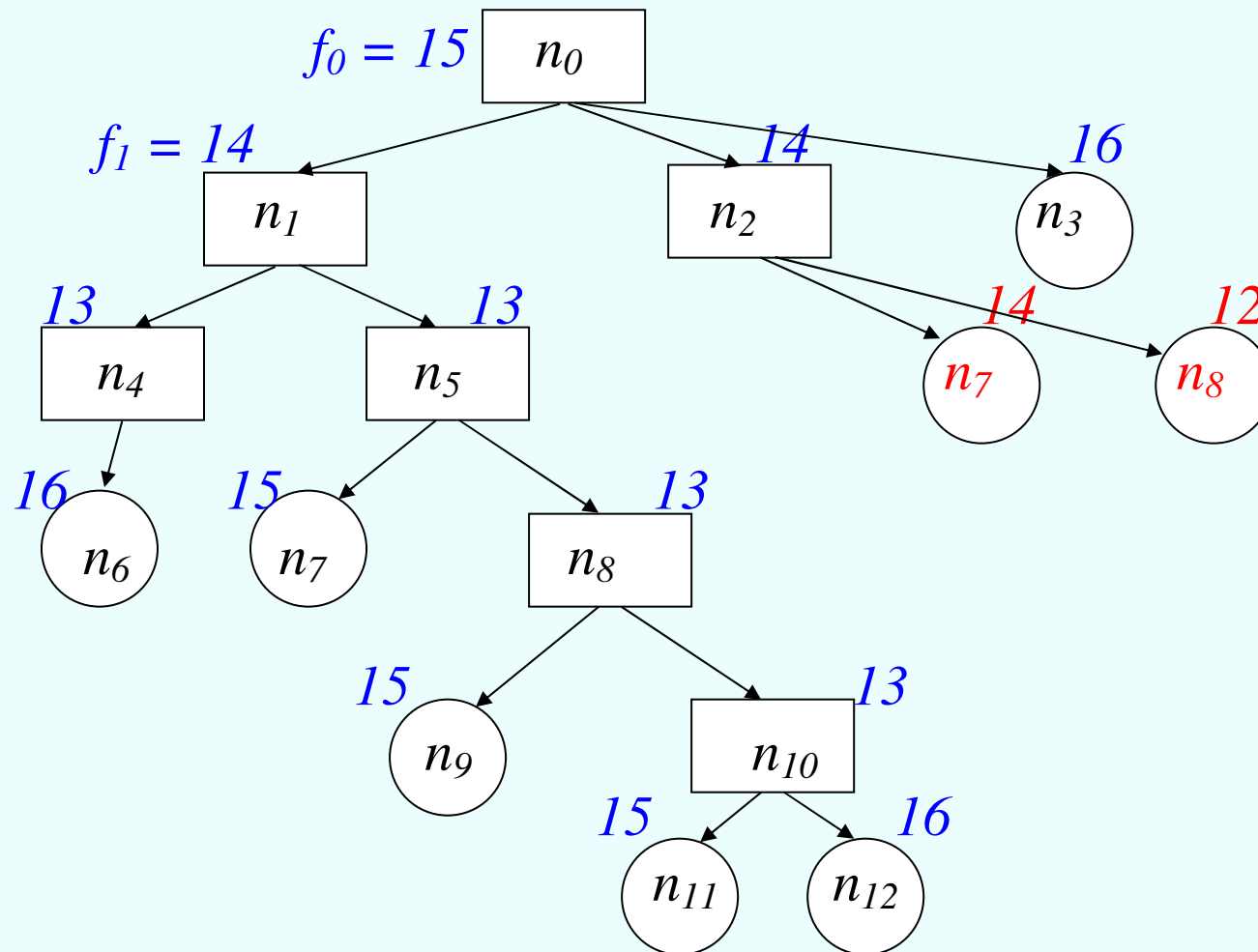
2. Řešení úloh

Princip vytváření stromu řešení úlohy:



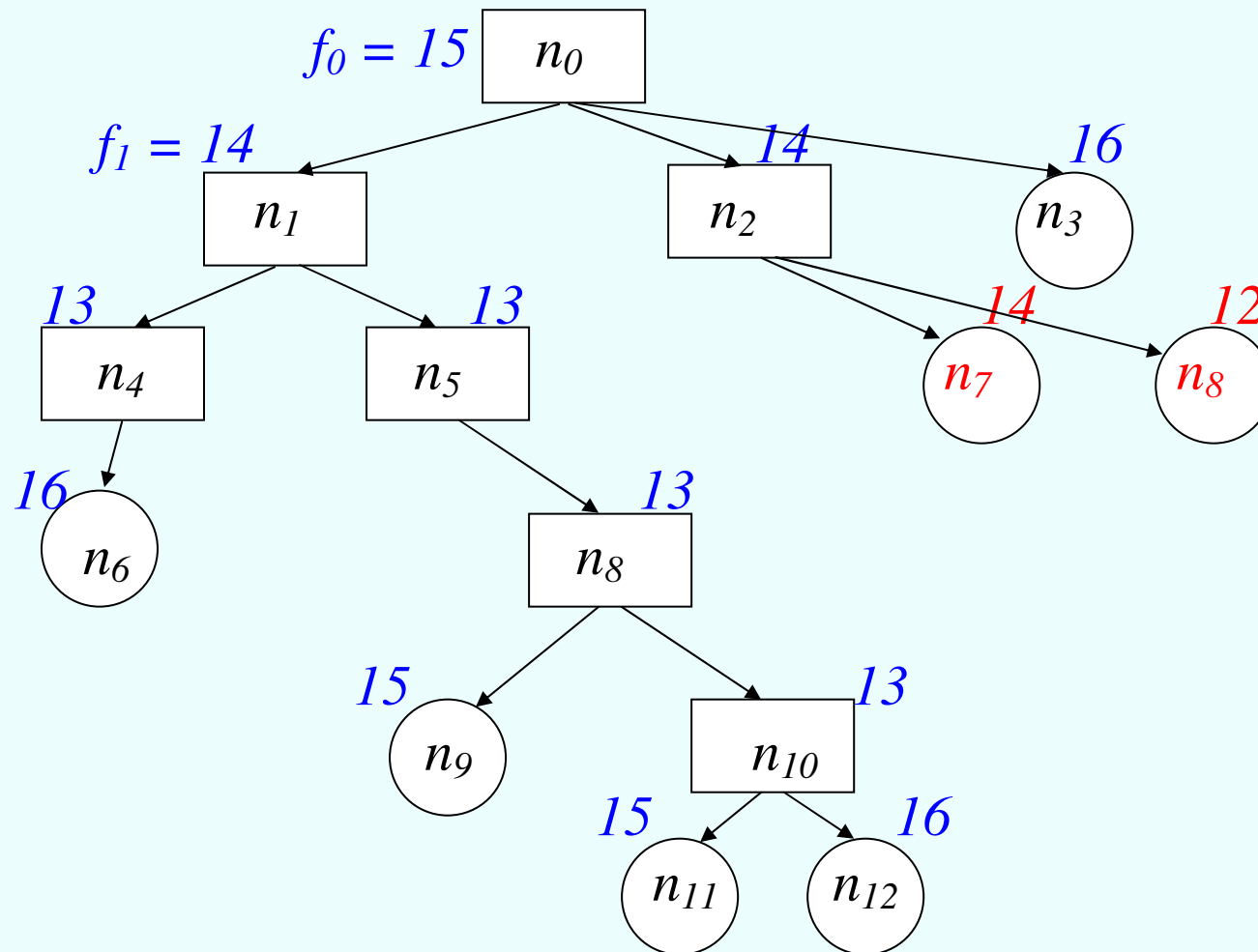
2. Řešení úloh

Princip vytváření stromu řešení úlohy:



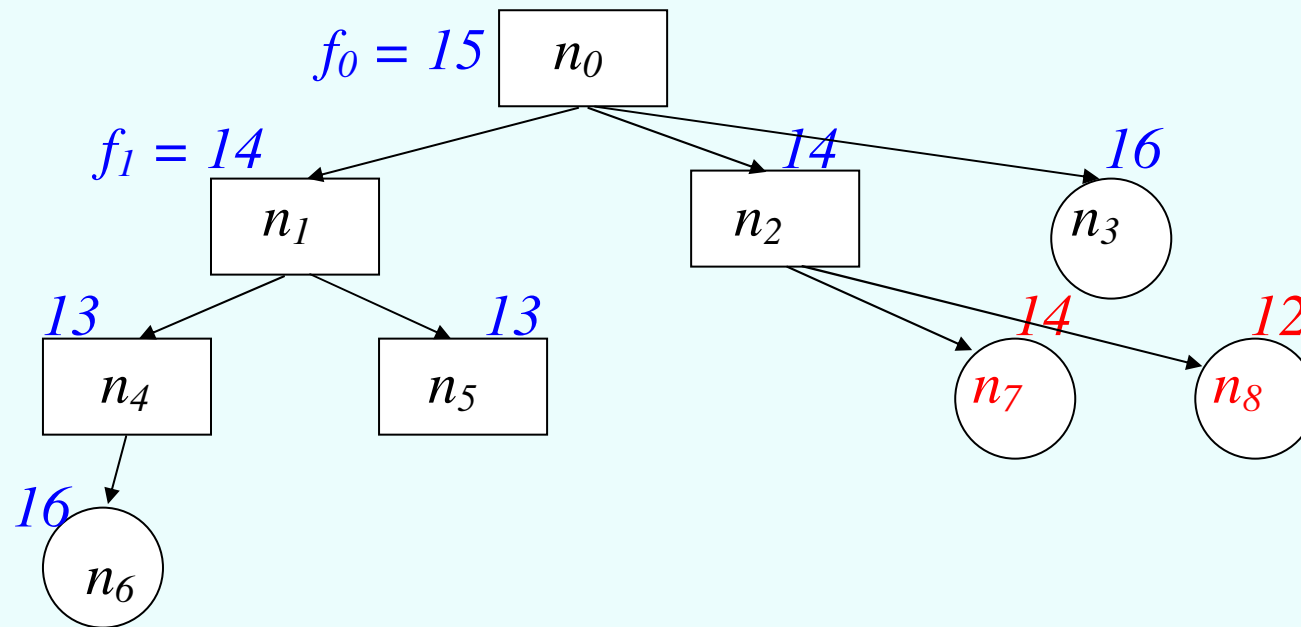
2. Řešení úloh

Princip vytváření stromu řešení úlohy:



2. Řešení úloh

Princip vytváření stromu řešení úlohy:



2. Řešení úloh

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

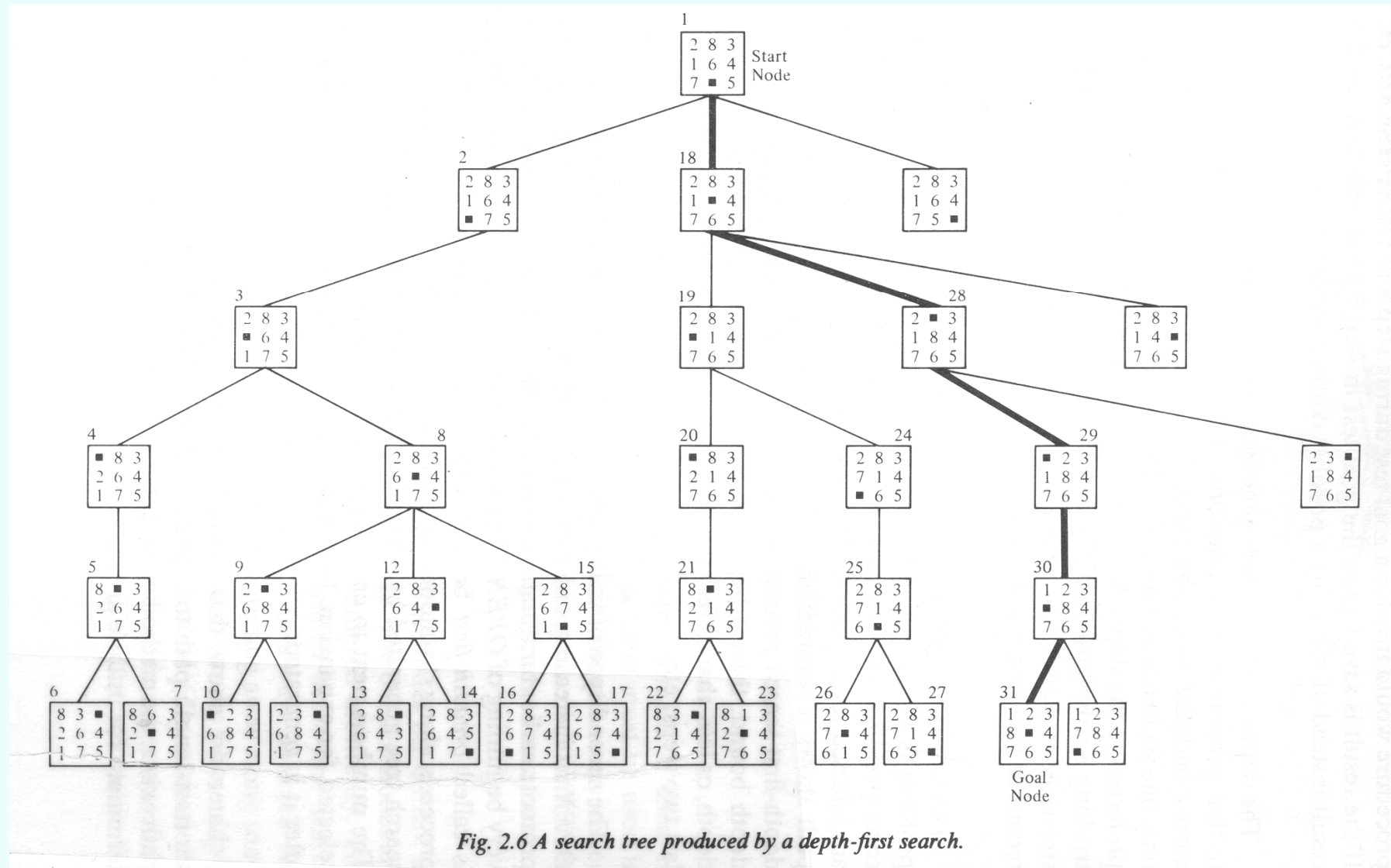


Fig. 2.6 A search tree produced by a depth-first search.

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

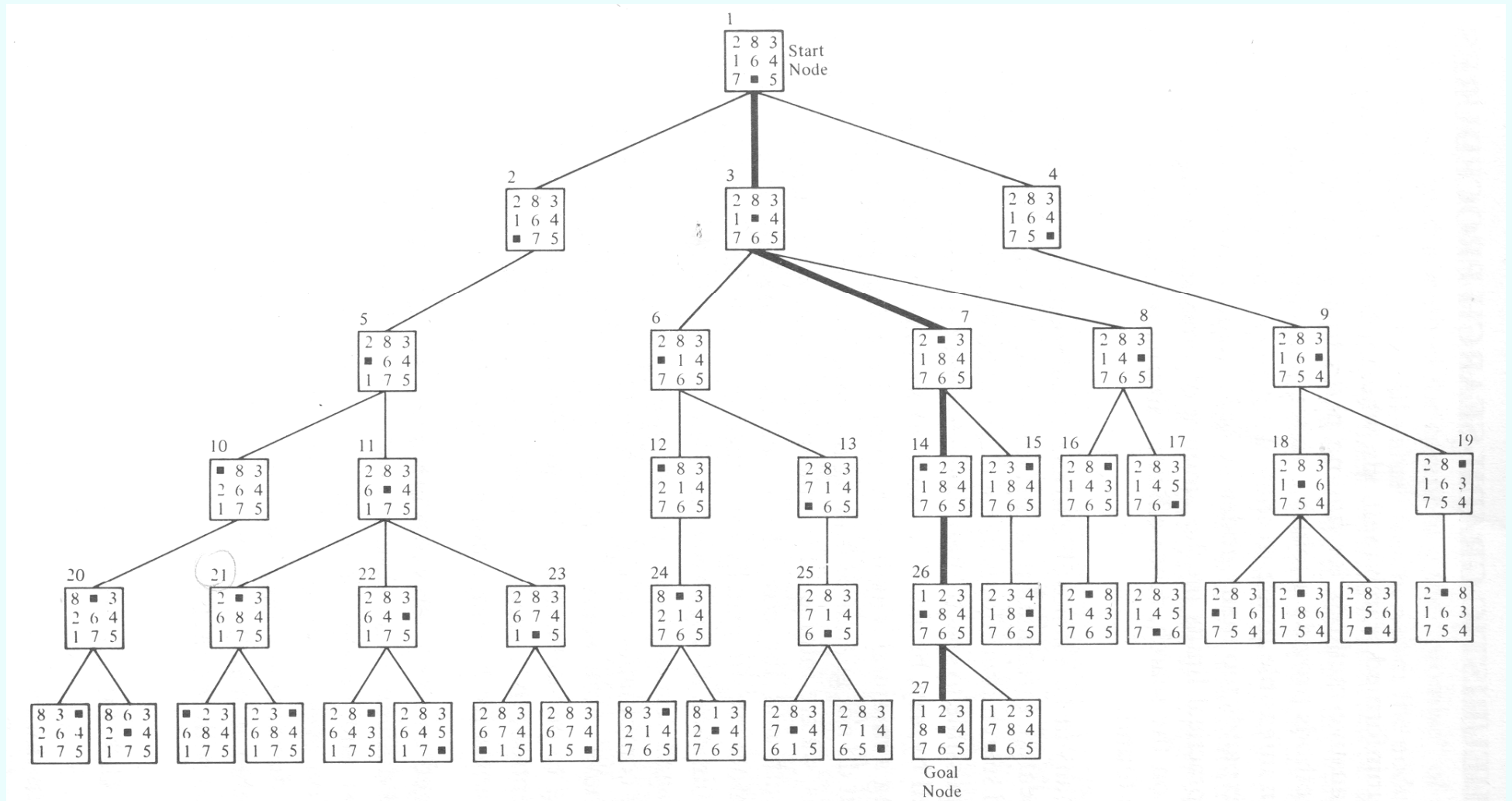


Fig. 2.7 A search tree produced by a breadth-first search.

2. Řešení úloh

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)

2. Řešení úloh

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.

2. Řešení úloh

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:

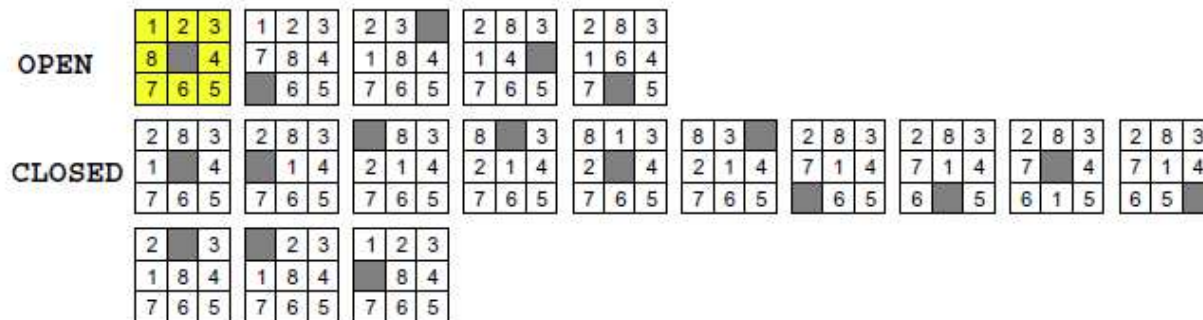
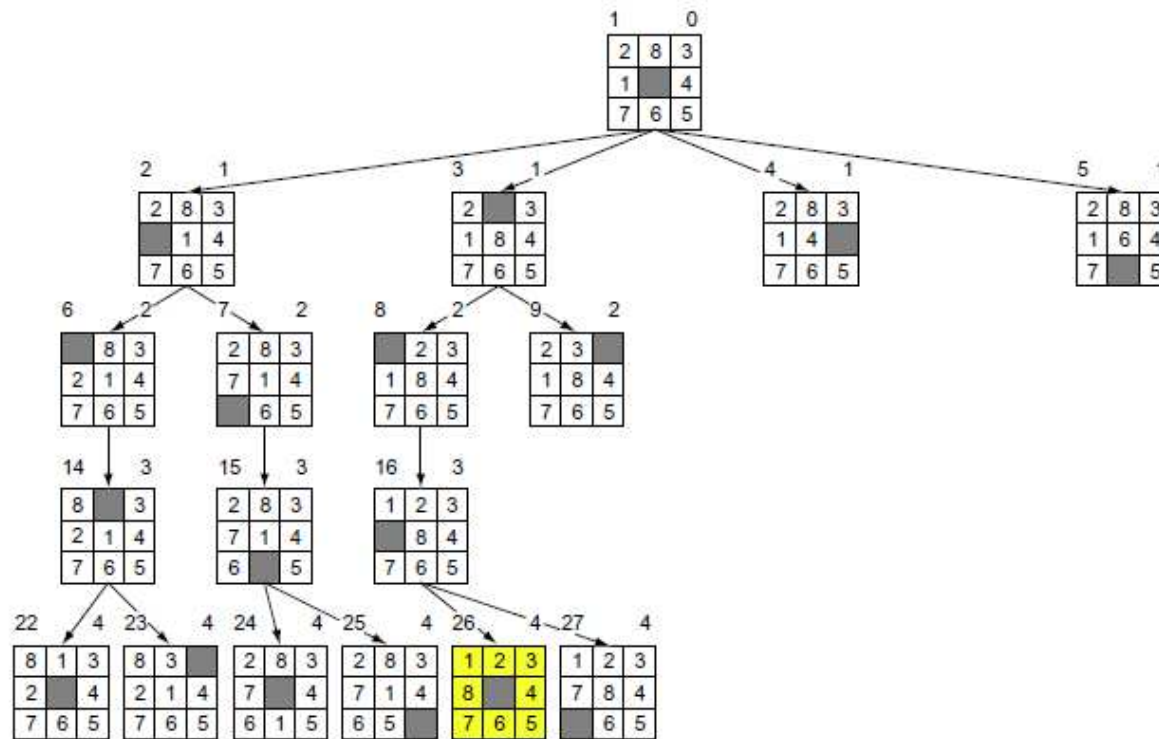
- Jedná se o kompromis mezi výhodami a nevýhodami vyhledávání do šířky a do hloubky.
- Je nejvhodnější neinformovanou metodou pro velké stavové prostory s neznámou hloubkou, ve které se nachází řešení.

Nevýhody algoritmu iterativního vyhledávání:

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

2. Řešení úloh

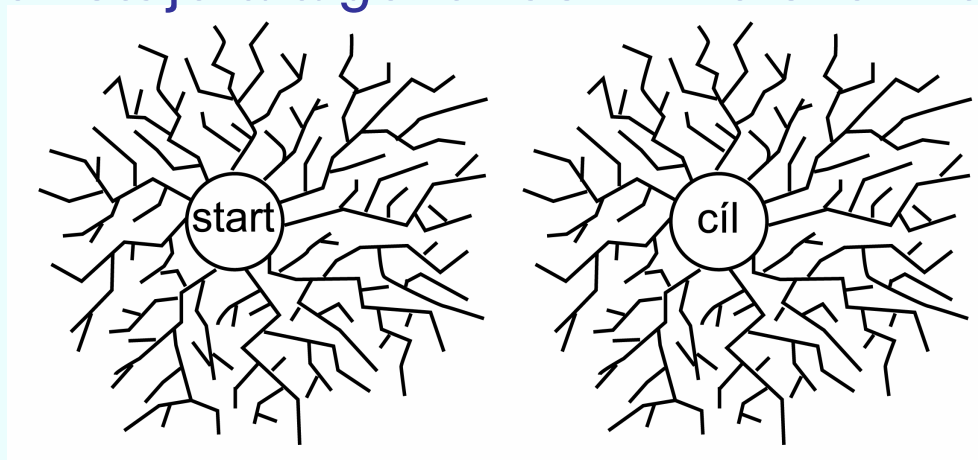
Příklad:



2. Řešení úloh

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.



2. Řešení úloh

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:



2. Řešení úloh

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
```

2. Řešení úloh

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

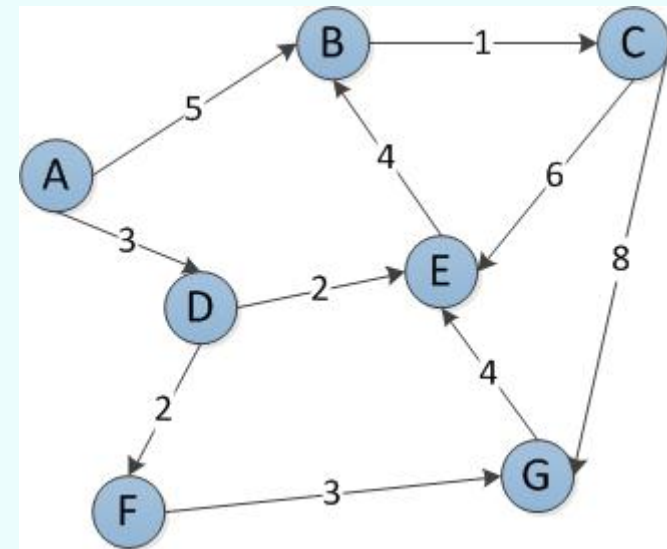
Example

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	\emptyset
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	\emptyset		



^{*} 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.

2. Řešení úloh

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:

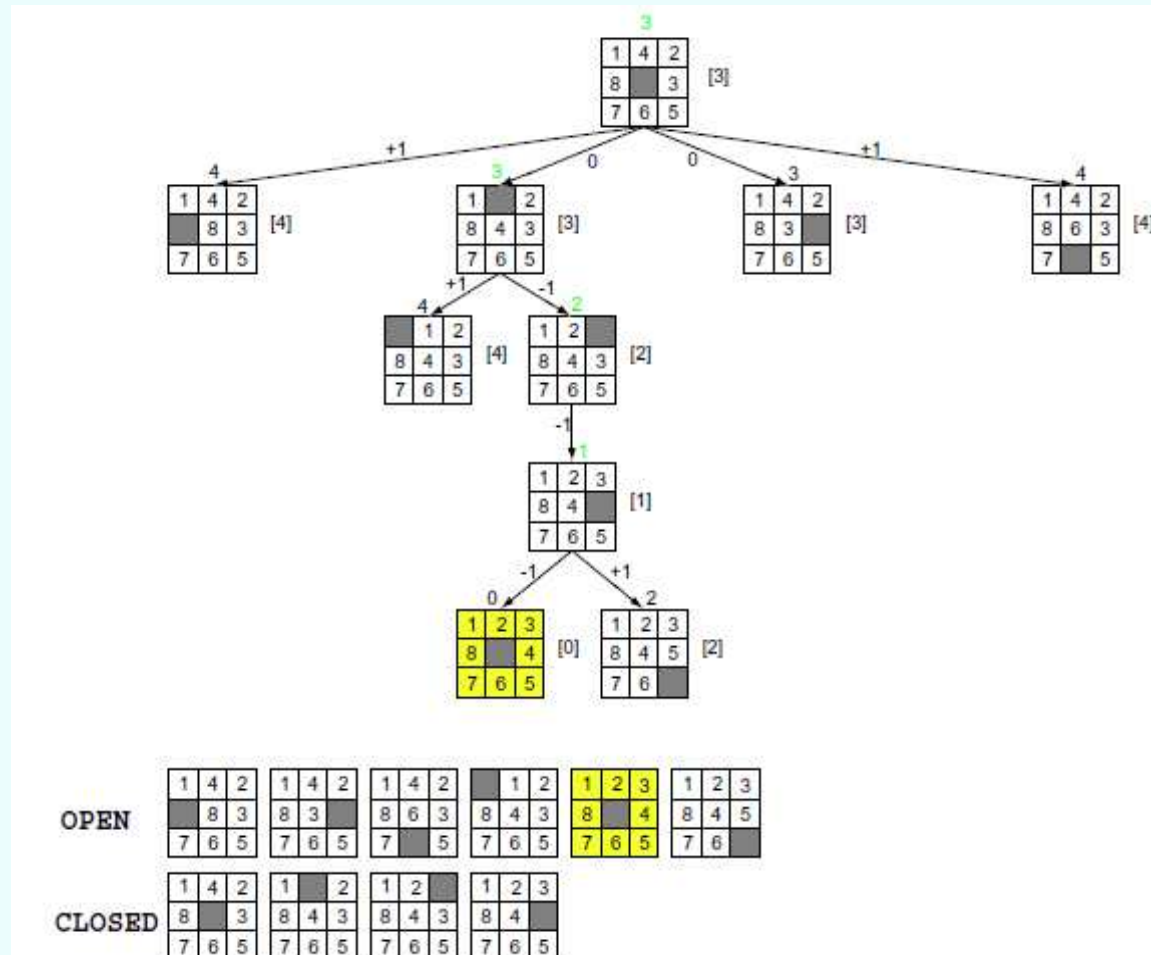
- Vždy nalezne optimální řešení (za předpokladu, že řešení existuje).
- Vhodně se používá u úlohy obchodního cestujícího.

Nevýhody uniformního prohledávání:

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

2. Řešení úloh

Příklad:



2. Řešení úloh

INFORMOVANÉ (CÍLENÉ) STRATEGIE HLEDÁNÍ ŘEŠENÍ (HEURISTICKÉ PROHLEDÁVÁNÍ)

Základní myšlenka: Nalézt takovou heuristiku, která pomůže najít řešení při relativně nízkém počtu vygenerovaných uzlů.

Ilustrace obrázkem:

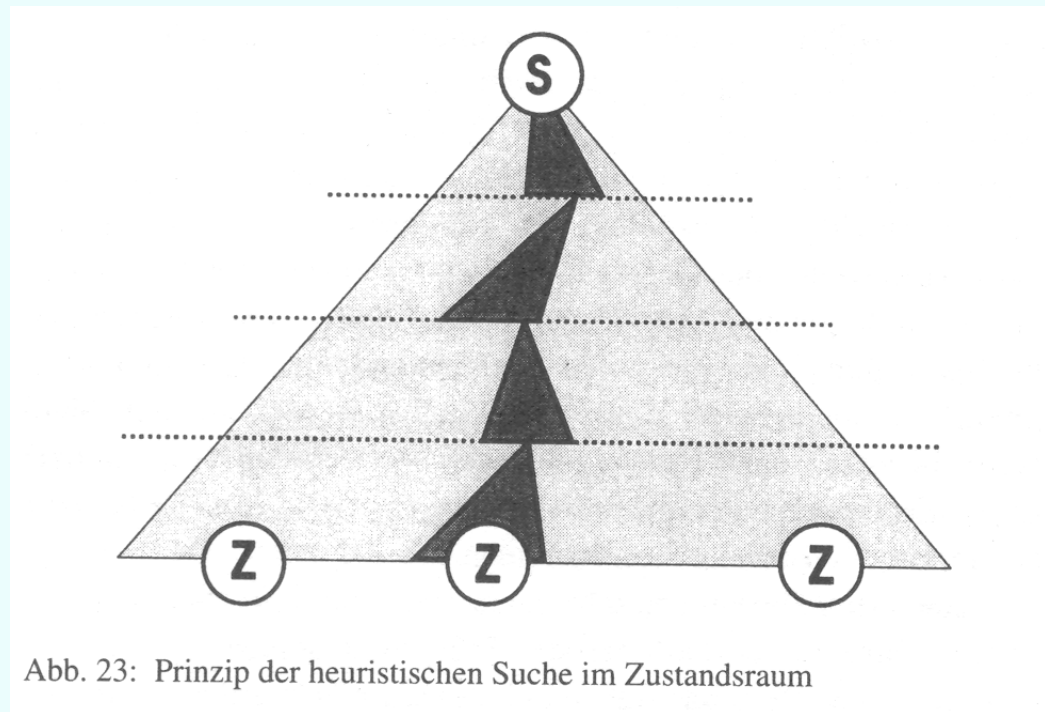


Abb. 23: Prinzip der heuristischen Suche im Zustandsraum

2. Řešení úloh

Princip: Hledáme algoritmus ohodnocující funkce pro nalezení optimální (např. nejkratší, „nejlevnější“...) cesty mezi daným a cílovým uzlem grafu řešení úlohy a její **extrém**.

Zpravidla narazíme na dva protichůdné požadavky:

- chceme nalézt optimální cestu (např. s minimálními náklady),
- požadujeme minimální náklady na prohledávání stromu řešení.

2. Řešení úloh

Princip: Hledáme algoritmus ohodnocující funkce pro nalezení optimální (např. nejkratší, „nejlevnější“...) cesty mezi daným a cílovým uzlem grafu řešení úlohy a její **extrém**.

Zpravidla narazíme na dva protichůdné požadavky:

- chceme nalézt optimální cestu (např. s minimálními náklady),
- požadujeme minimální náklady na prohledávání stromu řešení.

Př.: Heuristické prohledávání pro „8“:

Zvolíme ohodnocující funkci ve tvaru

$$f(n_i) = d(n_i) + w(n_i),$$

kde $d(n_i)$ buď délka cesty ve stromu řešení od výchozího uzlu k uzlu n_i ,

$w(n_i)$ počet kamenů v „databázi“ uzlu n_i , které neleží na svých místech.

Pozn.: Položíme – li $f(n_i) = d(n_i)$, dostáváme algoritmus prohledávání do šířky.

2. Řešení úloh

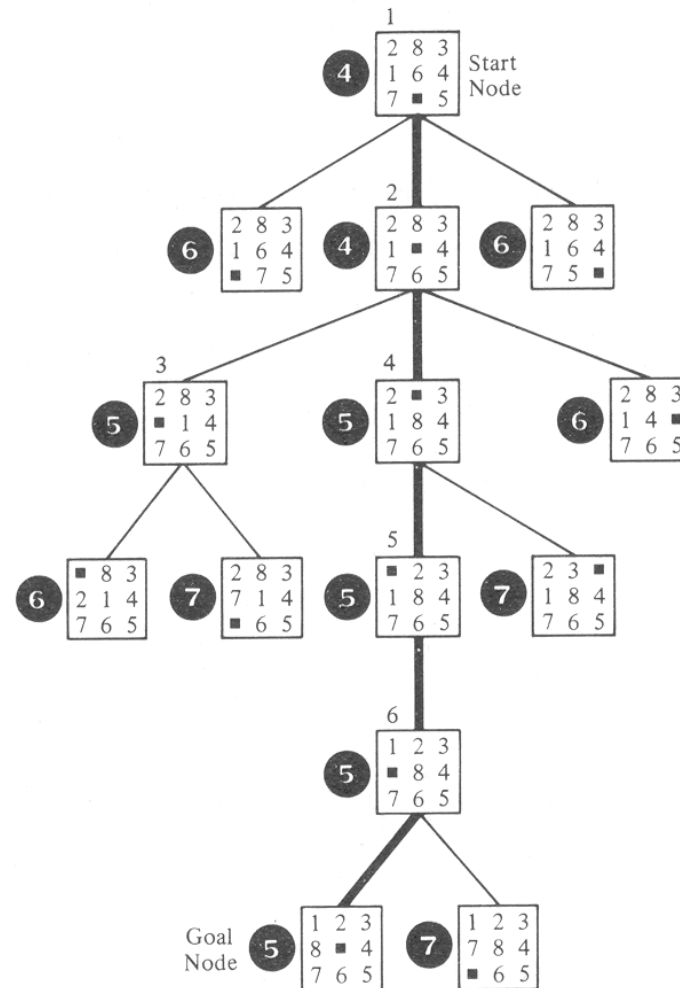


Fig. 2.8 A search tree using an evaluation function.

2. Řešení úloh

Podrobněji:

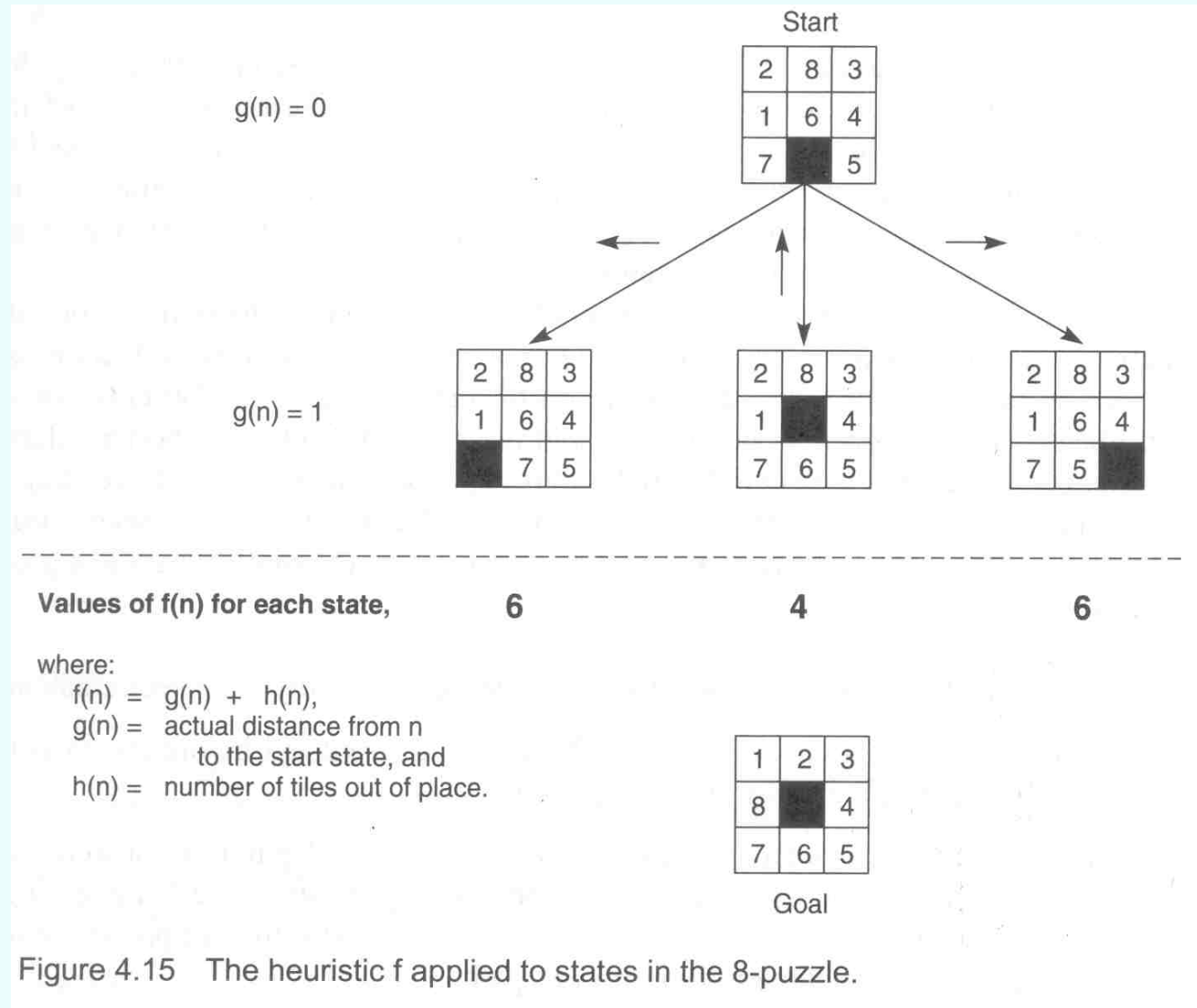


Figure 4.15 The heuristic f applied to states in the 8-puzzle.

2. Řešení úloh

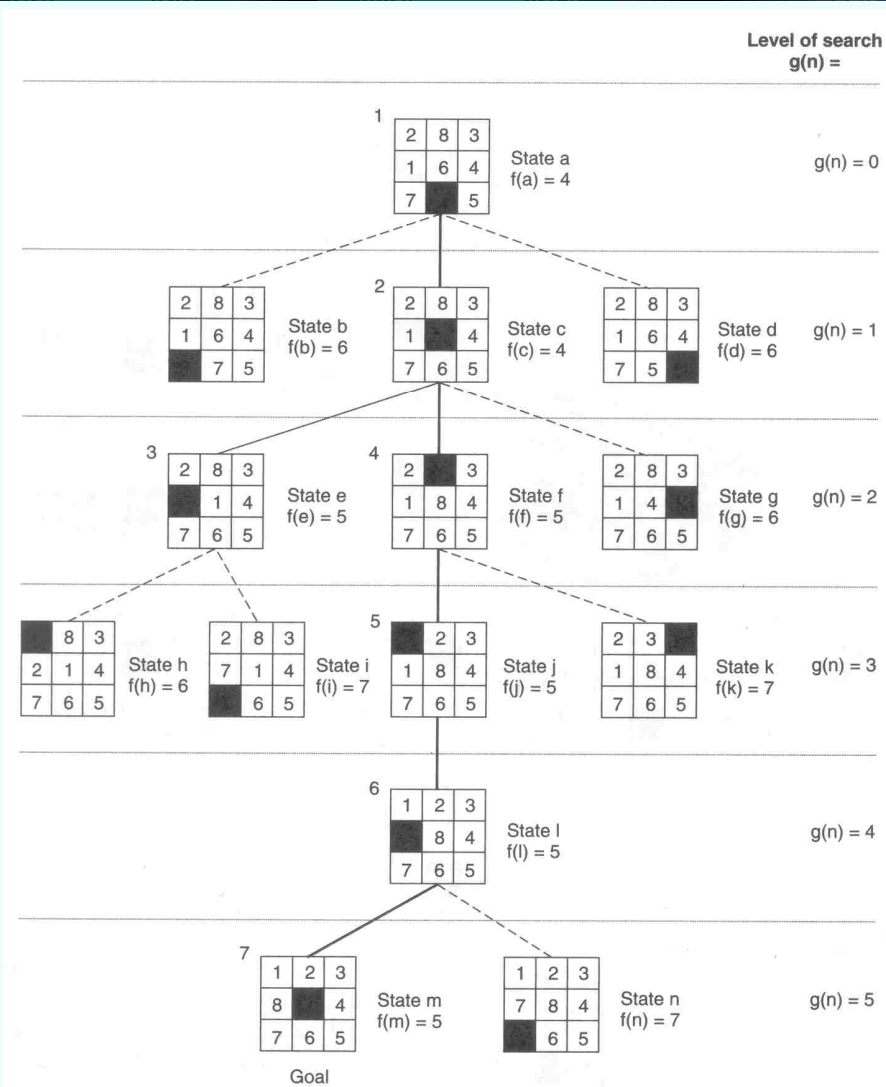


Figure 4.16 State space generated in heuristic search of the 8-puzzle graph.

2. Řešení úloh

Algoritmus prohledávání A, A*

Modifikace základního algoritmu hledání v grafu

Za optimální cestu v grafu budeme považovat cestu s minimální „cenou“ – definujeme proto ohodnocující funkci uzlu $f(n_i)$, která poskytne skutečnou (minimální) „cenu“ cesty z výchozího uzlu do cílového uzlu vedoucí přes uzel n_i :

$$f^*(n_i) = g^*(n_i) + h^*(n_i) ,$$

kde $g^*(n_i)$ je skutečná (minimální) „cena“ cesty z výchozího uzlu n_0 do uzlu n_i ,

$h^*(n_i)$ je skutečná (minimální) „cena“ cesty z uzlu n_i do cílového uzlu n_g .

Zřejmě $f^*(n_0) = h^*(n_0)$ a $f^*(n_g) = g^*(n_g)$.

2. Řešení úloh

Optimální cestu z n_0 do n_g apriorně neznáme, tedy nelze exaktně určit ani hodnotu $f^*(n_i)$ pro kterýkoli uzel n_i . Použijeme proto odhad „ceny“ optimální cesty ve tvaru

$$f'(n_i) = g'(n_i) + h'(n_i),$$

kde $g'(n_i)$ je odhad „ceny“ optimální cesty z výchozího uzlu n_0 do uzlu n_i (odhad hodnoty funkce $g^*(n_i)$),

$h'(n_i)$ je odhad „ceny“ optimální cesty z uzlu n_i do cílového uzlu n_g (odhad hodnoty funkce $h^*(n_i)$).

$g'(n_i)$ je zřejmě odhad „ceny“ cesty do uzlu n_i , což vyčíslíme jako součet ohodnocení všech hran na cestě z n_0 do n_i a zjevně platí

$$g'(n_i) \geq g^*(n_i).$$

2. Řešení úloh

Stanovení $h'(n_i)$ je obtížné, nazýváme ji ryze heuristickou funkcí.

Definice: Základní algoritmus hledání v grafu s ohodnocující funkcí vyčíslenou podle vzorce

$$f'(n_i) = g'(n_i) + h'(n_i)$$

nazýváme algoritmem **A** ; je-li $h'(n_i)$ nezáporným (v optimálním případě „nejtěsnějším“) dolním odhadem funkce $h^*(n_i)$, hovoříme o „optimálním“ algoritmu **A*** (platí pro $0 \leq h'(n_i) \leq h^*(n_i)$).

Na ryze heuristické funkci $h'(n_i)$ zpravidla vyžadujeme její monotónnost.

Příklad: „8“ s ryze heuristickou funkcí definovanou jako

$$h'(n_i) = P(n_i) + 3 * S(n_i), \quad \text{kde}$$

$P(n_i)$ je součet vzdáleností každého kamene od svého cílového místa v možných posuvech, $S(n_i)$ je míra porušení pořadí kamenů.

2. Řešení úloh

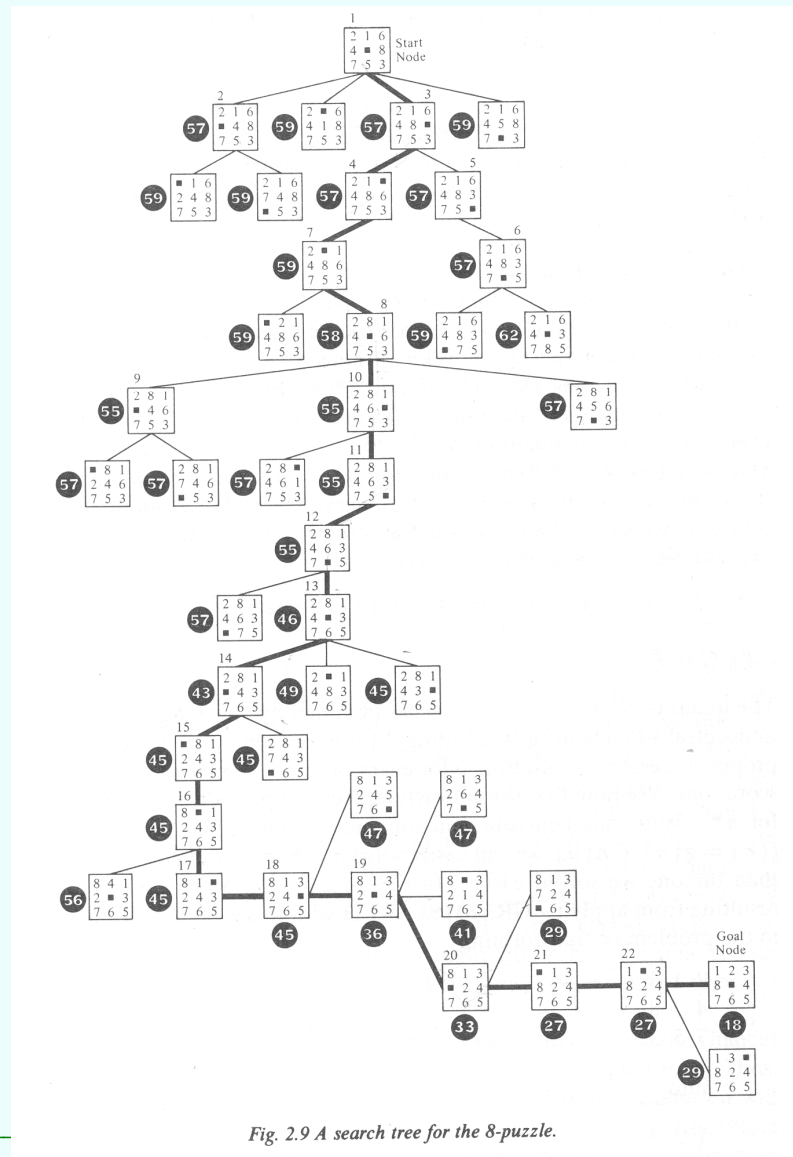


Fig. 2.9 A search tree for the 8-puzzle.

2. Řešení úloh

Efektivnost algoritmu hledání řešení úlohy

(algoritmu prohledávání grafu řešení úlohy)

Definice: Více informovaným algoritmem rozumíme algoritmus s ryze heuristickou funkcí $h'(n_i)$, která je „těsnějším“ dolním odhadem funkce $h^*(n_i)$ – více informovaný algoritmus vyžaduje přesnější heuristickou informaci.

2. Řešení úloh

Efektivnost algoritmu hledání řešení úlohy

(algoritmu prohledávání grafu řešení úlohy)

Definice: Více informovaným algoritmem rozumíme algoritmus s ryze heuristickou funkcí $h'(n_i)$, která je „těsnějším“ dolním odhadem funkce $h^*(n_i)$ – více informovaný algoritmus vyžaduje přesnější heuristickou informaci.

Efektivnost algoritmu prohledávání je dána:

- „cenou“ cesty z výchozího uzlu n_0 do cílového uzlu n_g ,
- počtem vygenerovaných uzlů v grafu řešení úlohy,
- algoritmickou složitostí výpočtu heuristické funkce.

2. Řešení úloh

Modifikace informovaných strategií hledání řešení úlohy:

a) Algoritmus uspořádaného prohledávání (best-first search algorithm)

Účinného informovaného prohledávání lze dosáhnout algoritmem uspořádaného prohledávání (angl. best-first search algorithm), který vznikl rozšířením klasického gradientního algoritmu o paměť. Tato paměť se opět skládá opět ze seznamů OPEN a CLOSED. Prvky těchto seznamů nejsou však pouhá jména uzlů (stavů), nýbrž trojice:

< jméno uzlu, hodnota f , jméno rodičovského uzlu >

Zápisem stavu v seznamu pak rozumíme zápis uvedené trojice.

2. Řešení úloh

Algoritmus uspořádaného prohledávání:

1. Počáteční stav zapiš do seznamu **OPEN**, seznam **CLOSED** je prázdný.
2. Pokud je seznam **OPEN** prázdný, řešení neexistuje, ukonči prohledávání.
3. Ze seznamu **OPEN** vyber stav i s nejmenší hodnotou $f(i)$. V případě většího počtu stavů se stejnou minimální hodnotou $f(i)$ proveř, zda některý z těchto stavů není stavem cílovým, v takovém případě jej vyber; jinak vyber mezi stavy se stejnou minimální hodnotou $f(i)$ libovolně.
4. Vymaž stav i ze seznamu **OPEN** a zařaď jej do seznamu **CLOSED**.
5. Je-li stav i cílovým stavem, řešení je nalezeno, ukonči prohledávání.
6. Expanduj stav i ; pro každého následovníka j stavu i vypočítej hodnotu $f(j)$. Pokud stav j není ani v seznamu **OPEN** ani v seznamu **CLOSED**, zařaď jej do seznamu **OPEN**. Pokud je stav j již v seznamu **OPEN** nebo **CLOSED**, avšak s ohodnocením větším než právě vypočtené $f(j)$, změň jeho ohodnocení na $f'(j)$, změň jméno rodičovského uzlu v zápisu uzlu a zařaď ho do seznamu **OPEN**.
7. Pokračuj krokem č. 2.

2. Řešení úloh

Vlastnosti uspořádaného prohledávání:

- Jako hodnotící funkci použijeme: $f(i) = h(i)$.
- Uzly jsou uspořádány tak, že ty s nejlepším ohodnocením jsou expandovány jako první.

Výhody uspořádaného prohledávání:

- Nejrychlejší (u jednoduchých typů úloh).

Nevýhody uspořádaného prohledávání:

- Není zaručeno nalezení nejkratší cesty.
- Nebere v úvahu náročnost úlohy.

2. Řešení úloh

b) Paprskové prohledávání s aperturou n (beam search)

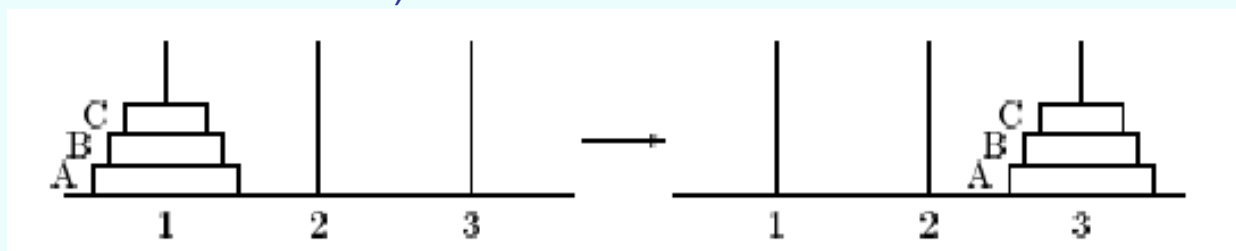
Další variantou obecného algoritmu prohledávání je algoritmus paprskového prohledávání. Užívá seznamu OPEN konečné délky n , přičemž v každém kroku se do seznamu OPEN zapíše jen ty nově expandované uzly, které mají lepší ohodnocení než uzly dosud zachycené v seznamu OPEN, ty se tím automaticky vyškrtnou. Tedy pouze n nejslibnějších uzlů se uchovává v seznamu OPEN, což má samozřejmě za následek značné zredukování expandovaného stromu, což může zabránit nalezení optimálního řešení.

Takto modifikovaný algoritmus je užíván při prohledávání stavového prostoru vytvořeného při rozpoznávání mluveného slova (určování jednotlivých vyslovených slov).

2. Řešení úloh

Rozklad úlohy na podúlohy

Mějme danu úlohu přemísťování hanojských věží podle obrázku. V počátečním stavu úlohy je hanojská věž skládající se z N kotoučů o různých průměrech situována na levém kolíku, který označme 1. Úkolem úlohy je přemístit jednotlivé kotouče na pravý kolík (označený 3) pomocí středního kolíku 2 tak, že se smí vždy přemísťovat jen vrchní kotouč a žádný kotouč nesmí nikdy ležet na kotouči menšího průměru. Kotouče označme podle velikosti (průměru) od největšího po nejmenší písmeny A, B, C, ..., N (pod symbolem N si představme nejvyšší písmeno označující nejmenší kotouč, např. pro věž o třech kotoučích $N = C$) :



Libovolný stav úlohy reprezentujeme seznamem ["1", "2", "3"], kde symboly "1", "2", "3" představují seznamovou reprezentaci uložení kotoučů na kolíku "1", "2" či "3" v pořadí zdola nahoru. Nenachází-li se na kolíku žádný kotouč, bude seznam prázdný (reprezentován *nil*).

2. Řešení úloh

Vyobrazená úloha se pak dá symbolicky zapsat jako

$$s_0 \equiv [[A, B, C], nil, nil] \rightarrow [nil, nil, [A, B, C]] \equiv s_g .$$

2. Řešení úloh

Vyobrazená úloha se pak dá symbolicky zapsat jako

$$s_0 \equiv [[A, B, C], nil, nil] \rightarrow [nil, nil, [A, B, C]] \equiv s_g .$$

Úlohu symbolicky rozložíme na tři dílčí úlohy:

1. $[[A, B, C], nil, nil] \rightarrow [X, Y, [A]]$
2. $[X, Y, [A]] \rightarrow [X', Y', [A, B]]$
3. $[X', Y', [A, B]] \rightarrow [nil, nil, [A, B, C]]$

2. Řešení úloh

Vyobrazená úloha se pak dá symbolicky zapsat jako

$$s_0 \equiv [[A, B, C], nil, nil] \rightarrow [nil, nil, [A, B, C]] \equiv s_g .$$

Úlohu symbolicky rozložíme na tři dílčí úlohy:

1. $[[A, B, C], nil, nil] \rightarrow [X, Y, [A]]$
2. $[X, Y, [A]] \rightarrow [X', Y', [A, B]]$
3. $[X', Y', [A, B]] \rightarrow [nil, nil, [A, B, C]]$

Jiný způsob rozkladu úlohy na dílčí úlohy můžeme zapsat např.:

1. $[[A, B, C], nil, nil] \rightarrow [[A], [B, C], nil]$
2. $[[A], [B, C], nil] \rightarrow [nil, [B, C], [A]]$
3. $[nil, [B, C], [A]] \rightarrow [nil, nil, [A, B, C]]$

2. Řešení úloh

Konjunktivně – disjunktivní (AND / OR) graf

Účel: Znázorňuje symbolicky rozklad úlohy na podúlohy.

Definice: Konjunktivně – disjunktivní graf (AND/OR graf nebo také transformační graf) je orientovaný acyklický graf s uzly, které nejsou listy, dvojího typu:

- **AND-uzel** představuje konjunkci podúloh, tj. k vyřešení (pod)úlohy reprezentované AND-uzlem je zapotřebí, aby každá z podúloh byla řešitelná (a vyřešena). Hrany vycházející z AND-uzlu jsou pro rozlišení typu uzlu spojeny obloučkem.
- **OR-uzel** představuje disjunkci podúloh, tj. k vyřešení (pod)úlohy reprezentované OR-uzlem stačí, aby alespoň jedna z podúloh byla řešitelná (a vyřešena).

2. Řešení úloh

Příklad: Reprezentujte následující soustavu logických formulí (v nichž A, B, \dots, Z označují výroky) konjunktivně – disjunktivním grafem:

$$(1) \quad A \wedge B \rightarrow M$$

$$(2) \quad C \rightarrow M$$

$$(3) \quad D \wedge B \rightarrow N$$

$$(4) \quad E \wedge B \rightarrow P$$

$$(5) \quad C \wedge G \rightarrow R$$

$$(6) \quad H \rightarrow R$$

$$(7) \quad E \wedge F \wedge G \rightarrow Q$$

$$(8) \quad M \rightarrow X$$

$$(9) \quad N \rightarrow X$$

$$(10) \quad P \rightarrow Y$$

$$(11) \quad Q \rightarrow Y$$

$$(12) \quad N \wedge R \rightarrow Z$$

2. Řešení úloh

Příklad: Reprezentujte následující soustavu logických formulí (v nichž A, B, \dots, Z označují výroky) konjunktivně – disjunktivním grafem:

$$(1) \quad A \wedge B \rightarrow M$$

$$(2) \quad C \rightarrow M$$

$$(3) \quad D \wedge B \rightarrow N$$

$$(4) \quad E \wedge B \rightarrow P$$

$$(5) \quad C \wedge G \rightarrow R$$

$$(6) \quad H \rightarrow R$$

$$(7) \quad E \wedge F \wedge G \rightarrow Q$$

$$(8) \quad M \rightarrow X$$

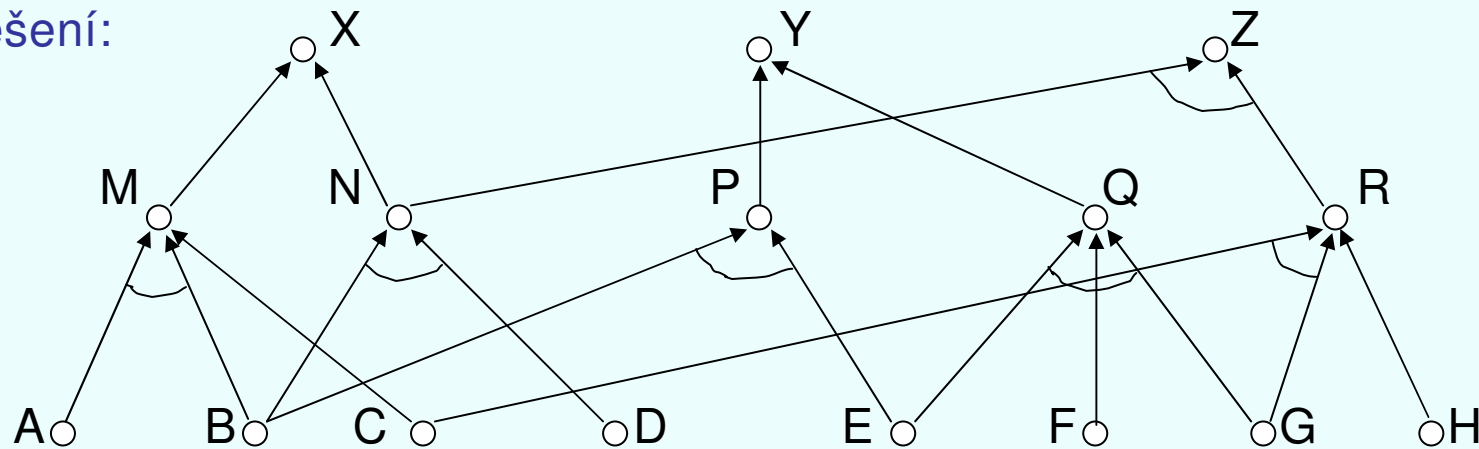
$$(9) \quad N \rightarrow X$$

$$(10) \quad P \rightarrow Y$$

$$(11) \quad Q \rightarrow Y$$

$$(12) \quad N \wedge R \rightarrow Z$$

Řešení:



2. Řešení úloh

Algoritmus rozkladu úloh na podúlohy

Předpokládejme libovolný stav s_i úlohy G popsany

$$s_i = stav (s_i^1, s_i^2, \dots, s_i^M),$$

kde „stav“ je funktor reprezentující strukturu stavu s_0 , která má M složek (argumentů). Úlohu G můžeme zjednodušeně zapsat

$$G: s_0 \rightarrow s_g,$$

kde s_0 je stav výchozí a s_g stav koncový (cílový). Pak lze úlohu přepsat do tvaru

$$G: stav (s_0^1, s_0^2, \dots, s_0^M) \rightarrow stav (s_g^1, s_g^2, \dots, s_g^M).$$

Poznámka: Pro $s_0^m = s_g^m$ (pro každé m) platí, že úloha je vyřešena. To, co činí úlohu úlohou určenou k řešení, je nesouhlas odpovídajících si složek struktury stavu (argumentů funkce *stav*).

2. Řešení úloh

Nesouhlas mezi složkami s_0^m a s_g^m (pro každé m) se nazývá m -tá **diference stavu**. Pro některé druhy výrazů (příp. funktorů) lze diferenci vyjádřit číselně, jindy strukturně nebo logicky (musíme se spokojit s konstatováním, že „diference je či není“).

Úloha bude vyřešena, nalezneme-li takový kompoziční operátor R_{Kog} , který odstraní všechny diference, tj. postupně odstraní první a druhou a třetí a ... a M -tou diferenci. Tím dojde k rozkladu úlohy G na M podúloh:

$$G_1: stav (s_0^1, s_0^2, \dots, s_0^M) \rightarrow stav (s_g^1, s_0^2, \dots, s_0^M),$$

$$G_2: stav (s_g^1, s_0^2, \dots, s_0^M) \rightarrow stav (s_g^1, s_g^2, \dots, s_0^M),$$

...

$$G_M: stav (s_g^1, s_g^2, \dots, s_0^M) \rightarrow stav (s_g^1, s_g^2, \dots, s_g^M).$$

Pozor: Pravidla pro odstraňování diferencí mívají vedlejší efekty !

2. Řešení úloh

Př.: Rozklad řešení integrálu:

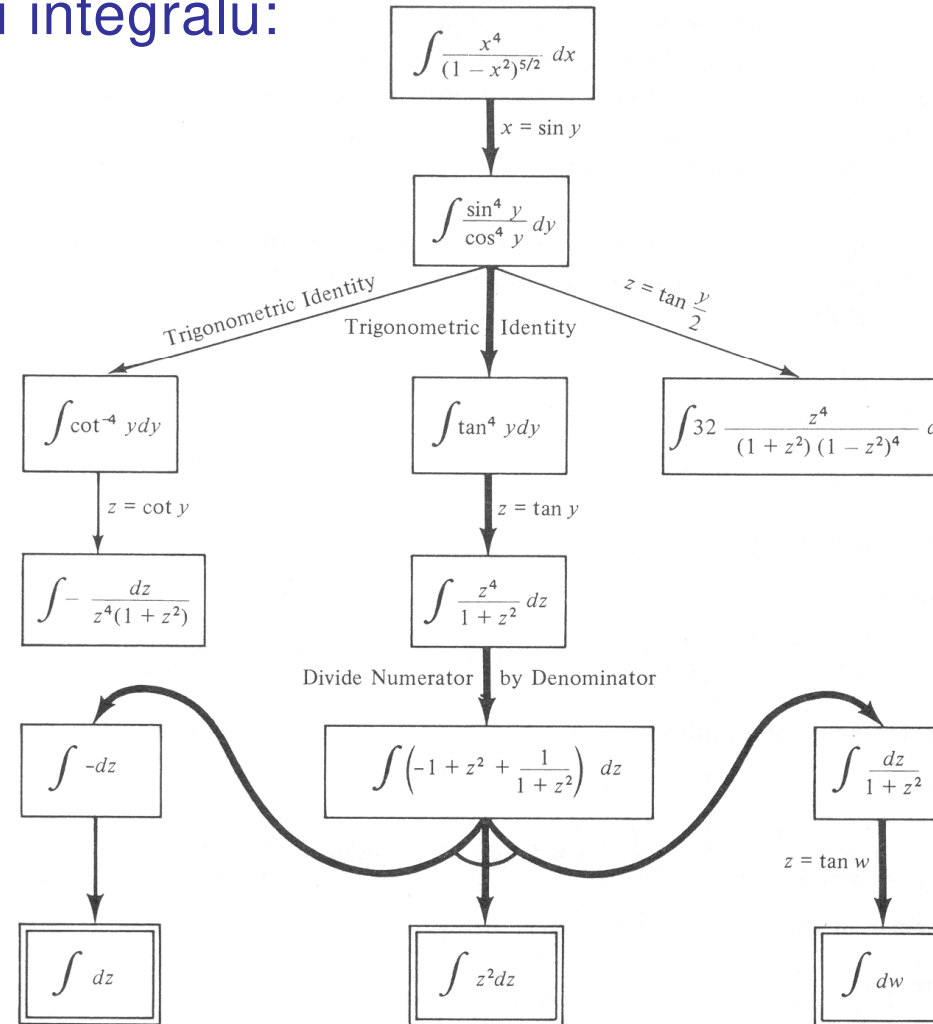
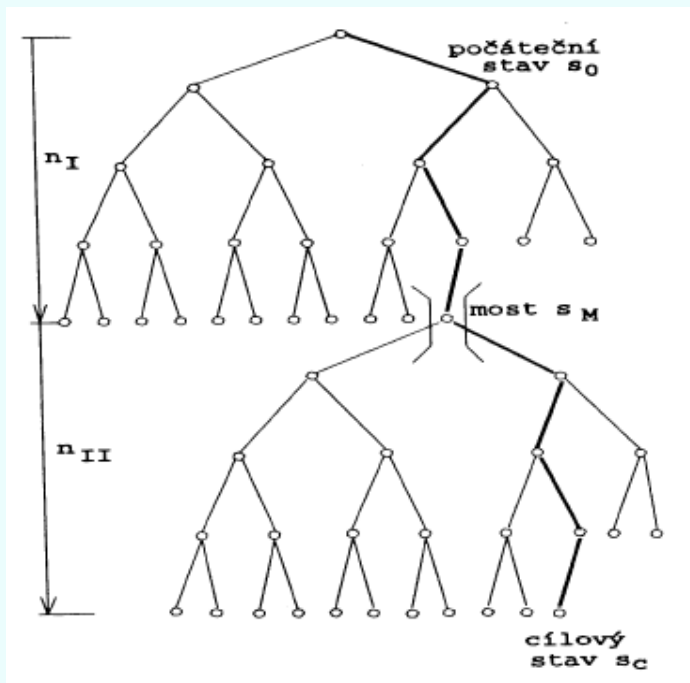


Fig. 1.13 An AND/OR tree for an integration problem.

2. Řešení úloh

Rozklad úlohy na podúlohy detailněji:

Předpokládejme, že řešení úlohy musí procházet mezilehlým stavem, tzv. mostem. Pak je možné úlohu rozdělit na dvě části – část I s počátečním stavem s_0 a cílovým s_M a část II s počátečním stavem s_M a cílovým $s_{c(g)}$. Uvažujme nadále slepé prohledávání stavového prostoru. Pak je vygenerováno maximálně $2^{n_I} + 2^{n_{II}}$ stavů, kde horní indexy reprezentují hloubku stromu.



Evidentně platí, že $2^{n_I} + 2^{n_{II}} < 2^{n_I + n_{II}}$, přičemž úspora prohledávání může být značná.

Vliv rozkladu na rozsah prohledávaného stavového prostoru

2. Řešení úloh

Jak bylo řečeno dříve, pravidla pro odstraňování diferencí mívají *vedlejší efekty*.

Nechť je každý stav popsán R parametry – $s_i = (s_{i,1}, \dots, s_{i,R})$. Potom můžeme úlohu U formulovat takto:

$$U: s_0 = (s_{0,1}, \dots, s_{0,R}) \rightarrow s_c = (s_{c,1}, \dots, s_{c,R}).$$

Pokud si všechny stavy odpovídají, je úloha vyřešena. U netriviálních úloh se však vyskytují *diference* některých či všech složek. Úloha je vyřešena až tehdy, nalezneme-li takové operátory, které diference odstraní. Možnost rozkladu úlohy na R podúloh, z nichž první odstraňuje diference u první složky, druhá u druhé atd. nemusí být použitelná vlivem *vedlejších (postranních) efektů* jednotlivých operátorů. Kromě diference, kterou operátor odstraní, přidá či zvětší jinou diferenci, navíc není známo, jaké operátory budou použity v dalším procesu úprav \rightarrow není možné apriorně přesně stanovit cílové stavy podúloh (mosty) a tudíž ani počáteční stavy dalších navazujících podúloh.

2. Řešení úloh

Rozklad úlohy na podúlohy řeší rozklad systému reprezentujícího úlohu na podsystemy podle hledisek daných úlohou. Přitom je třeba dbát na dodržení následujících postulátů:

- postulát integrity – nedojde ke ztrátě jakékoli části systému,
- postulát soudržnosti – z dekomponovaného systému lze opět sestavit systém původní,
- postulát rovnoměrnosti – dekomponované části jsou přibližně stejně složité.

Všeobecně rozlišujeme dekompozici topologickou, funkční, věcnou a hierarchickou.

2. Řešení úloh

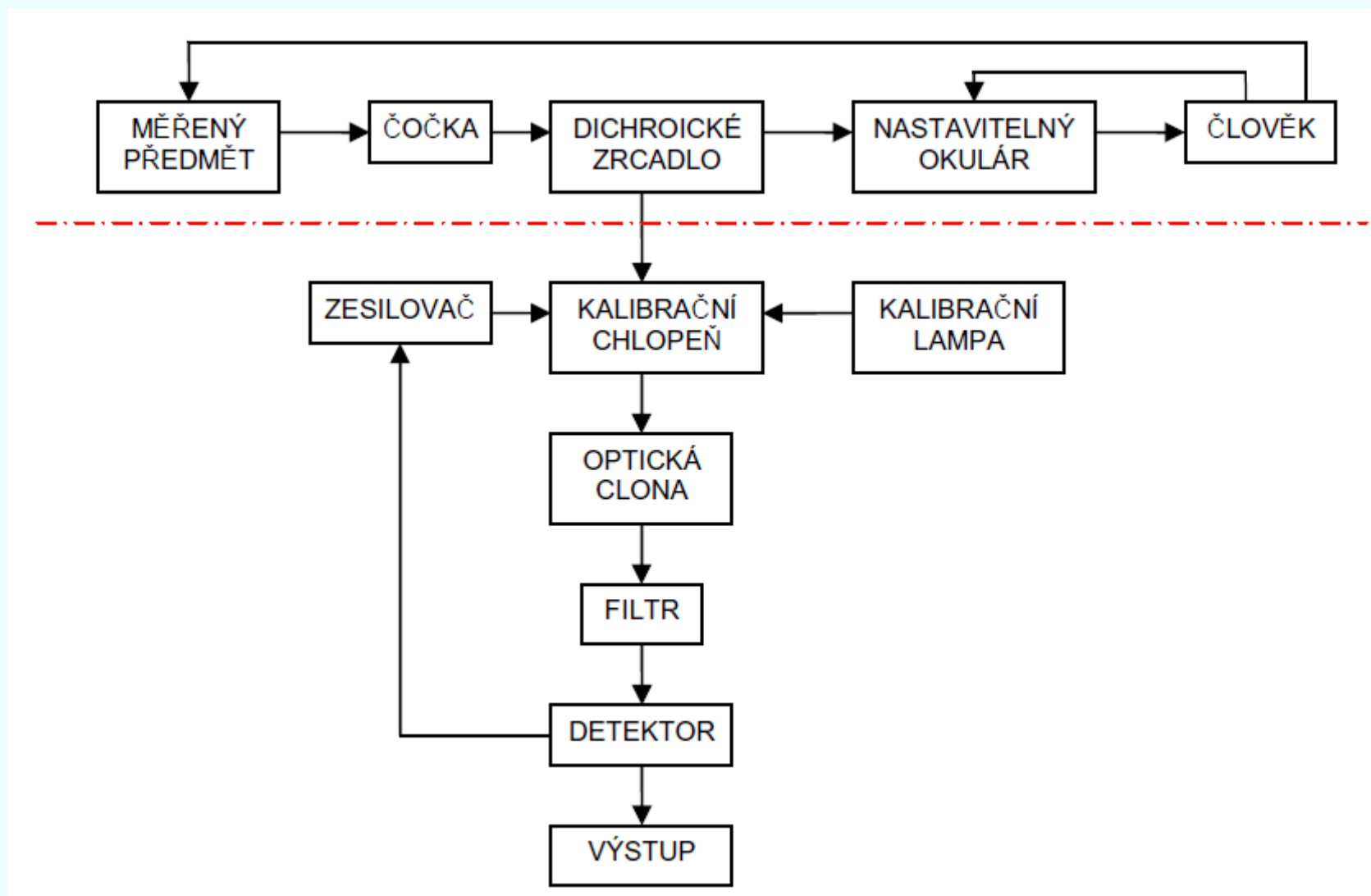
Topologická dekompozice

Hlavním hlediskem topologické dekompozice je minimalizace vazeb dekomponovaných částí systémů s okolím. Při pohledu na schématické znázornění zvoleného systému a v souladu s postulátem rovnoměrnosti lze přímo provést následující topologickou dekompozici:

1. podsystém – prvky: MP, Č, DZ, NO, Č,
2. podsystém – prvky: KC, KL, Z, OC, F, D, V.

Grafické znázornění topologické dekompozice zvoleného systému (optického pyrometru) je na následujícím obrázku:

2. Řešení úloh



2. Řešení úloh

Funkční dekompozice

Funkční dekompozice je charakterizována tím, že jednotlivé dekomponované části vykonávají určité makrofunkce.

U zvoleného systému lze identifikovat dvě makrofunkce:

1. funkce měřicí a vyhodnocovací

– prvky: MP, Č, DZ, KC, KL, Z, OC, F, D, V

2. funkce kontrolní (člověk kontroluje, zda je předmět správně umístěn)

– prvky: MP, Č, DZ, NO, ČI

2. Řešení úloh

Věcná dekompozice

Věcné dekompozici se také někdy říká obsahová, případně sémantická a dekomponované části systému by měly být tvořeny prvky se stejnými vlastnostmi.

Zvolený systém lze rozčlenit do několika podsystémů z hlediska společných vlastností:

1. prvky, které nejsou přímo součástí měřicího zařízení: MP, ČI, V
2. optické prvky v zařízení: Č, DZ, NO, OC, F
3. prvky, které doplňují prvky optické tak, aby bylo měření efektivní –
– Z, KL, KC, D

2. Řešení úloh

Hierarchická dekompozice

Uplatňuje se, jak již název napovídá, u hierarchických systémů.

Dekompozice se pak provádí následujícími způsoby:

- **po vrstvách** – dekomponovaná část systému je tvořena prvky ze stejné hierarchické úrovně,
- **po větvích** – dekomponovaná část systému je tvořena prvky dané větve.

Poznámka: Hierarchická dekompozice není pro každý systém vhodná, jelikož ne každý systém představuje hierarchický systém, případně se v systému vyskytují zpětné vazby, které neodpovídají hierarchickému uspořádání systému.