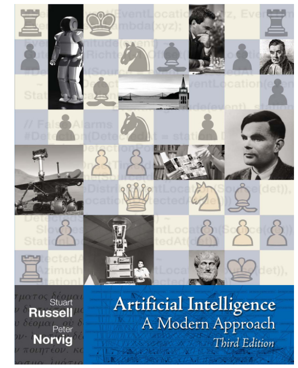
Informované metody prohledávání stavového prostoru

Michal Pěchouček, Milan Rollo

Department of Cybernetics
Czech Technical University in Prague

http://cw.felk.cvut.cz/doku.php/courses/a3b33kui/start

Použitá literatura pro umělou inteligenci



:: Artificial Intelligence: A Modern Approach (Third Edition) by Stuart Russell and Peter Norvig, 2007 Prentice Hall.

http://aima.cs.berkeley.edu/

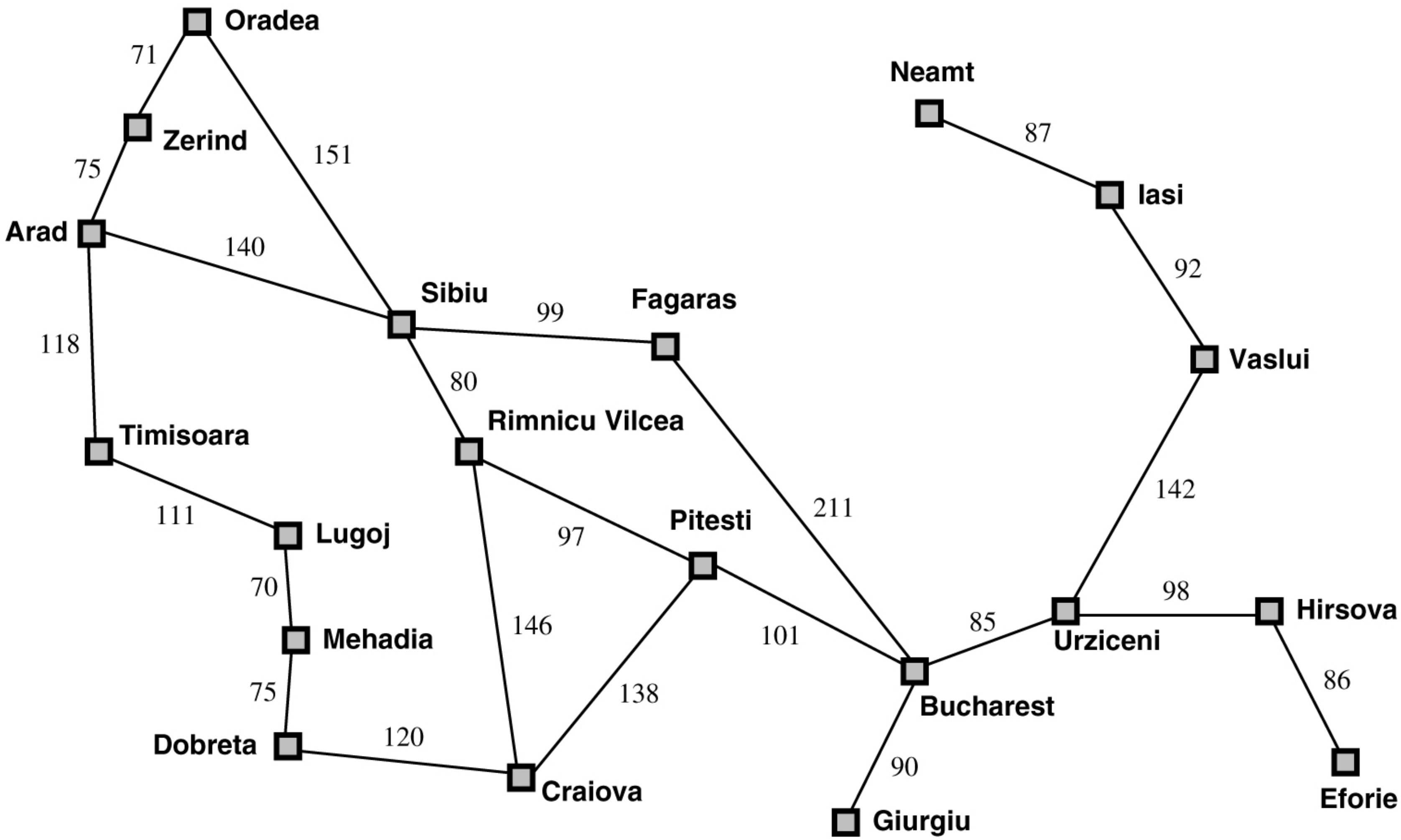
Informované metody prohledávání stavového prostoru

:: implementují efektivní metody nalezení optimálního řešení a využívají při tom kvalitativní informaci o různých stavech stavového prostoru.

Při prohledávání používáme (alespoň jednu z):

- konkrétní informaci o ceně daného stavu ve stavovém prostoru
- konkrétní informaci o ceně použití každého možného stavového operátoru
- heuristickou informaci, odhad o vhodnosti použití daného stavového s ohledem na efektivnost prohledávání stavového prostoru

:: Tyto informace používáme pro návrh **heuristického** algoritmu (také označovaného jako *Best-First-Search*), který vybírá vhodný uzel k expanzi. Takovýto uzel vede proces prohledávání k optimálnímu řešení. Funguje-li heuristický algoritmus dobře, minimalizuje prohledávání částí stavového prostoru, které nevedou k optimálnímu řešení.


Oradea
71
Zerind
151
75
Arad
140
Neamt
87
Iasi
92
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Vaslui
Timisoara
142
211
111
Lugoj
97
Pitesti
70
98
Hirsova
146
85
Urziceni
Mehadia
101
86
75
138
Bucharest
Dobreta
120
90
Craiova
Eforie
Giurgiu

Algoritmus uspořádaného prohledávání stavového prostoru

Návrh obecného algoritmu uspořádaného prohledávání stavového prostoru (Best-First Search) vychází z klasického algoritmu pro neinformované prohledávání stavového prostoru:

```
begin
    open := [Start], closed := []
    while (open <> []) do begin
        X := FIRST(open)
        closed := closed + [X], open := open - [X]
        if X = GOAL then return(SUCCESS)
        else begin
                E := expand(X)
                E := E - closed
               open := open + E
             end
       end
return(failure)
end.
```

Algoritmus uspořádaného prohledávání stavového prostoru

jen výběr **prvního** prvku na seznamu je nahrazen výběrem **nejlepšího** prvku seznamu

```
begin
    open :=  [Start], closed :=  []
    while (open <>  []) do begin
        X :=  BEST(open)
        closed := closed + [X], open := open - [X]
        if X = GOAL then return(SUCCESS)
        else begin
                E := expand(X)
                E := E - closed
               open := open + E
             end
   end
return(failure)
end.
```

Hodnotící Funkce

:: Když se algoritmus snaží vybírat nejlepší stav pro expanzi (např. s_n) z aktuálního stavu (např. s_m) pracuje s následujícími parametry:

- $c(m, n)$ – cena aplikace operátoru pro přechod ze stavu m do stavu n
- $g(m)$ – celková cena, součet cen všech operátorů aplikovaných z počátečního stavu až do stavu m
- $h(n)$ – reálná nebo odhadovaná celková cena, součet všech operátorů které je potřeba aplikovat ze stavu n do cílového stavu.

:: je tedy třeba navrhnout následující **hodnotící funkci** f, která bude sofistikovaně integrovat funkce c, g a h a zajistit, že má rozumné chování v konkrétní doméně.

Příklady hodnotící funkce

- **Gradientní prohledávání** (hill-climbing search)– kde $\forall m, n : f(m, n) = c(m, n)$
  - jednoduchá na implementaci, rychlá, odolná vůči zacyklení – nicméně často *uvízne v lokálním optimu* !!!

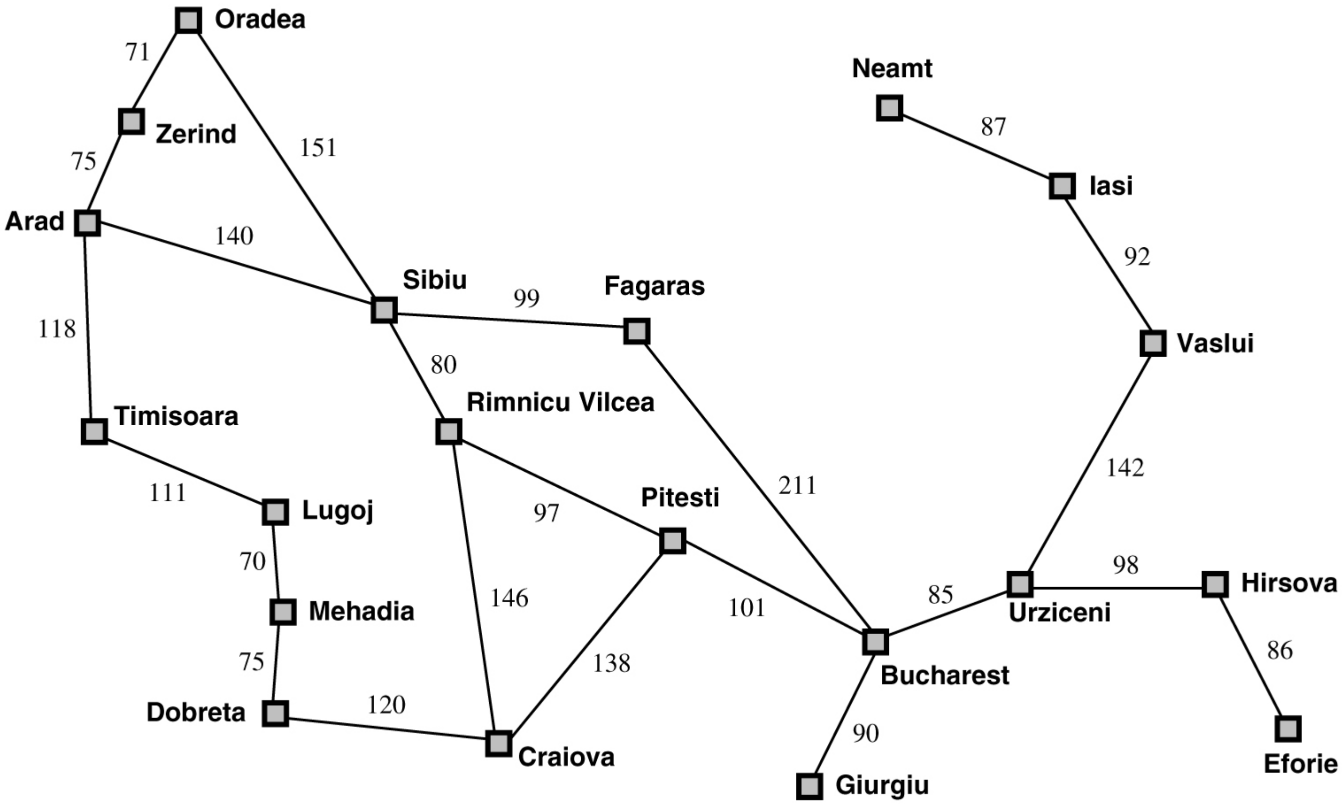
Příklady hodnotící funkce

- **Gradientní prohledávání** (hill-climbing search)– kde $\forall m, n : f(m,n) = c(m,n)$
  - jednoduchá na implementaci, rychlá, odolná vůči zacyklení – nicméně často *uvízne v lokálním optimu* !!!
- **Prohledávání do šířky** – $\forall m, n : c(m,n) = 1$ existuje-li hrana z m do n. Zde platí $f(m,n) = g(m) + 1$
  - minimalizuje počet kroků (hloubku) řešení

Příklady hodnotící funkce

- **Gradientní prohledávání** (hill-climbing search)– kde $\forall m, n : f(m, n) = c(m, n)$
  - jednoduchá na implementaci, rychlá, odolná vůči zacyklení – nicméně často *uvízne v lokálním optimu* !!!
- **Prohledávání do šířky** – $\forall m, n : c(m, n) = 1$ existuje-li hrana z m do n. Zde platí $f(m, n) = g(m) + 1$
  - minimalizuje počet kroků (hloubku) řešení
- **Hladový algoritmus** (greedy algoritmus)– $\forall m, n : f(m, n) = h(n)$ existuje-li hrana z m do n. Zde $h(n)$ je heuristický odhad vzdálenosti z uzlu n do cíle
  - neoptimální, neúplný

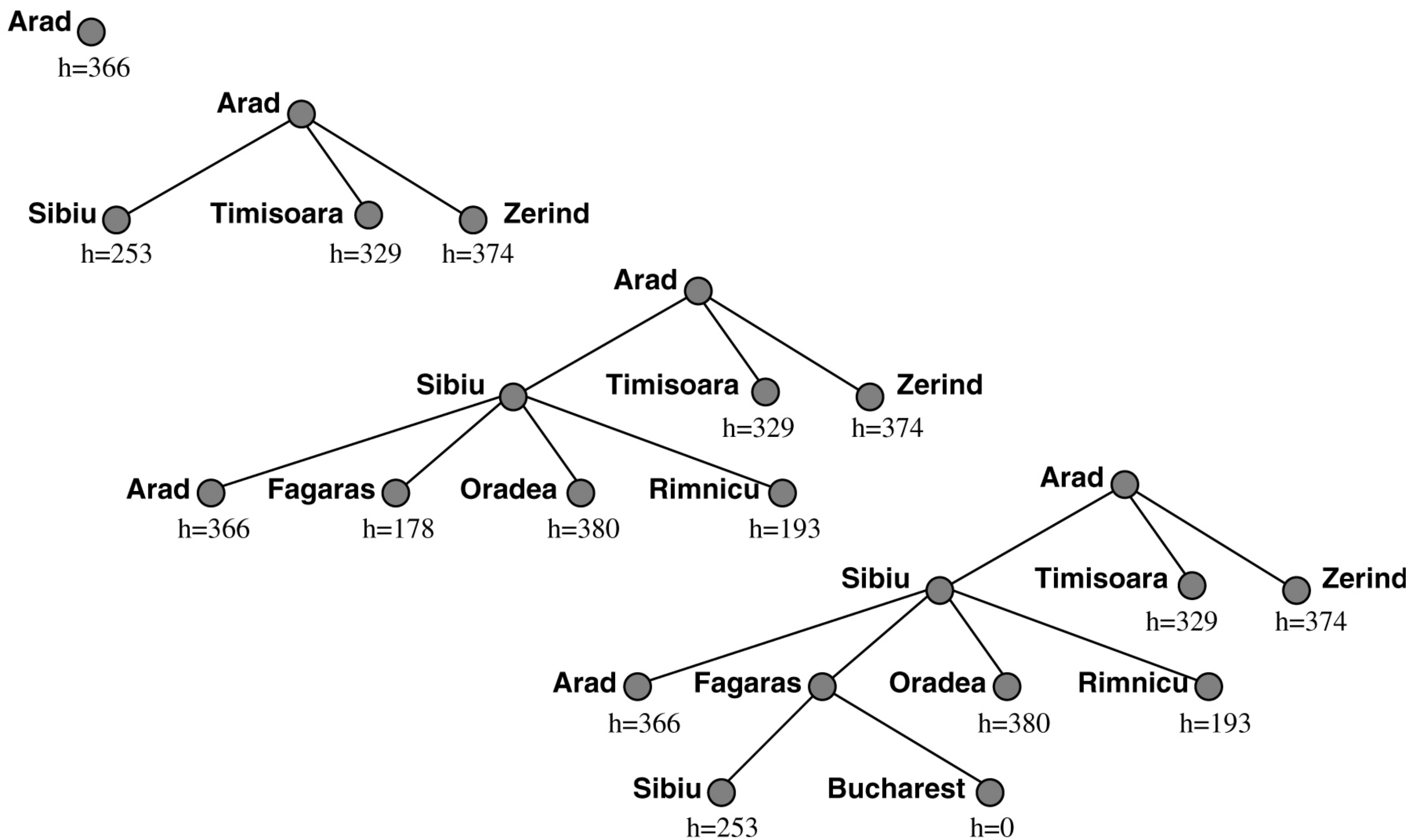
Hladový Algoritmus



Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Hladový Algoritmus



Algoritmus A^*

- Algoritmus A^* používá algoritmus uspořádaného prohledávání stavového prostoru, kde každý prvek je ohodnocen funkcí:
  - $f(n) = g(n) + h(n)$
  - pozn.: ohodnocení sčítající složky (c,g a h) ve tvaru $f(n,m) = c(m,n) + g(m) + h(n)$ lze napsat jako $f(n) = g(n) + h(n)$ protože $g(n) = g(m) + c(m,n)$, kde argument m již neovlivňuje hodnotu funkce.
- Nastavení hodnotící funkce f ve formě $f(n) = g(n) + h(n)$ je netriviální problém protože hodnotu $h(n)$ která není a priori známá je ji třeba odhadnout.
- Vzhledem k tomu, že optimalizujeme chování algoritmu, chceme aby byl odhad co nejpřesnější – aby hodnota $h(n)$ byla co nejblíže hodnotě $h^*(n)$ (přesná hodnota). Funkci $h(n)$ nazýváme **heuristická funkce**.
- Hodnotící funkce $f(n)$ je tedy odhadem přesných hodnot, které vrátí funkce $f^*(n) = g^*(n) + h^*(n)$ kde $g(n) = g^*(n)$ (ve většině případů)

Přípustnost Algoritmu A^*

- Jaké vlastnosti musí mít heuristická funkce $h(n)$? Co se stane když bude $h(n) > h^*(n)$? A co když $h(n) < h^*(n)$?

Přípustnost Algoritmu A^*

- Jaké vlastnosti musí mít heuristická funkce $h(n)$? Co se stane když bude $h(n) > h^*(n)$? A co když $h(n) < h^*(n)$?
- Aby se algoritmus choval rozumně, *tudíž aby našel jako první optimální řešení*, musí platit:

$$\forall n : 0 \leq h(n) \leq h^*(n)$$

- Je-li to pravda, říkáme, že heuristcká funkce je **přípustná**.
- Algoritmus A^* používá algoritmus uspořádaného prohledávání stavového prostoru, kde každý prvek je ohodnocen funkcí $f(n) = g(n) + h(n)$, kde $h(n)$ je přípustná
- A^* algoritmus vždy najde optimální řešení.

je BFS optimální?

Přípustnost Algoritmu A^*

- Jaké vlastnosti musí mít heuristická funkce $h(n)$? Co se stane když bude $h(n) > h^*(n)$? A co když $h(n) < h^*(n)$?
- Aby se algoritmus choval rozumně, *tudíž aby našel jako první optimální řešení*, musí platit:

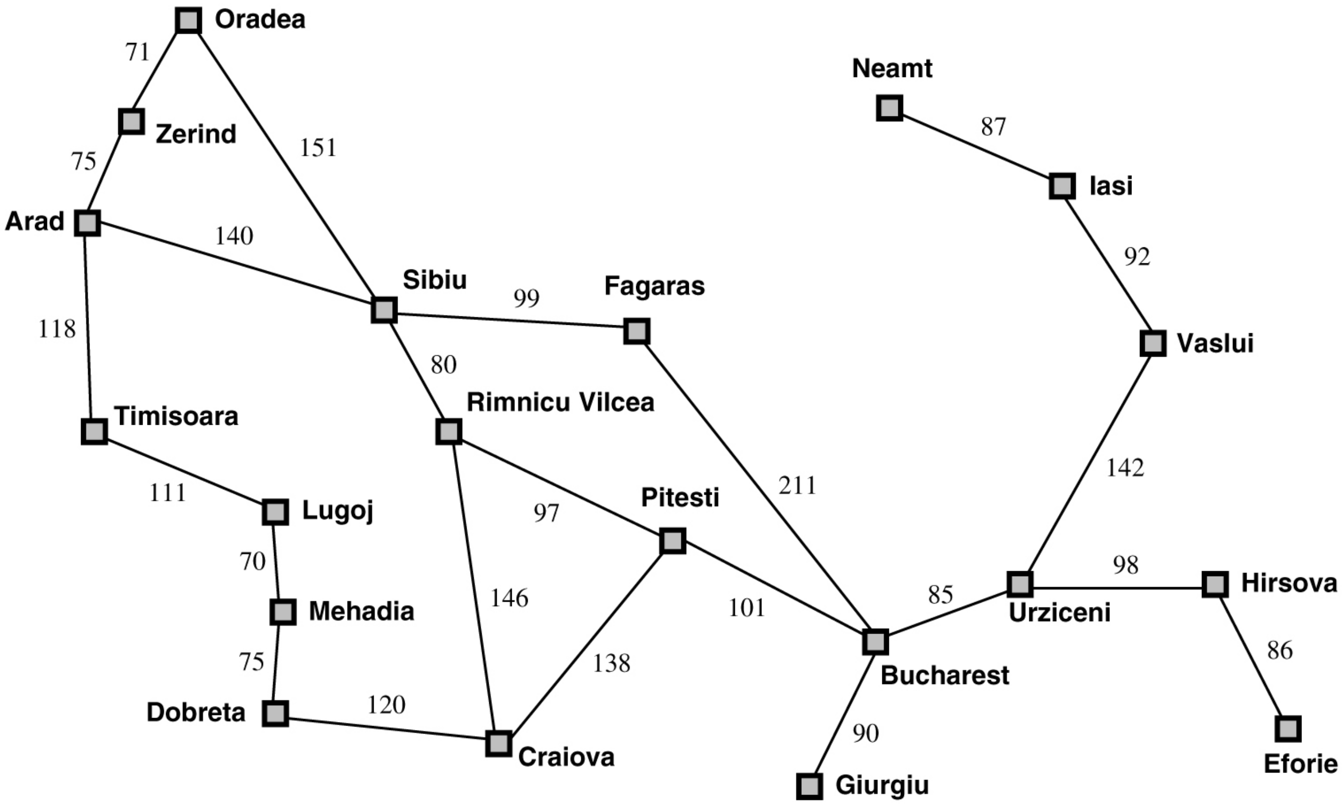
$$\forall n : 0 \leq h(n) \leq h^*(n)$$

- Je-li to pravda, říkáme, že heuristcká funkce je **přípustná**.
- Algoritmus A^* používá algoritmus uspořádaného prohledávání stavového prostoru, kde každý prvek je ohodnocen funkcí $f(n) = g(n) + h(n)$, kde $h(n)$ je přípustná
- A^* algoritmus vždy najde optimální řešení.

je BFS optimální?

ano, protože u BFS $f(n) = g(n) + h(n) = g(n)$. Tudíž $0 = h(n) < h^*(n)$ a platí, že heuristika je přípustná.

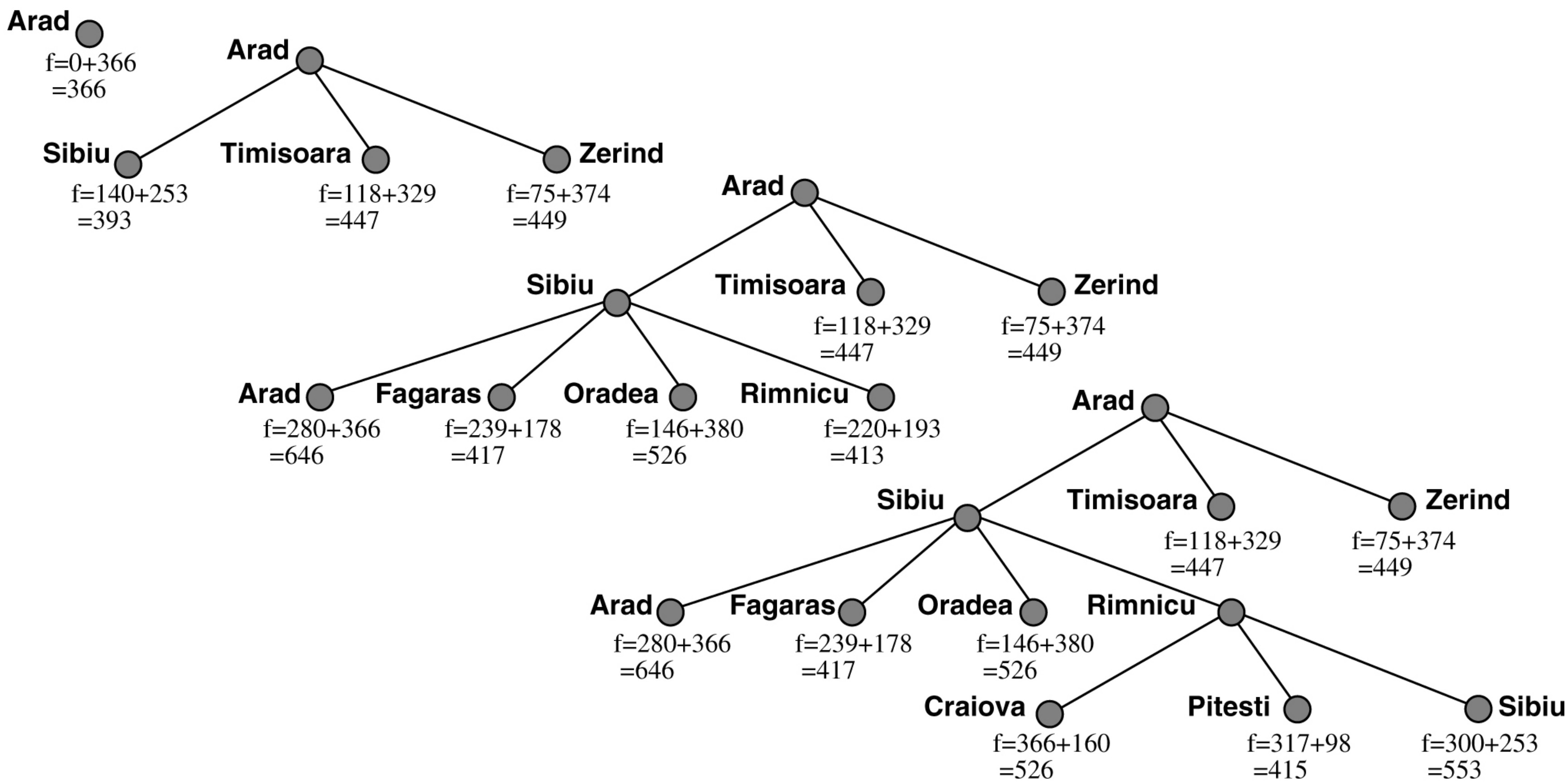
Příklad heuristik pro hledání cesty



Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Příklad heuristik pro hledání cesty



Doplňkové vlastnosti A*

```
begin
    open :=  [Start], closed :=  []
    while (open <>  []) do begin
        X :=  BEST(open)
        closed := closed + [X], open := open - [X]
        if X = GOAL then return(SUCCESS)
        else begin
                E := expand(X)
                E := E - closed
               open := open + E
             end
   end
return(failure)
end.
```

Doplňkové vlastnosti A*

upřesnění operací v A^* na řádku 9 - 10:

- v případě, že pro nějaký uzel $e \in$ `E` platí že se už objevuje v seznamu `open`
  - s hodnotou $f(e)$ lepší, pak se tento na řádku 10 do seznamu `open` nepřidá
  - s hodnotou $f(e)$ horší, pak se na řádku 10 ten horší odebere a lepší se přidá
- v případě, že pro nějaký uzel $e \in$ `E` platí že se už objevuje v seznamu `closed`
  - s hodnotou $f(e)$ lepší, pak se tento na řádku 9 ze seznamu `E` odebere
  - s hodnotou $f(e)$ horší, pak se tento na řádku 9 ze seznamu `E` neodebírá a naopak se odebere z `closed`.

Monotónnost

Heuristická funkce je **monotónní** (lokálně přípustná) platí-li

i. $\forall n_1, n_2$, kde n_1 expanduje do n_2: $h(n_1) - h(n_2) \leq cost(n_1, n_2)$,
   kde $c(n_1, n_2)$ opravdová cena z n_1 do n_2

ii. $h(\texttt{goal}) = 0$.

každá monotónní heuristická funkce je přípustná.

Monotónnost

Heuristická funkce je **monotónní** (lokálně přípustná) platí-li

i. $\forall n_1, n_2$, kde n_1 expanduje do n_2: $h(n_1) - h(n_2) \leq cost(n_1, n_2)$, kde $c(n_1, n_2)$ opravdová cena z n_1 do n_2

ii. $h(\texttt{goal}) = 0$.

každá monotónní heuristická funkce je přípustná.

Důkaz:

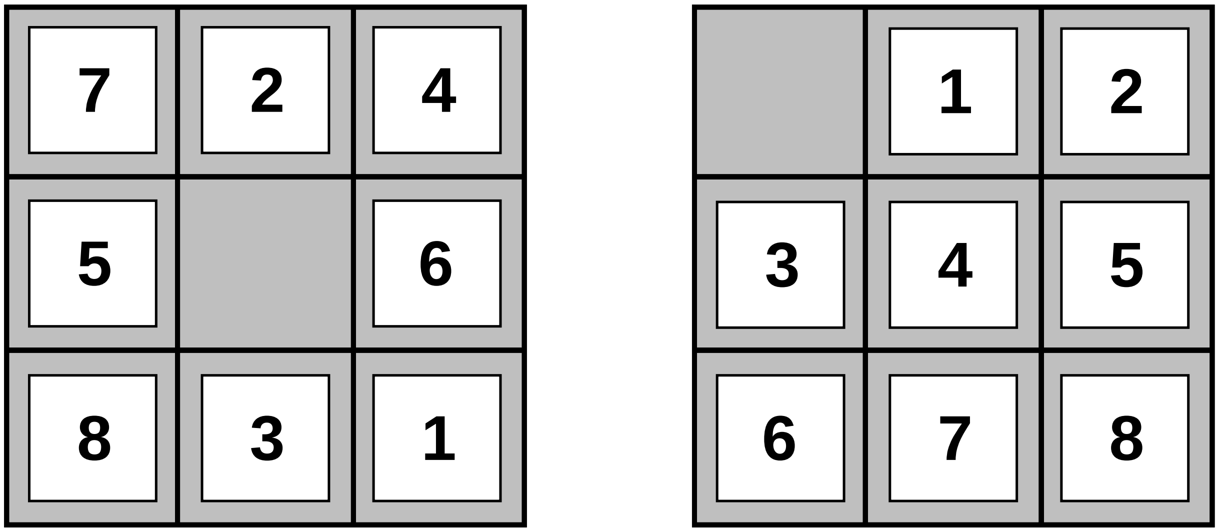
for $n_0 \to n_1$ ... $h(n_0) - h(n_1) \leq c(n_0, n_1)$ díky monotónosti
for $n_1 \to n_2$ ... $h(n_1) - h(n_2) \leq c(n_1, n_2)$ díky monotónosti
...
for $n_{k-1} \to \texttt{goal}$ ... $h(n_{k-1}) - h(\texttt{goal}) \leq c(n_{k-1}, \texttt{goal})$

je-li $h(\texttt{goal}) = 0$ pak po sečtení všech řádků platí $h(n_0) \leq c(n_0, \texttt{goal})$

Dominance (Informovanost) Heuristiky

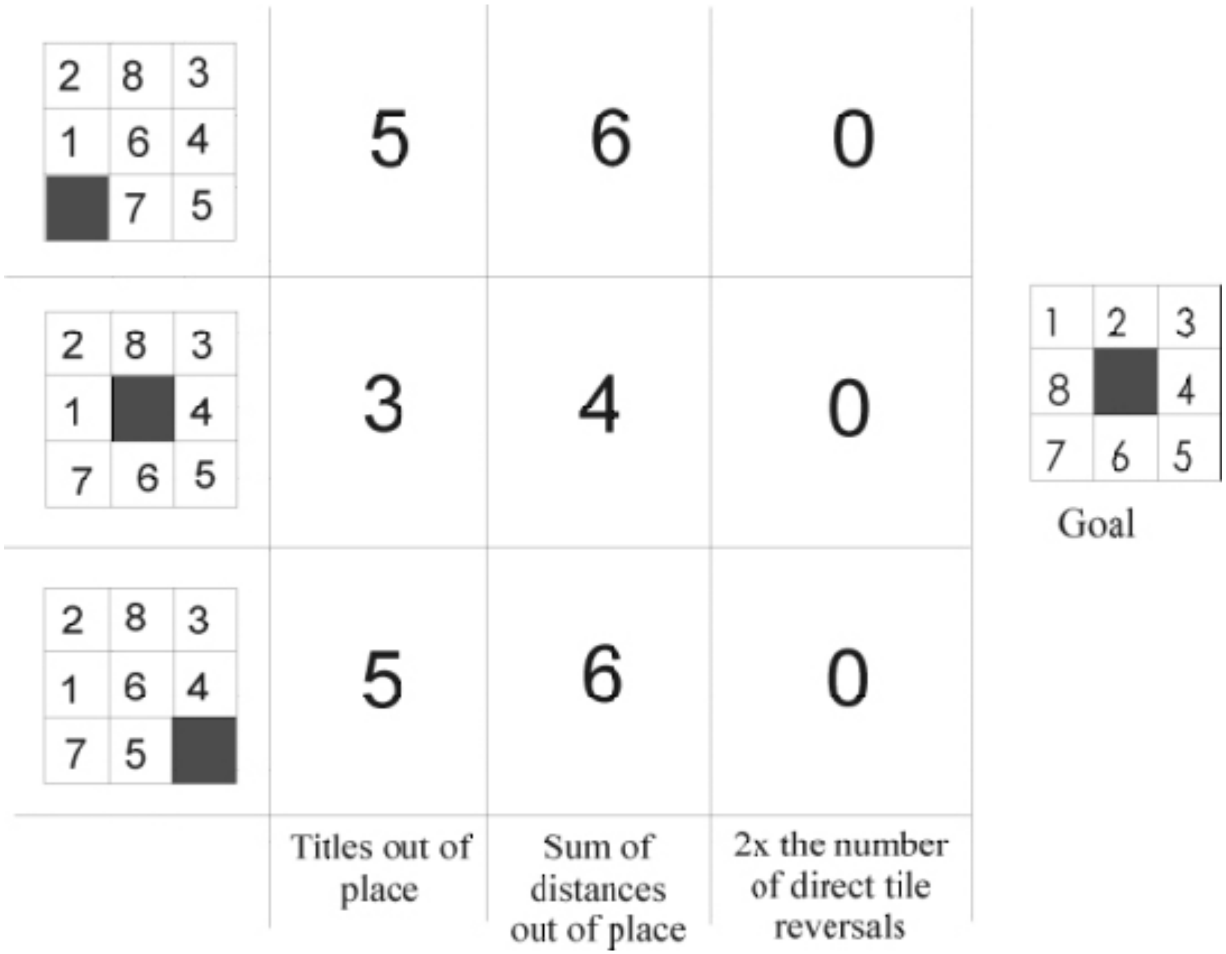
- Máme-li dvě přípustné heuristiky h_1 a h_2 tak, že $\forall n : h_1(n) \leq h_2(n)$, pak říkáme, že heuristika h_2 **dominuje** (je více informovaná) než h_1.
  - Obě heuristiky naleznou optimální řešení, ale h_2 potřebuje expandovat menší prostor než h_1.
- Je třeba dát pozor na to aby výpočet nebo použití hodně informované heuristiky nezabral více času než prohledání větší části stavového prostoru.

Heuristiky pro 8-puzzle



Start State **Goal State**

Heuristiky pro 8-puzzle



Effective Branching Factor: metrika popisující stavový prostor b^* je-li pro hloubku d a celkový počet expandovaných uzlů N pak musí platit $N = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

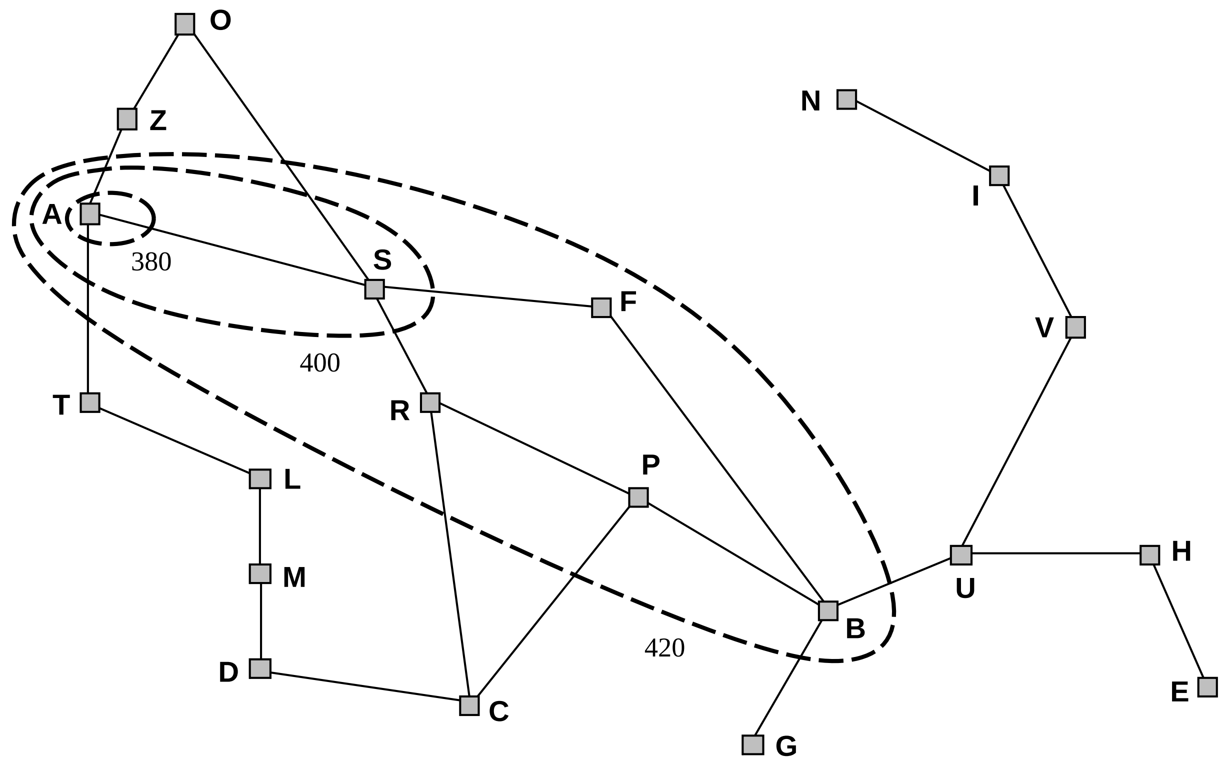
Optimální efektivita A^*

:: O A^* říkáme, že je **optimálně efektivní**. To znamená, že pro libovolnou heuristikou funkci neexistuje jiný optimální algoritmus, který by expandoval méně uzlů než A^*.

O A^* víme, že je optimální, úplný a optimálně efektivní. Nicméně to neznamená, že by byl vhodný na všechny problémy prohledávání. Bohužel paměťová náročnost zůstává nadále exponenciálně rostoucí (bylo dokázáno, že tomu tak je v případě, že $|h(n) - h^*(n)| > O(\log h^*(n))$).

:: Časová výpočetní náročnost není hlavním problémem A^*. Vzhledem k tom, že A^* musí udržovat všechny otevřené uzly v paměti, stává se, že často dojde na přetečení paměti dřív než vyprší čas.

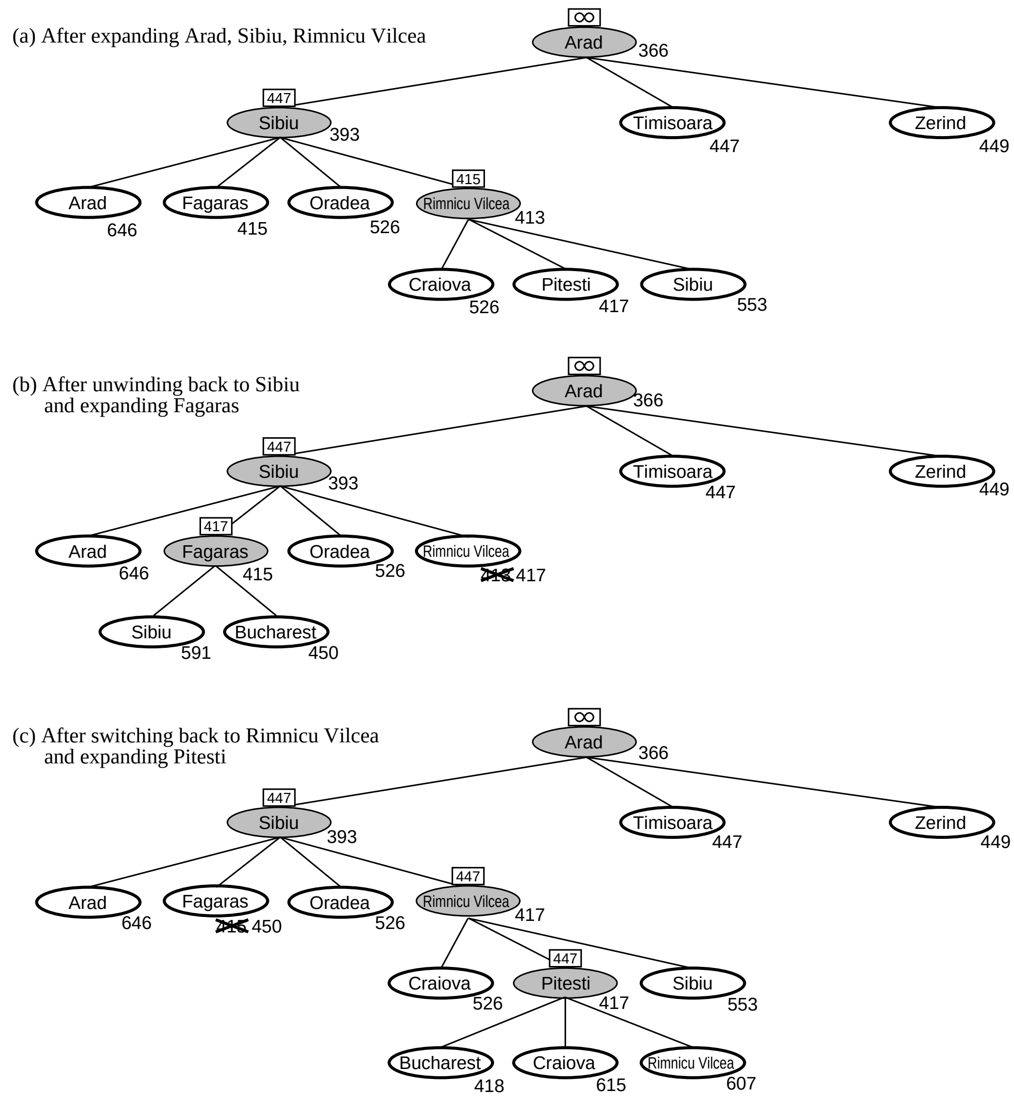
Optimální efektivita A^*



Varianty algoritmů zlepšující paměťové nároky

- **IDA*** – iterative deepening A^* algoritmus: Pracuje stejně jako iterativně prohlubující se prohledávání do hloubky (IDDFS), s tím, že zvětšující se limitní hodnota není hloubka, ale nejmenší hodnota f která je vyšší než f z předchozí úrovně.
- **RBFS** – Recursive best first search, rekurzivní IDA*. Omezuje hodnotu f druhou nejlepší hodnotou v dané úrovni.

Varianty algoritmů zlepšující paměťové nároky



Varianty algoritmů zlepšující paměťové nároky

- **IDA*** – iterative deepening A^* algoritmus: Pracuje stejně jako iterativně prohlubující se prohledávání do hloubky (IDDFS), s tím, že zvětšující se limitní hodnota není hloubka, ale nejmenší hodnota f která je vyšší než f z předchozí úrovně.
- **RBFS** – Recursive best first search, rekurzivní IDA*. Omezuje hodnotu f druhou nejlepší hodnotou v dané úrovni.

Zatímco IDA* má menší paměťové nároky, RFBS najde řešení rychleji, protože si udržuje větší `OpenList`.

Varianty algoritmů zlepšující paměťové nároky

- **IDA*** – iterative deepening A^* algoritmus: Pracuje stejně jako iterativně prohlubující se prohledávání do hloubky (IDDFS), s tím, že zvětšující se limitní hodnota není hloubka, ale nejmenší hodnota f která je vyšší než f z předchozí úrovně.
- **RBFS** – Recursive best first search, rekurzivní IDA*. Omezuje hodnotu f druhou nejlepší hodnotou v dané úrovni.

Zatímco IDA* má menší paměťové nároky, RFBS najde řešení rychleji, protože si udržuje větší `OpenList`.

- **MA*** – memory bounded A^* – využívá veškerou dostupnou paměť. Zjednodušený algorithms SMA* (simplified MA*) udržuje pouze určitý počet uzlů na open-seznamu. je-li plno vyhodí ten nejhorší uzel.

Varianty algoritmů zlepšující paměťové nároky

- **IDA*** – iterative deepening A^* algoritmus: Pracuje stejně jako iterativně prohlubující se prohledávání do hloubky (IDDFS), s tím, že zvětšující se limitní hodnota není hloubka, ale nejmenší hodnota f která je vyšší než f z předchozí úrovně.
- **RBFS** – Recursive best first search, rekurzivní IDA*. Omezuje hodnotu f druhou nejlepší hodnotou v dané úrovni.

Zatímco IDA* má menší paměťové nároky, RFBS najde řešení rychleji, protože si udržuje větší `OpenList`.

- **MA*** – memory bounded A^* – využívá veškerou dostupnou paměť. Zjednodušený algorithms SMA* (simplified MA*) udržuje pouze určitý počet uzlů na open-seznamu. je-li plno vyhodí ten nejhorší uzel.

IDA* a RFBS jsou optimální (tzn., nemohou minout nejlepší řešení), MA* a SMA* mohou minout optimum a uvíznout v lokálním extrému (je-li mez velikosti seznamu `OpenList` malá).