Pokročilá Umělá Inteligence

Prohledávání

Martin Macaš

Algoritmus A* - Definice

Zdroj:

Dechter, Pearl (1985): Generalized Best-First Strategies and The Optimality of A*

*Algorithm BF**

1. Put the start node s on a list called OPEN of unexpanded nodes.
2. If OPEN is empty, exit with failure; no solution exists.
3. Remove from OPEN a node n at which f is minimum (break ties arbitrarily, but in favor of a goal node), and place it on a list called CLOSED to be used for expanded nodes.
4. If n is a goal node, exit successfully with the solution obtained by tracing back the path along the pointers from n to s (pointers are assigned in Steps 5 and 6).
5. Expand node n, generating all its successors with pointers back to n.
6. For every successor n' of n:
   a. Calculate $f(n')$.
   b. If n' was neither in OPEN nor in CLOSED, then add it to OPEN. Assign the newly computed $f(n')$ to node n'.
   c. If n' already resided in OPEN or CLOSED, compare the newly computed $f(n')$ with that previously assigned to n'. If the new value is lower, substitute it for the old (n' now points back to n instead of to its predecessor). If the matching node n' resided in CLOSED, move it back to OPEN.
7. Go to Step 2.

Algoritmus A* - Definice

- Algoritmus Best First Search:

1. Ulož počáteční uzel **s** do **OPEN**
2. IF **OPEN** je prázdný, ukonči prohledávání neúspěchem
3. Odstraň z **OPEN** uzel **n** s nejmenší hodnotou **f** a ulož ho do **CLOSED**
4. IF **n** je cílový stav, ukonči prohledávání úspěšně
5. Expanduj uzel **n**
6. FOR každého jeho následníka **n'**:
   a. Spočti **f(n')**
   b. IF **n'** není v **OPEN** ani **CLOSED**,
      přidej do OPEN s ohodnocením a odkazem na rodiče
   c. IF **n'** je v **OPEN** nebo **CLOSED** s horším ohodnocením,
      přepiš ohodnocení i rodiče a případně přesuň z **CLOSED** do **OPEN**
7. GOTO 2

Algoritmus A* - Definice

- A* je Best First Search s funkcí **f** tvaru:

$$f(n)=g(n)+h(n)$$

- g(n) je cena právě ohodnocované cesty z počátečního stavu do stavu n
- h(n) je heuristický odhad ceny optimální cesty z n do cíle
- Přípustná heuristika:

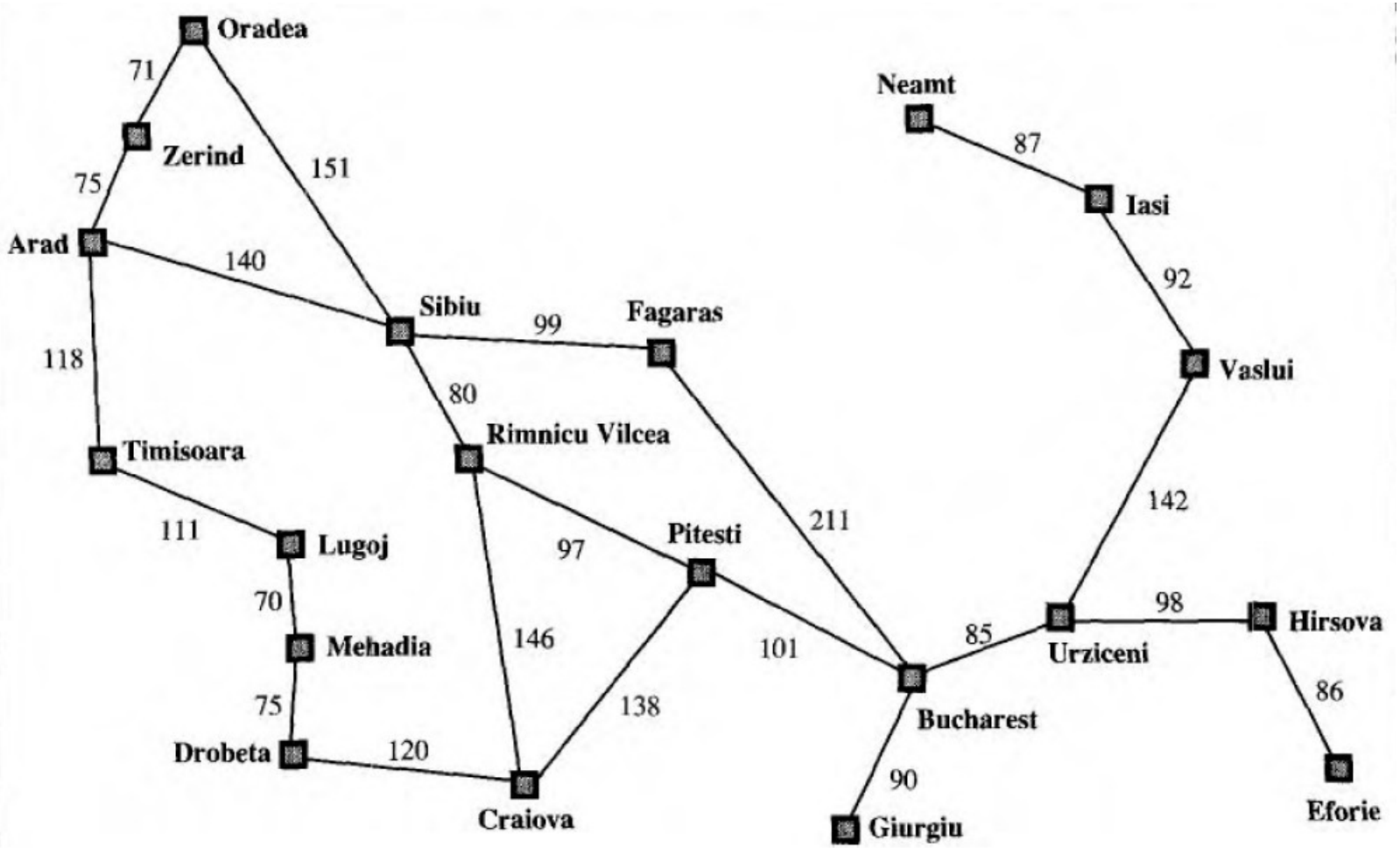
$$h(n)<h^*(n),$$

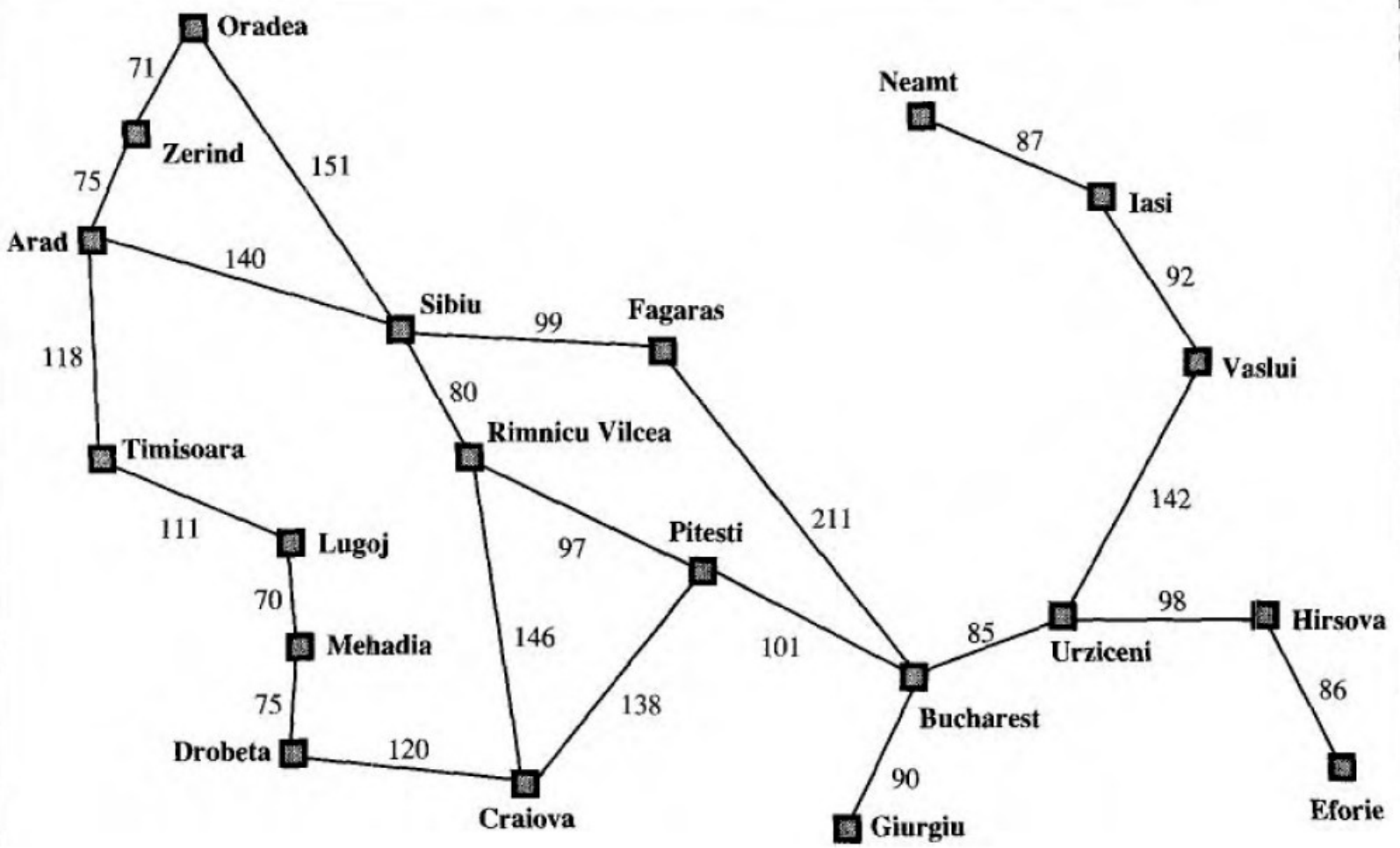
kde h*(n) je skutečná cena optimální cesty z n do cíle

Algoritmus A* - Definice

- **POZOR!!!!** V LITERATUŘE JSOU 2 MOŽNOSTI:
  - Někdy se toto uvádí jako postačující definice A*, potom takový A* nemusí být přípustný
  - Někdy se takový algoritmus nazývá algoritmus A a pouze pokud je splněna podmínka $h(n)<h^*(n)$, je algoritmus nazýván A* a je vždy přípustný

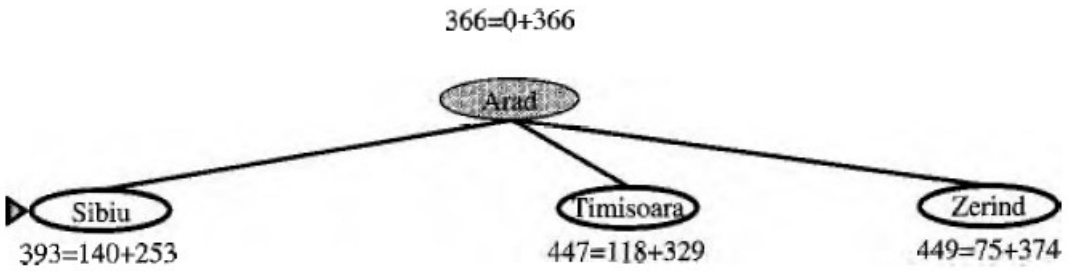
Algoritmus A* - Příklad



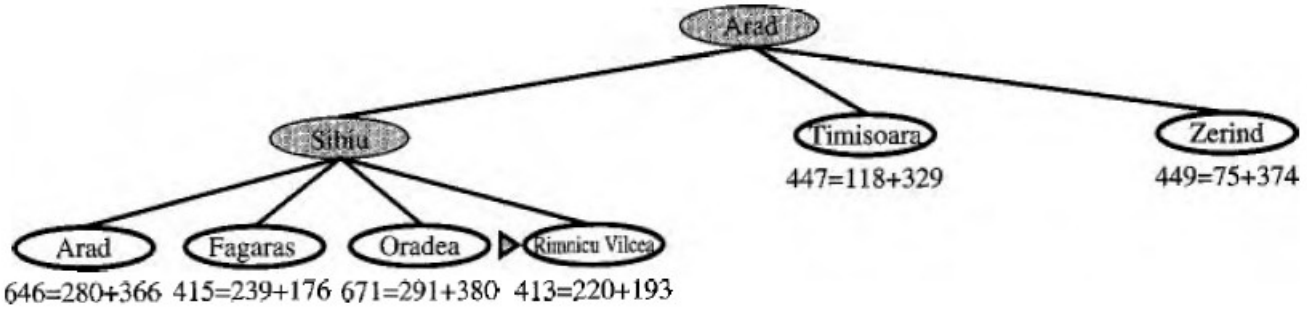


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

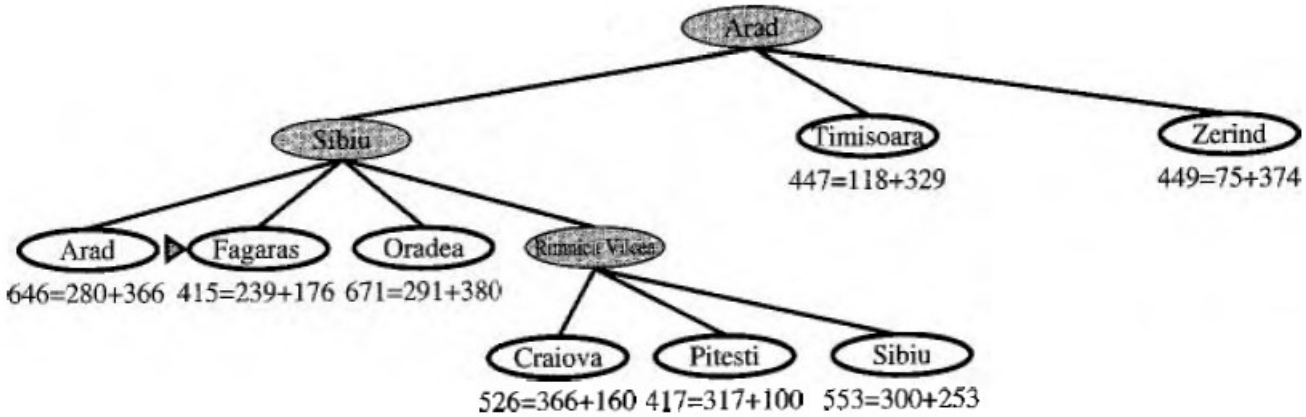
Algoritmus A* - Příklad



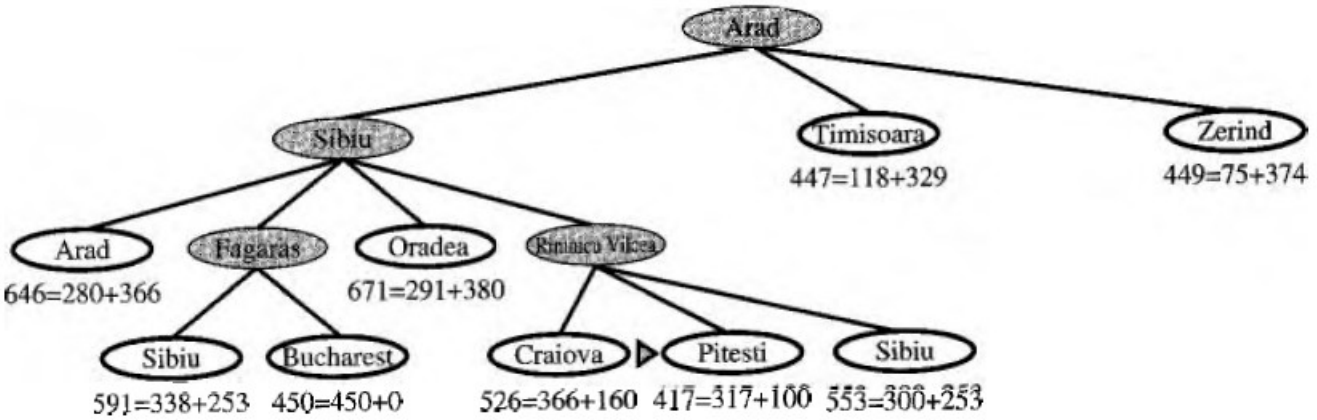
Algoritmus A* - Příklad



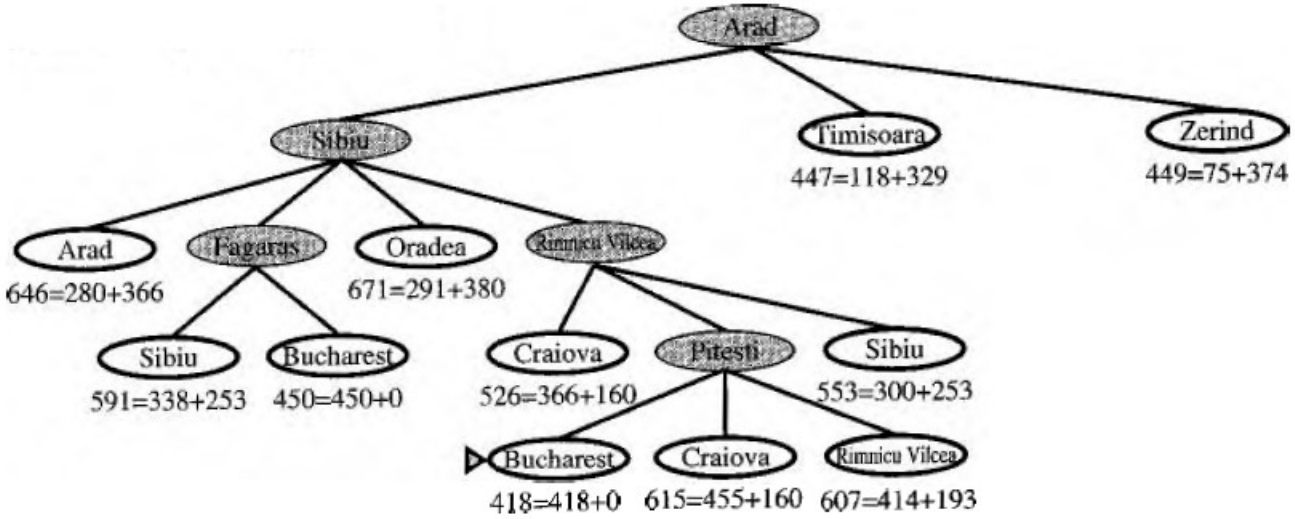
Algoritmus A* - Příklad



Algoritmus A* - Příklad



Algoritmus A* - Příklad



Algoritmus A* – Monotónní heuristika

Pro každý uzel **n** a jeho následníka **n'**, který vznikne použitím operátoru **o** platí:

$$h(n) - h(n') <= c(n,o,n'),$$

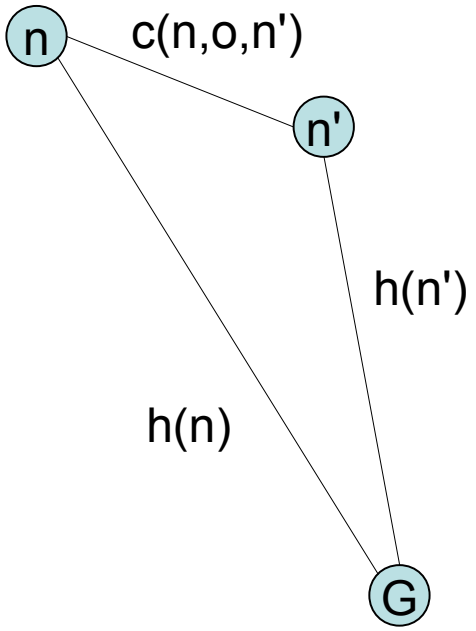
kde **c(n,o,n')** je cena použití operátoru **o** na stav **n.**

Lze přepsat jako

$$h(n) <= c(n,o,n') + h(n'),$$

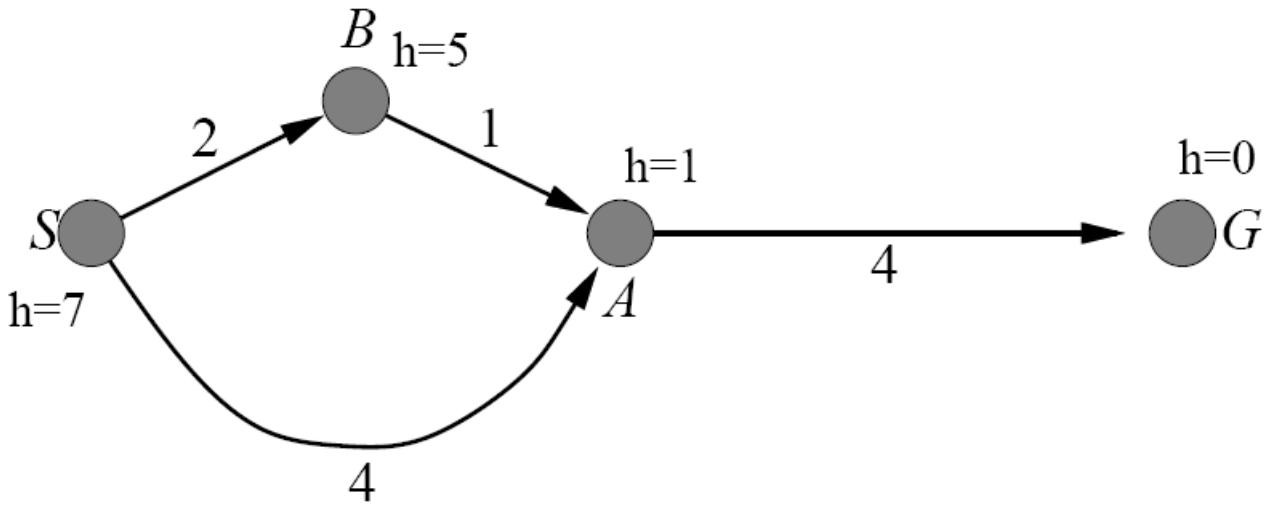
Důsledkem monotonnicity je:

Kdykoliv je nějaký uzel poprvé navštíven, je nalezená cesta z počátku do takového uzlu optimální!!!



Algoritmus A* - Nemonotónní heuristika

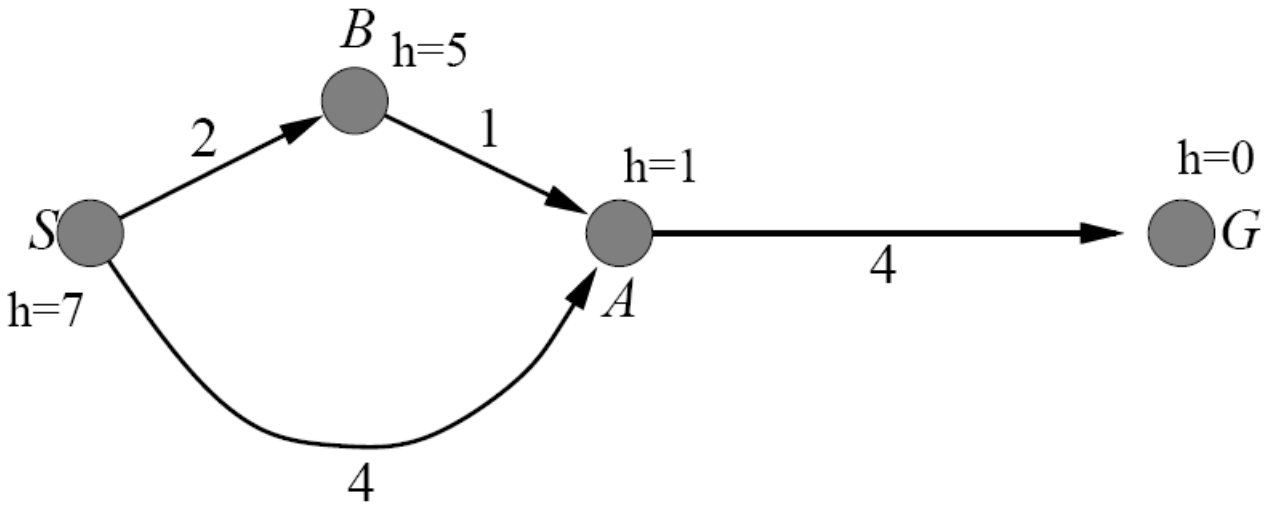
6c. IF **n'**je v **OPEN** nebo **CLOSED** s horším ohodnocením,
přepiš ohodnocení i rodiče a případně
přesuň z **CLOSED** do **OPEN**



Algoritmus A* - Nemonotónní heuristika

OPEN **CLOSED**

S-7+0=7-X

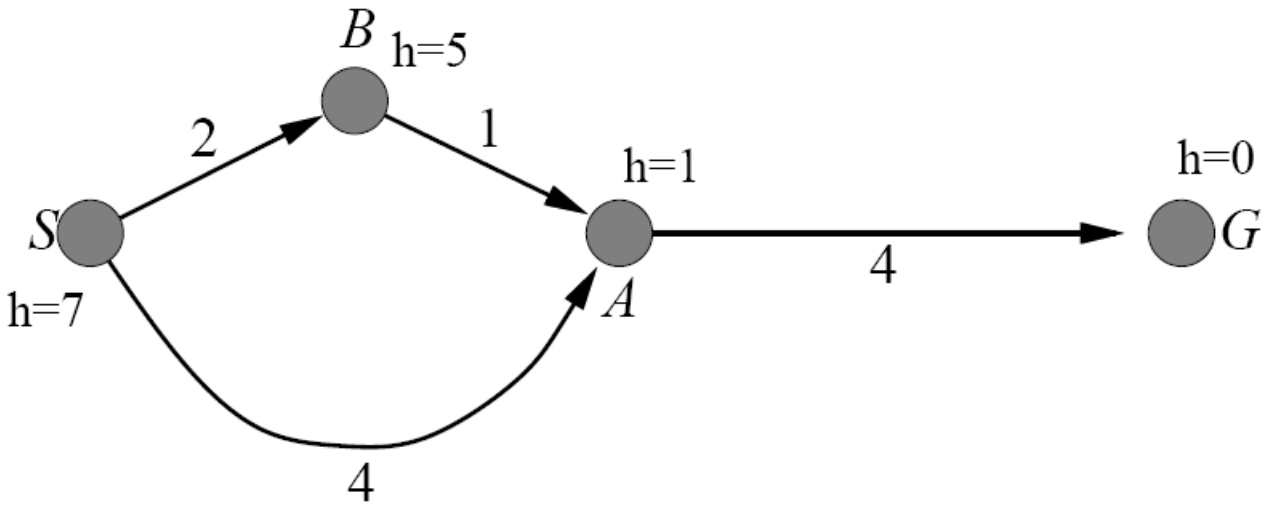


Algoritmus A* - Nemonotónní heuristika

OPEN

CLOSED

S-7+0=7-x



Algoritmus A* - Nemonotónní heuristika

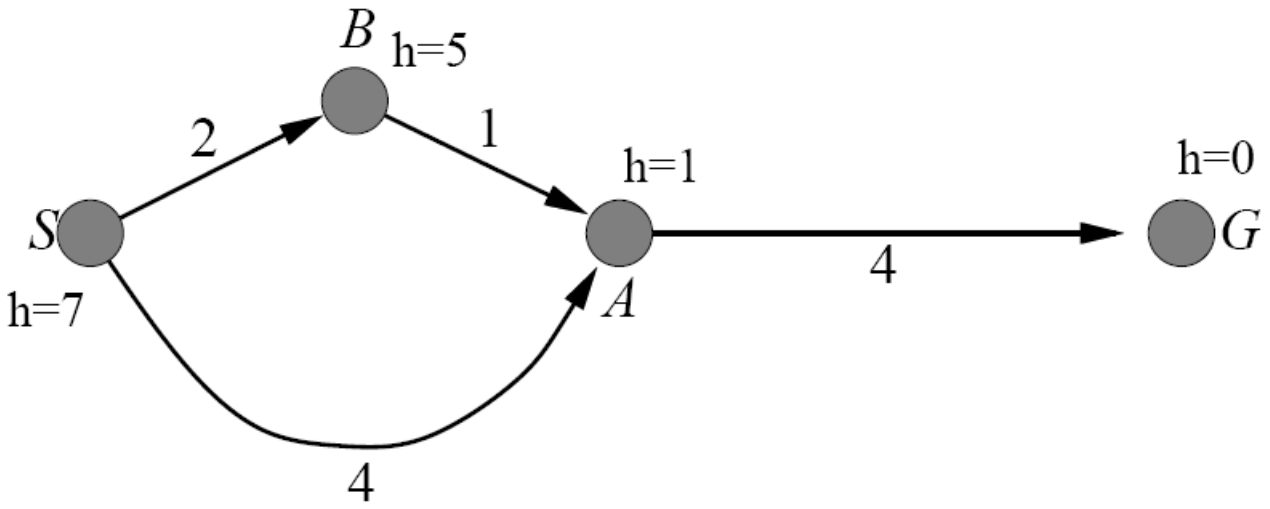
OPEN

A-4+1=5-S

B-2+5=7-S

CLOSED

S-7+0=7-X



Algoritmus A* - Nemonotónní heuristika

OPEN

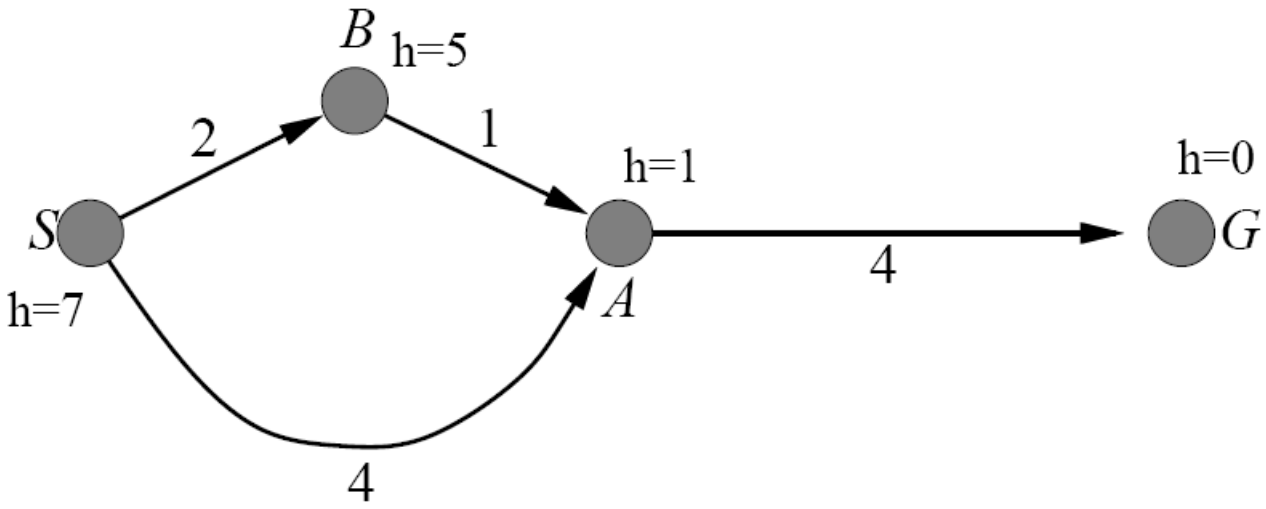
CLOSED

S-7+0=7-X

A-4+1=5-S

B-2+5=7-S

G-8+0=8-A



Algoritmus A* - Nemonotónní heuristika

OPEN

G-8+0=8-A

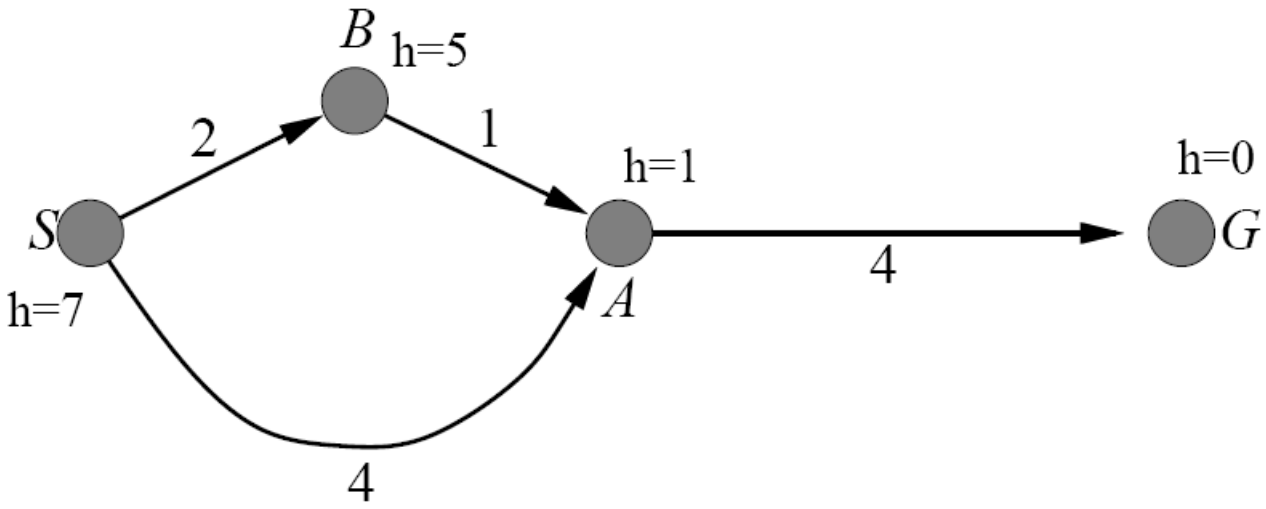
A-3+1=4-B

CLOSED

S-7+0=7-X

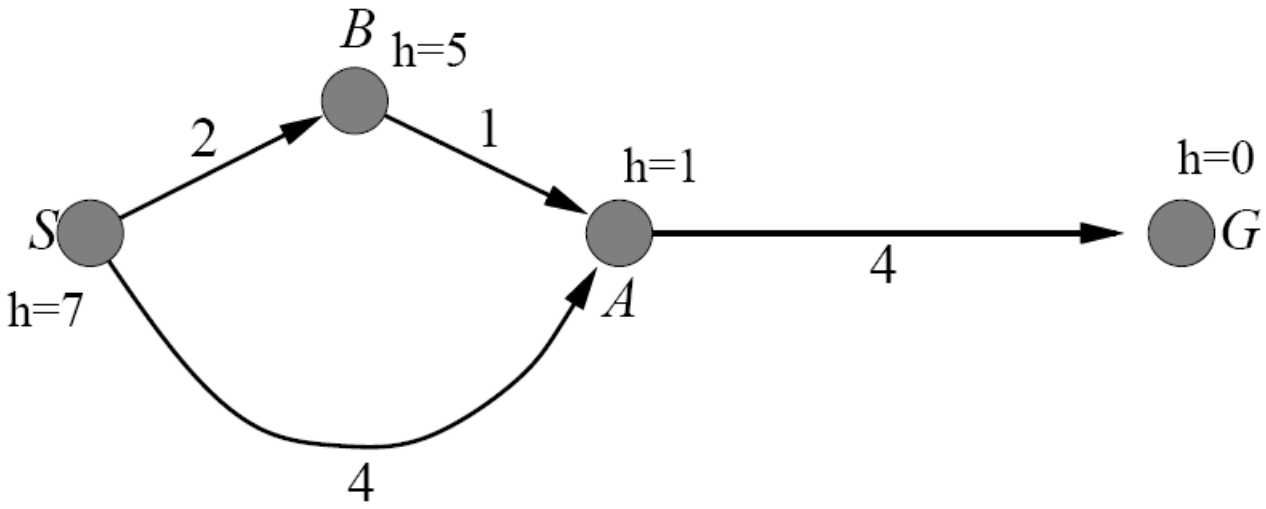
A-4+1=5-S

B-2+5=7-S



Algoritmus A* - Nemonotónní heuristika

OPEN	CLOSED
	S-7+0=7-X
	B-2+5=7-S
G-8+0=8-A	A-3+1=4-B
G-7+0=7-A	



Algoritmus A* - Nemonotónní heuristika

OPEN

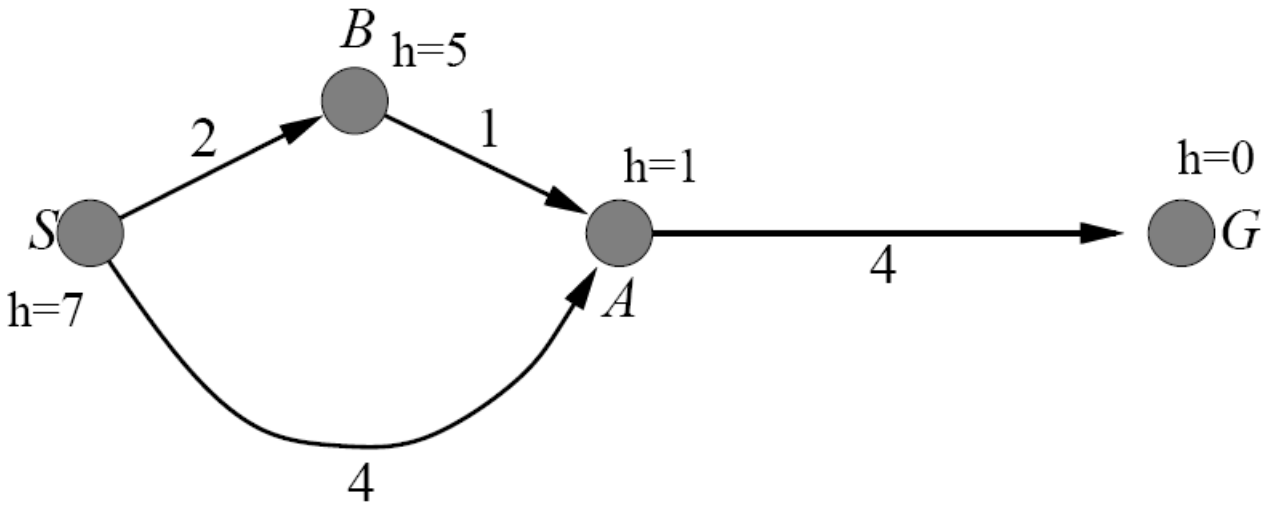
CLOSED

S-7+0=7-X

B-2+5=7-S

A-3+1=4-B

G-7+0=7-A



Algoritmus A* – Nemonotónní heuristika

- Algoritmus A* tak, jak jsem ho definovali, vždy najde optimální řešení (pokud existuje) pokud heuristika je přípustná

- Není třeba, aby byla heuristika monotónní, zajišťuje to bod 6c v našem pseudokódu

- Pokud je heuristika monotónní, pak není bod 6c třeba, protože kdykoliv je nějaký uzel poprvé navštíven, je nalezená cesta z počátku do takového uzlu optimální

Algoritmus A*

- Pokud je heuristika monotónní, pak je také přípustná!!!
- Přípustná heuristika nemusí být monotónní!!!

- ...ALE VĚTŠINOU JE...

Algoritmus A*
Shrnutí

- A* je úplný
- A* je optimální
- A* je nejen optimální, ale také tzv. optimálně efektivní - neexistuje jiný optimální algoritmus, u kterého by byla zajištěna expanze menšího počtu uzlů než expanduje A*

Algoritmus A*
Shrnutí

- Pokud prohledáváme pouze strom, nemusíme používat seznam CLOSED, nepotřebujeme bod 6c a staří přípustná heuristika (nemusí být monotónní)

Algoritmus A*
Nevýhody

- Počet expandovaných uzlů roste většinou exponenciálně s velikostí řešení
- Může vést na vysoké časové nároky
- Ale hlavně na vysokou paměťovou náročnost
- Algoritmus A* není vhodný pro řešení větších úloh

REKURZIVNÍ BEST FIRST SEARCH

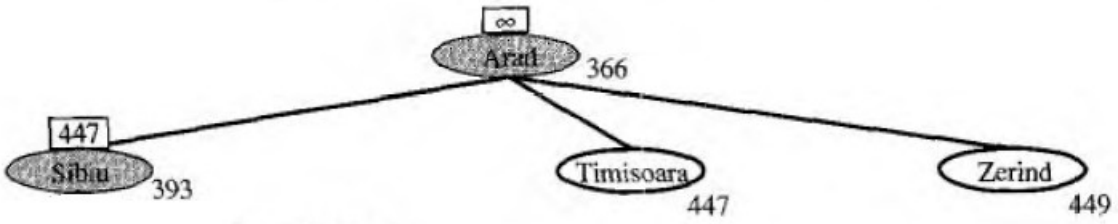
- Pro aktuální uzel si udržuje záznam o ceně nejlepšího jeho předchůdce (odhad ceny nejlepší alternativní cesty) jako práh
- Pokud cena aktuálního uzlu přesáhne tento práh, algoritmus se rekurzí vrátí k onomu nejlepšímu předchůdci
- Během této rekurze nahrazuje cenu každého uzlu cenou jeho nejlepšího následníka

function RECURSIVE-BEST-FIRST-SEARCH(*problem*) **returns a** solution, or failure
    RBFS(*problem*, MAKE-NODE(INITIAL-STATE[*problem*]), ∞)

function RBFS(*problem*, *node*, *f-limit*) **returns** a solution, or failure and a new *f*-cost limit
    if GOAL-TEST[*problem*](STATE[*node*]) **then return** *node*
    successors ← EXPAND(*node*, *problem*)
    if *successors* **is** empty **then return** *failure*, ∞
    for each *s* **in** *successors* **do**
        $f[s] \leftarrow \max(g(s) + h(s), f[node])$
    repeat
        best ← the lowest *f*-value node in *successors*
        iff *[best]* > *f-limit* **then return** *failure*, *f[best]*
        alternative ← the second-lowest *f*-value among *successors*
        result, f *[best]*← RBFS(*problem*, *best*, min(*f-limit*, *alternative*))
        if *result* ≠ *failure* **then return** *result*

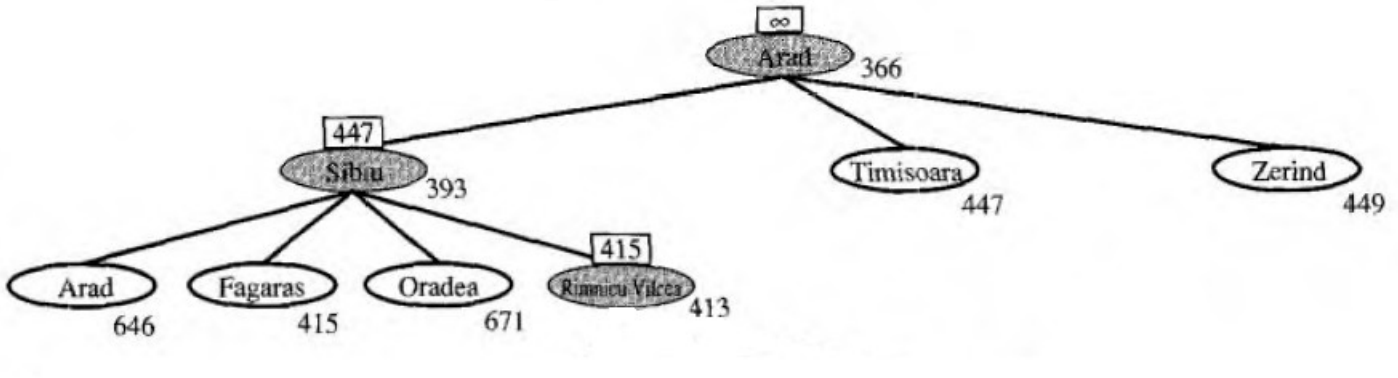
REKURZIVNÍ BEST FIRST SEARCH

- Pro aktuální uzel si udržuje záznam o ceně nejlepšího jeho předchůdce (odhad ceny nejlepší alternativní cesty) jako práh
- Pokud cena aktuálního uzlu přesáhne tento práh, algoritmus se rekurzí vrátí k onomu nejlepšímu předchůdci
- Během této rekurze nahrazuje cenu každého uzlu cenou jeho nejlepšího následníka



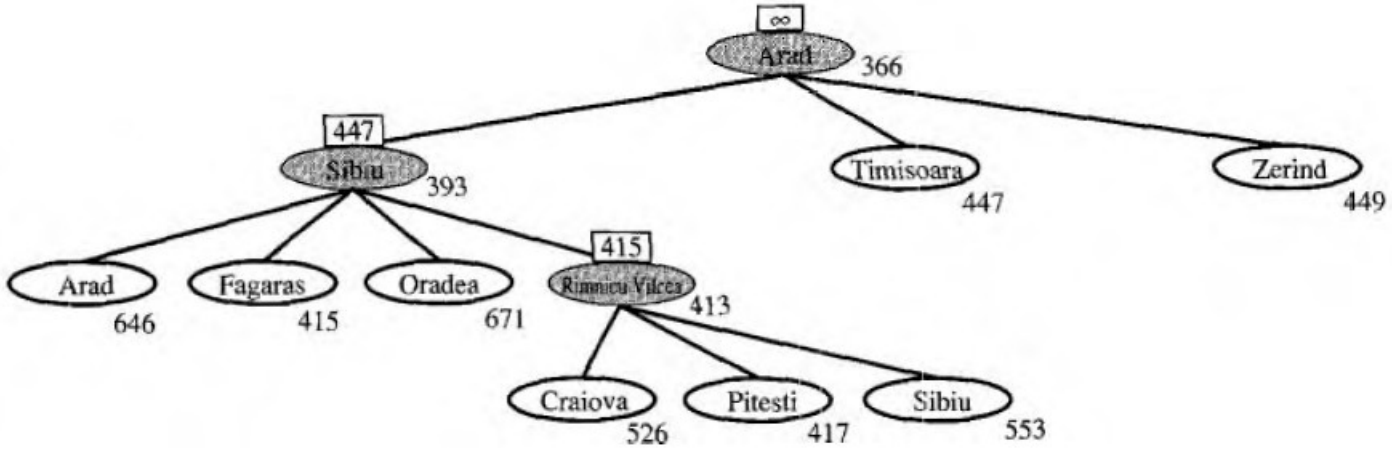
REKURZIVNÍ BEST FIRST SEARCH

- Pro aktuální uzel si udržuje záznam o ceně nejlepšího jeho předchůdce (odhad ceny nejlepší alternativní cesty) jako práh
- Pokud cena aktuálního uzlu přesáhne tento práh, algoritmus se rekurzí vrátí k onomu nejlepšímu předchůdci
- Během této rekurze nahrazuje cenu každého uzlu cenou jeho nejlepšího následníka



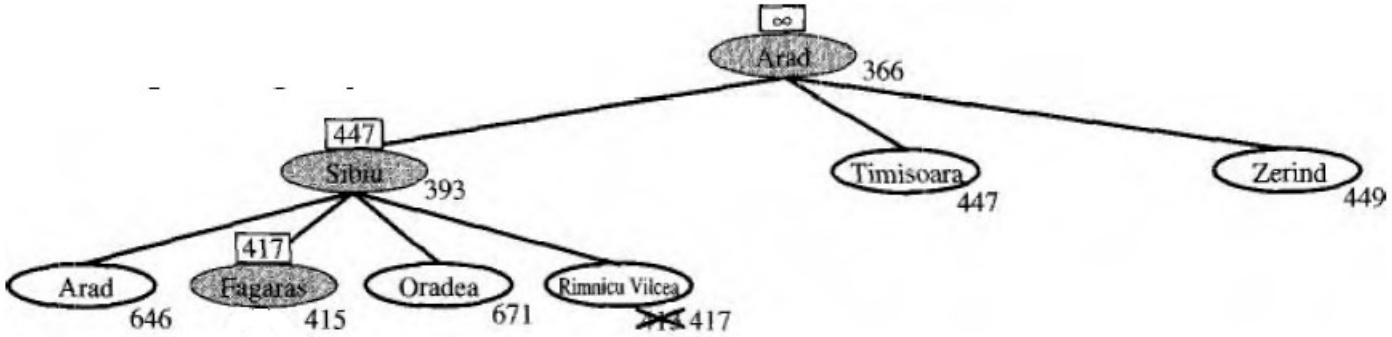
REKURZIVNÍ BEST FIRST SEARCH

- Pro aktuální uzel si udržuje záznam o ceně nejlepšího jeho předchůdce (odhad ceny nejlepší alternativní cesty) jako práh
- Pokud cena aktuálního uzlu přesáhne tento práh, algoritmus se rekurzí vrátí k onomu nejlepšímu předchůdci
- Během této rekurze nahrazuje cenu každého uzlu cenou jeho nejlepšího následníka



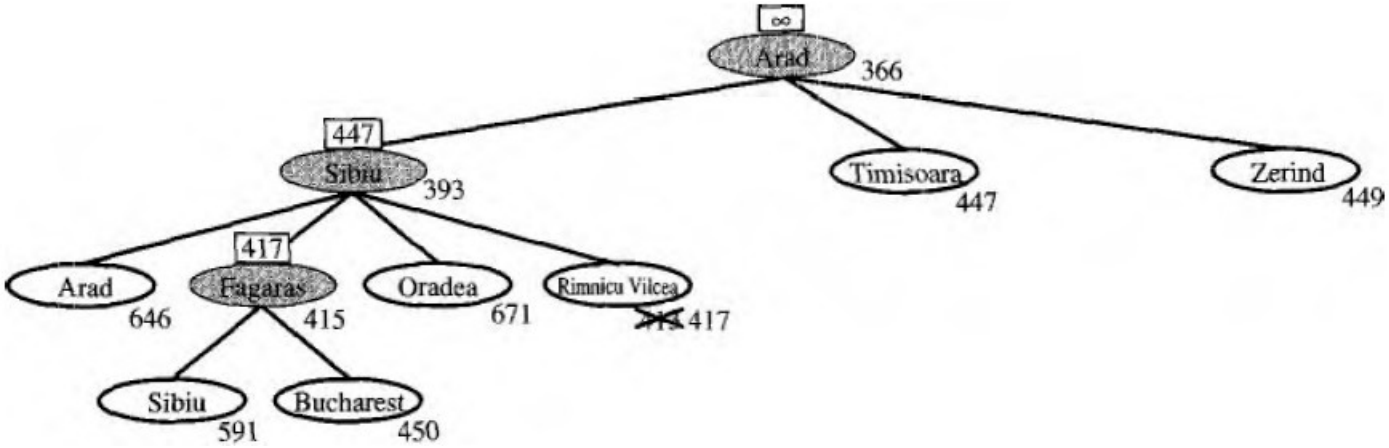
REKURZIVNÍ BEST FIRST SEARCH

- Pro aktuální uzel si udržuje záznam o ceně nejlepšího jeho předchůdce (odhad ceny nejlepší alternativní cesty) jako práh
- Pokud cena aktuálního uzlu přesáhne tento práh, algoritmus se rekurzí vrátí k onomu nejlepšímu předchůdci
- Během této rekurze nahrazuje cenu každého uzlu cenou jeho nejlepšího následníka



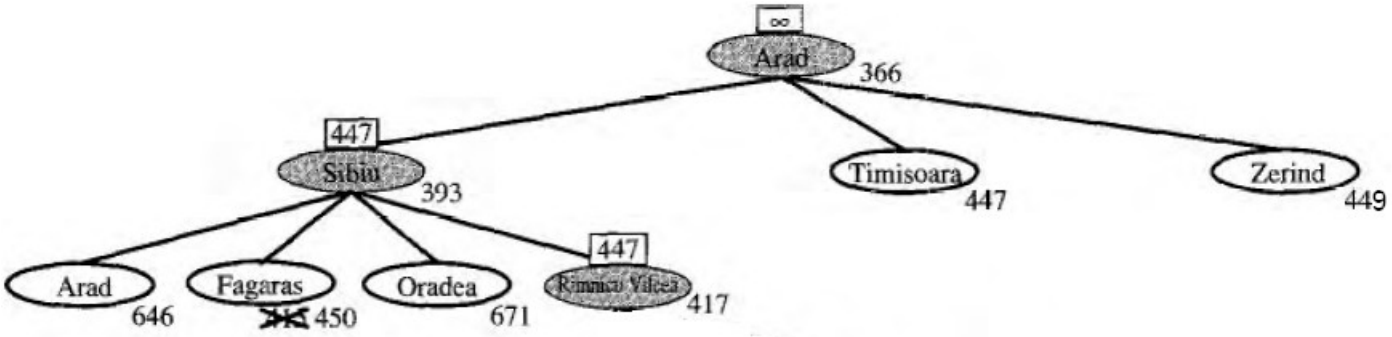
REKURZIVNÍ BEST FIRST SEARCH

- Pro aktuální uzel si udržuje záznam o ceně nejlepšího jeho předchůdce (odhad ceny nejlepší alternativní cesty) jako práh
- Pokud cena aktuálního uzlu přesáhne tento práh, algoritmus se rekurzí vrátí k onomu nejlepšímu předchůdci
- Během této rekurze nahrazuje cenu každého uzlu cenou jeho nejlepšího následníka



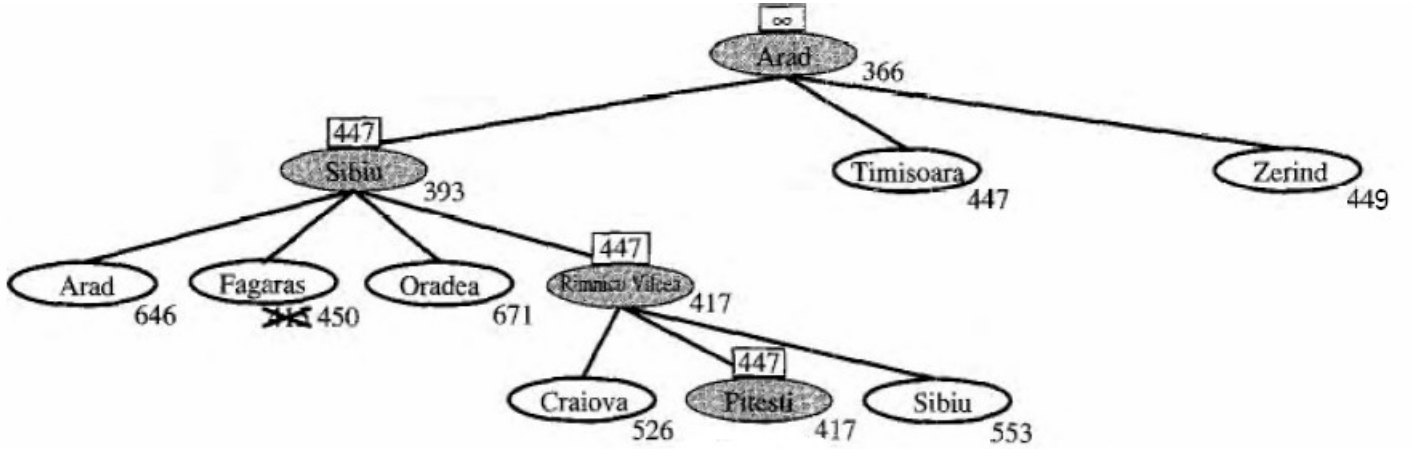
REKURZIVNÍ BEST FIRST SEARCH

- Pro aktuální uzel si udržuje záznam o ceně nejlepšího jeho předchůdce (odhad ceny nejlepší alternativní cesty) jako práh
- Pokud cena aktuálního uzlu přesáhne tento práh, algoritmus se rekurzí vrátí k onomu nejlepšímu předchůdci
- Během této rekurze nahrazuje cenu každého uzlu cenou jeho nejlepšího následníka



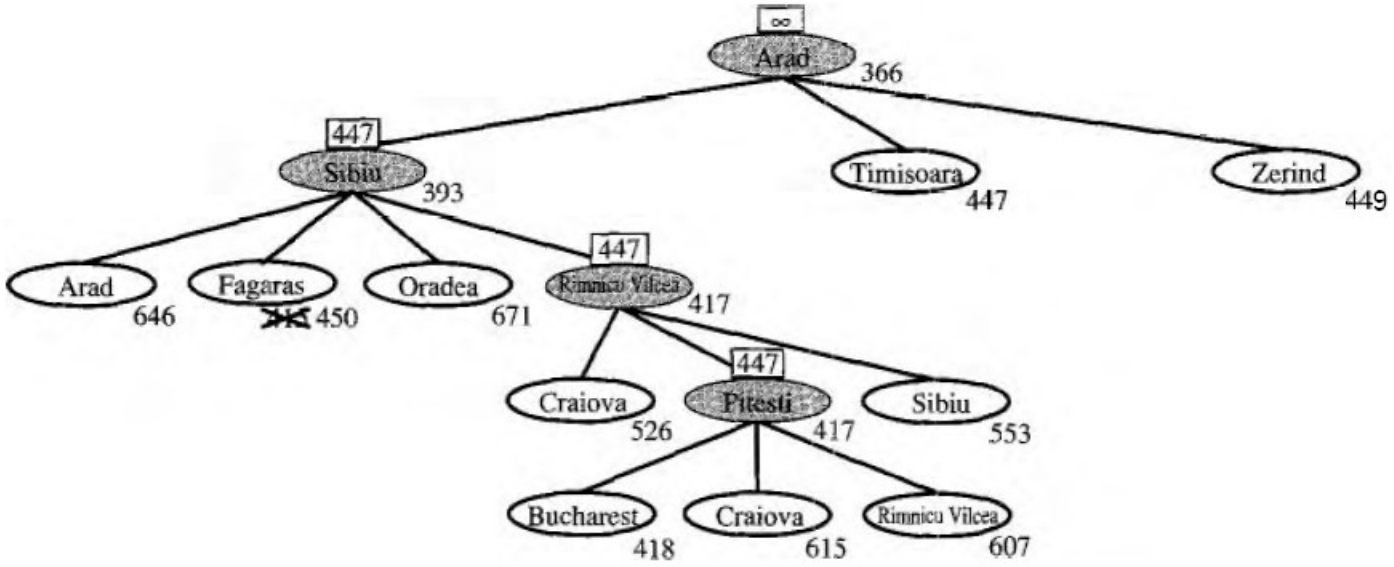
REKURZIVNÍ BEST FIRST SEARCH

- Pro aktuální uzel si udržuje záznam o ceně nejlepšího jeho předchůdce (odhad ceny nejlepší alternativní cesty) jako práh
- Pokud cena aktuálního uzlu přesáhne tento práh, algoritmus se rekurzí vrátí k onomu nejlepšímu předchůdci
- Během této rekurze nahrazuje cenu každého uzlu cenou jeho nejlepšího následníka



REKURZIVNÍ BEST FIRST SEARCH

- Pro aktuální uzel si udržuje záznam o ceně nejlepšího jeho předchůdce (odhad ceny nejlepší alternativní cesty) jako práh
- Pokud cena aktuálního uzlu přesáhne tento práh, algoritmus se rekurzí vrátí k onomu nejlepšímu předchůdci
- Během této rekurze nahrazuje cenu každého uzlu cenou jeho nejlepšího následníka



REKURZIVNÍ BEST FIRST SEARCH

- RBFS je optimální pokud je použita přípustná heuristika
- pokud stavový prostor neobsahuje cykly,paměťová náročnost roste lineárně s hloubkou nejhlubšího optimálního řešení
- časová složitost může být mnohem vyšší než u BFS
- paměťová náročnost může být také zbytečně malá

SIMPLIFIED MEMORY BOUNDED A*

- algoritmus, který využívá maximální povolenou paměť
- funguje jako A* do okamžiku, kdy je paměť plná
- pak je nutné "zapomínat" některé stavy
- algoritmus vždy zapomíná nejhorší stav (s nejvyšší cenou) a místo něho si pamatuje pouze nejnižší cenu v jeho podstromu
- je kompletní, pokud cesta k cílovému stavu není větší než kapacita paměti
- je použitelný i s přítomností cyklů

Doporučení

- uvědomit si, zda stavový prostor má formu grafu nebo pouze stromu
- zvolit vhodnou reprezentaci: např. N královen
  - REPREZENTACE 1 - každá královna může být umístěna do libovolného prázdného sloupce
  - REPREZENTACE 2 - každá královna může být umístěna do prvního prázdného sloupce zleva
- v REP1 může být každý stav dosažen N! cestami
- v REP2 může být každý stav dosažen pouze jedním způsobem

Učení heuristické funkce

- h může být problém zkonstruovat
- Někdy může být vhodné naučit nějaký systém odhadnout pro každý stav hodnotu heuristické funkce
- Může to být rychlejší
- Takový odhad může být informovanější (větší než klasické odhady)
- Může to fungovat s chybami
- Lze použít neuronové sítě, rozhodovací stromy, atd.

Pokročilá Umělá Inteligence

Lokální prohledávání

Martin Macaš

Prohledávání stavového prostoru

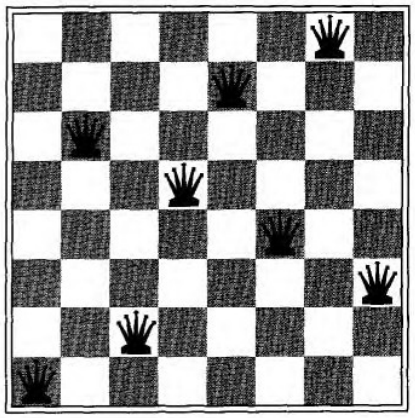
- systematické prohledávání
- navrací celou cestu od počátečního do koncového stavu
- často paměťově náročné

Lokální prohledávání

- systematické prohledávání
- neprohledává systematicky
- navrací celou cestu od počátečního do koncového stavu
- cesta nás nezajímá
- často paměťově náročné
- nízké paměťové nároky, často zabírá konstantní paměť

Lokální prohledávání

- neprohledává systematicky
- cesta nás nezajímá



- nízké paměťové nároky, často zabírá konstantní paměť
  - například jeden stav v paměti, který je postupně pozměňován

Lokální prohledávání

- Může najít **rozumné** řešení velice složitých problémů
  - Návrh logických obvodů
  - job-shop rozvrhování
  - automatické programy
  - optimalizace telekomunikačních sítí
  - ekonomika
  - data-mining
  - ...

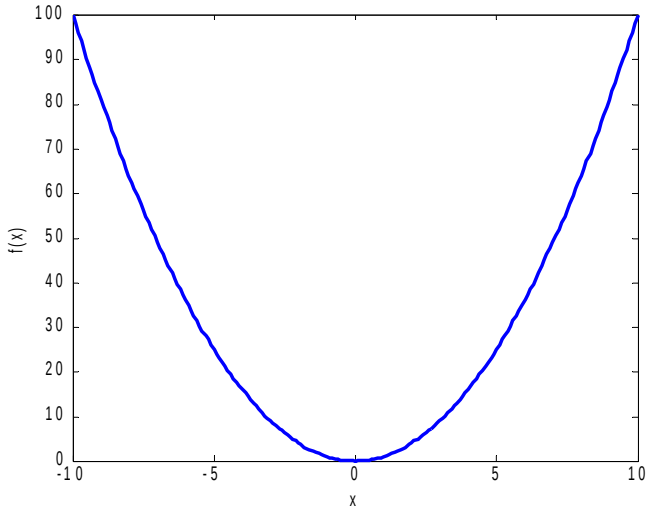
Optimalizace

- neexistuje žádný test cílového stavu
- neexistuje cena cesty
- každý stav **x** je ohodnotitelný fitness funkcí f(**x**)
- **PŘÍKLAD**

Hledáme minimum funkce $y=x^2$

Stavový prostor je R

Fitness funkce $f(x)=-x^2$ ohodnocuje stav x



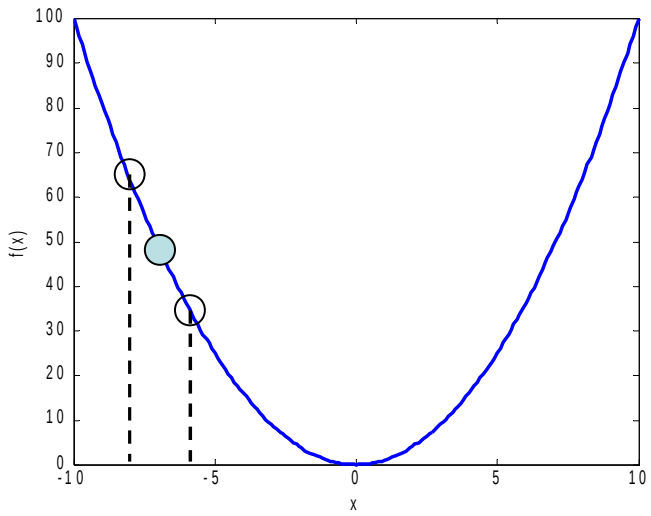
Lokální prohledávání

Lokální prohledávání

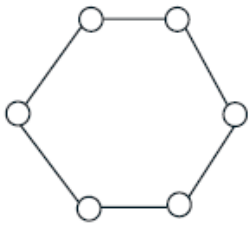
LOKÁLNÍ ZMĚNA

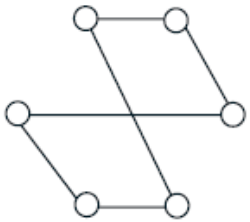
Lokální změna závisí na konkrétní úloze.

Jedná se vlastně o výběr nějakého souseda.



TSP





Lokální prohledávání

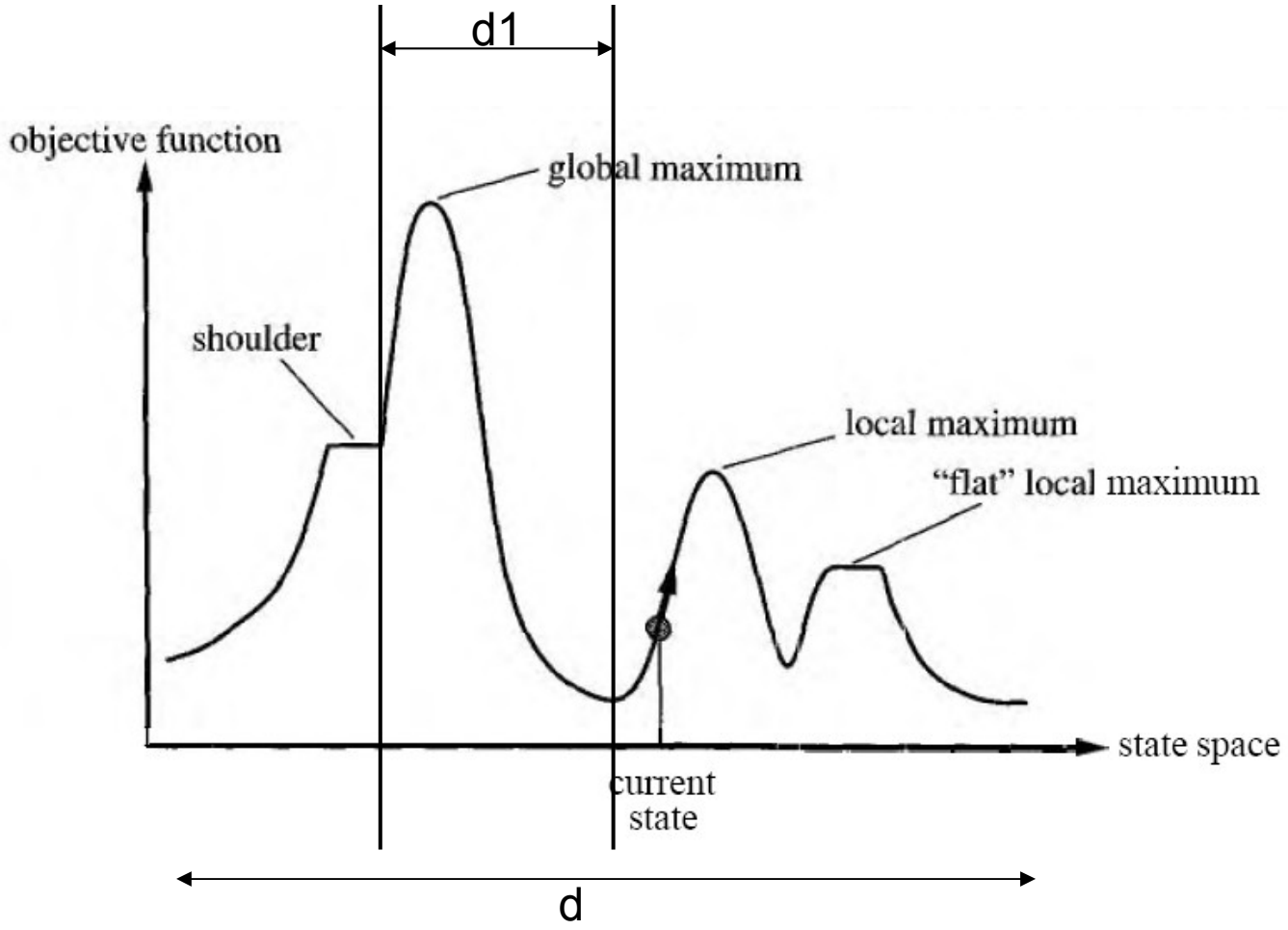
- Hill climbing

Algorithm 1 Hill-climbing

```
x=initialize randomly
loop
  s =a best successor of x (choose randomly in case of tie)
  if f(s) < f(x) then
    return x
  end if
  x = s
end loop
```

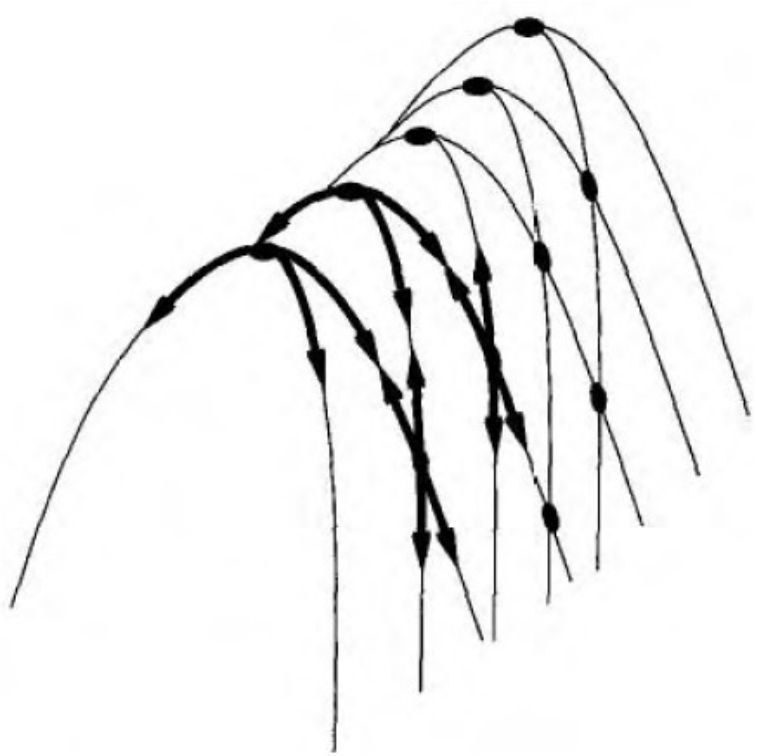
Lokální prohledávání

- Hill climbing algoritmus bude mít problémy
- Pravděpodobnost nalezení globálního optima je **p=d1/d**



Lokální prohledávání

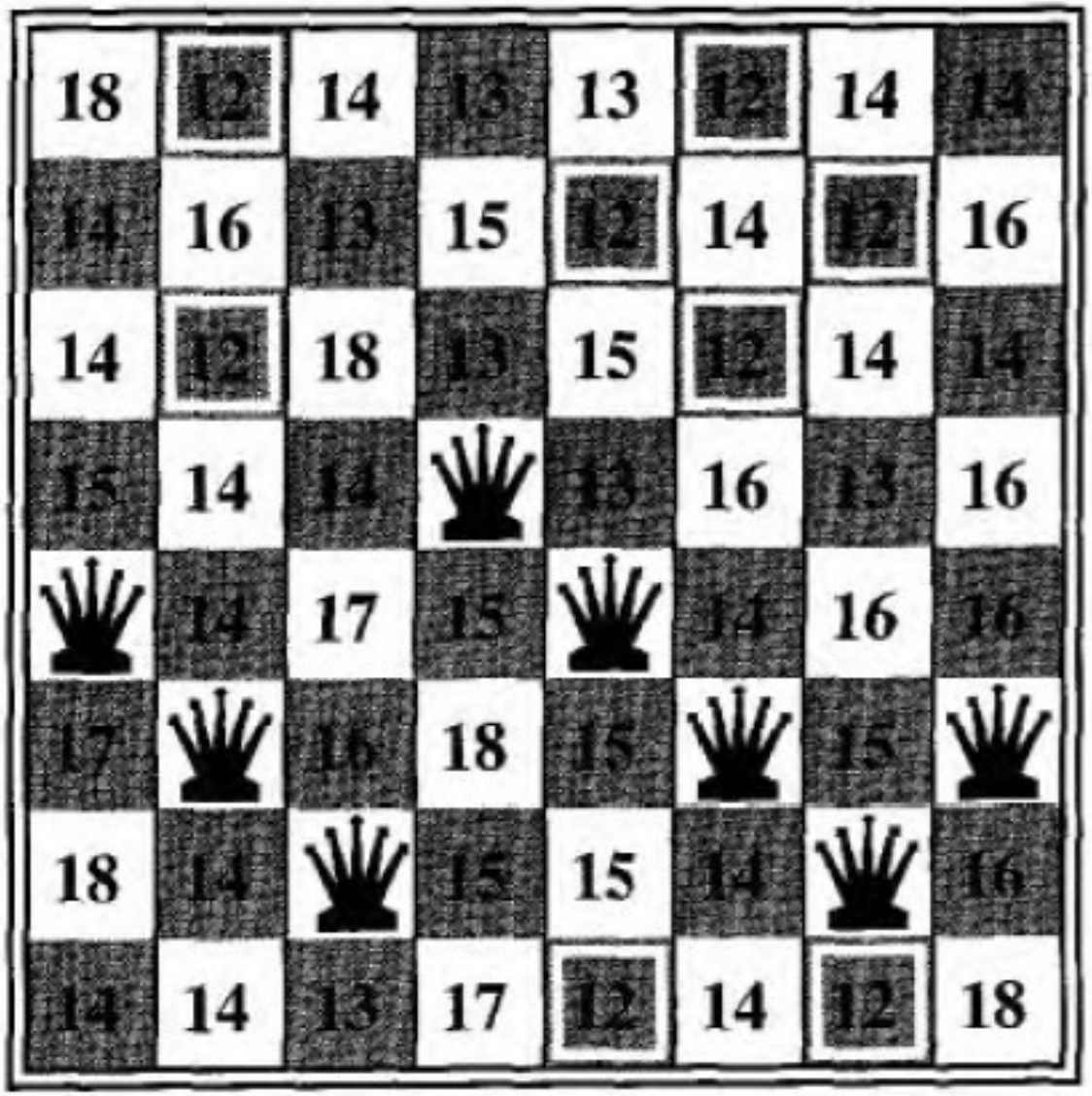
- Hill climbing algoritmus bude mít problémy
- Zvyšující se lokální maxima



Lokální prohledávání

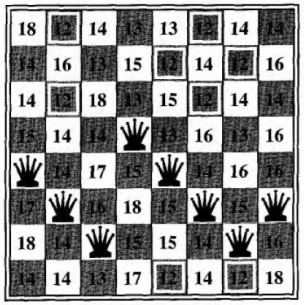
- N královen
  - stav: pozice každé královny v jejím sloupci
  - lokální změna: přenesení jedné královny na jiné políčko v jejím sloupci
  - počet následníků pro jeden stav je 8x7
  - Hledáme **minimum** funkce h=počet královen, které se atakují
  - Globální minimum je ohodnoceno h*=0

h=17, nejlepší následník: h=12

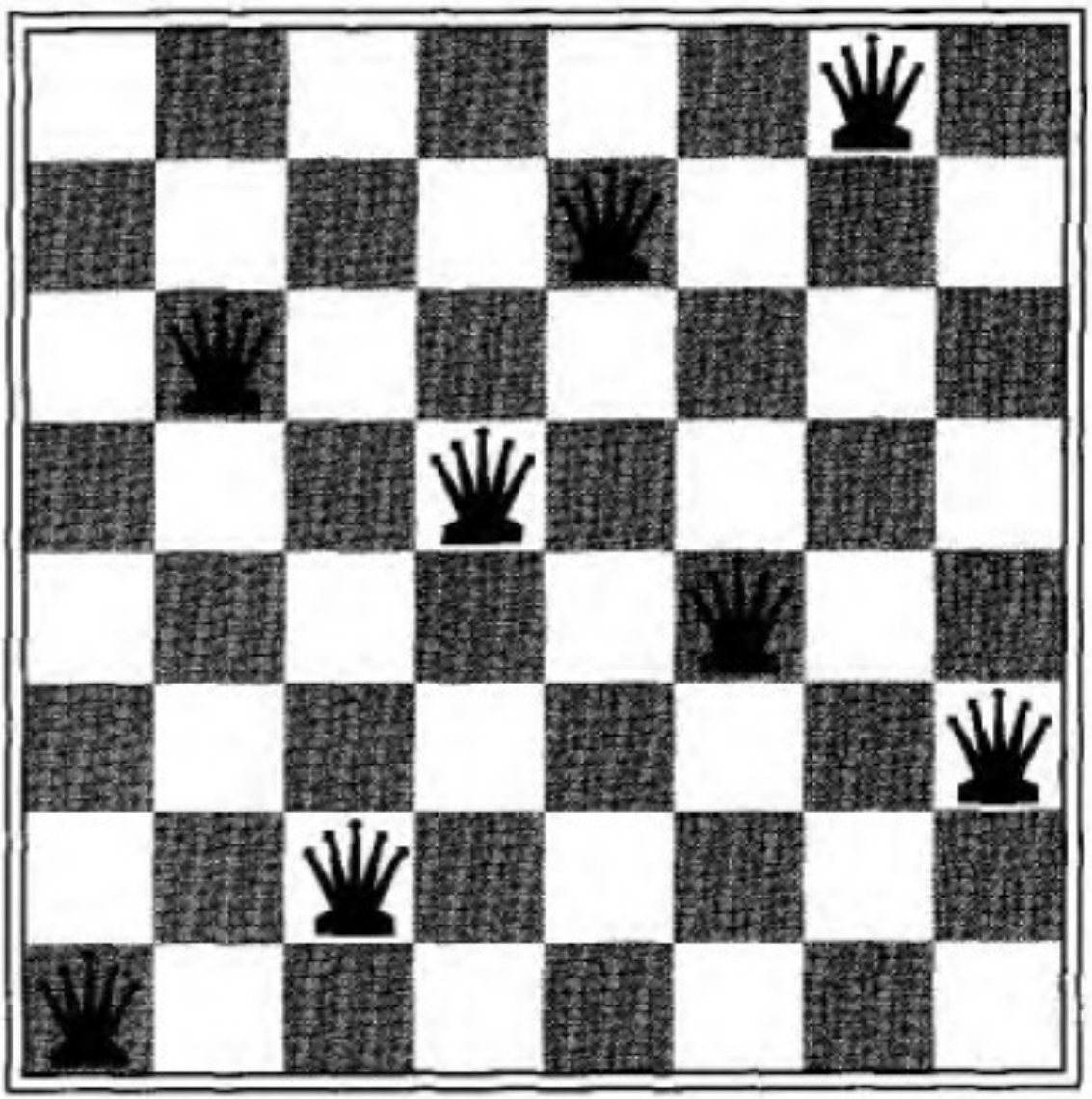


h=17, nejlepší následník: h=12

- 8 královen
  - stav: pozice každé královny v jejím sloupci
  - lokální změna: přenesení jedné královny na jiné políčko v jejím sloupci
  - počet následníků pro jeden stav je 8x7
  - Hledáme **minimum** funkce h=počet královen, které se atakují
  - Globální minimum je ohodnoceno h*=0
  - stačí pouze 5 kroků k dosažení stavu s h=1



h=1, nejlepší následník: h=1



h=1, nejlepší následník: h=1

Lokální prohledávání

- Hill climbing algoritmus bude mít problémy
- MODIFIKACE:
- STOCHASTIC HILL CLIMBING
  - vybírá náhodně lepšího souseda
  - pravděpodobnost výběru může záviset na ohodnocení
- FIRST CHOICE HILL CLIMBING
  - generuje sousedy náhodně a vybere prvního lepšího souseda
  - výhoda v případě velkého množství sousedů
- RANDOM RESTART HILL CLIMBING

"If at first you don't succeed, try, try again."

  - pokud pravděpodobnost úspěchu je **p**, očekávaný počet restartů nutných k nalezení globálního optima je **1/p**

Lokální prohledávání

- RANDOM RESTART HILL CLIMBING
  - pokud pravděpodobnost úspěchu je **p**, očekávaný počet restartů nutných k nalezení globálního optima je **1/p**

Lokální prohledávání

- Hill climbing tedy udržuje v paměti pouze jeden stav plus následníky
- Paměť často umožňuje více než to
- **BEAM SEARCH (Paprskové prohledávání)**
  - udržuje **k** stavů a všechny expanduje, z následníků vybere **k** nejlepších a nahradí rodiče
  - užitečná informace je rozdělena mezi **k** paralelních vláken

For example, if one state generates several good successors and the other $k-1$ states all generate bad successors, then the effect is that the first state says to the others, "Come over here, the grass is greener!" The algorithm quickly abandons unfruitful searches and moves its resources to where the most progress is being made.

Lokální prohledávání

- Hill climbing nedovoluje kroky, které by vedly k poklesu fitness
- Takové kroky ovšem mohou vést k úniku z lokálního optima
- SIMULATED ANNEALING (SIMULOVANÉ ŽÍHÁNÍ)

SIMULATED ANNEALING (SIMULOVANÉ ŽÍHÁNÍ)

- inspirace STATISTICKOU MECHANIKOU
- KONFIGURACE ATOMŮ
- Simulace pomocí Metropolisova algoritmu

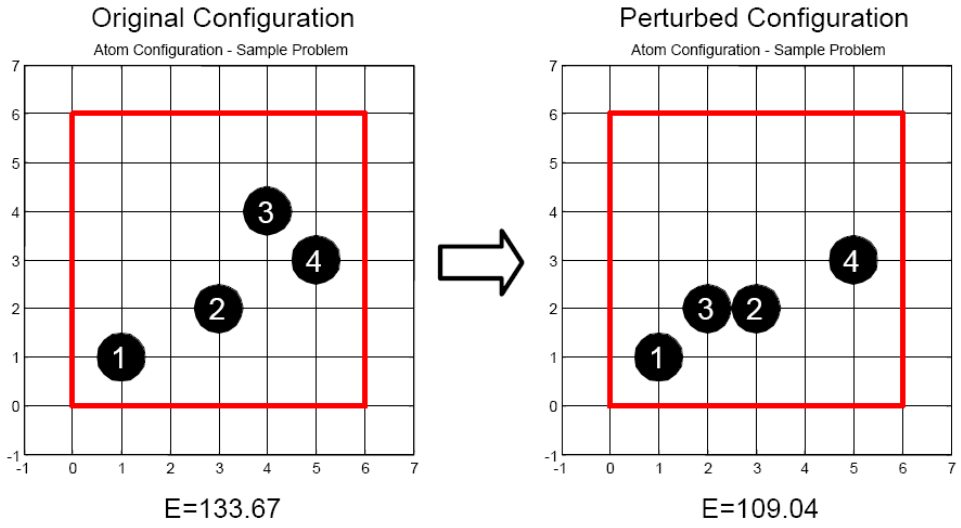
SIMULATED ANNEALING (SIMULOVANÉ ŽÍHÁNÍ)

- **STATISTICKÁ MECHANIKA**
  - zabývá se chováním systémů s velkým množstvím stupňů volnosti ve stavu tepelné rovnováhy a konečné teploty
- **KOMBINATORICKÁ OPTIMALIZACE**
  - hledání **minima** nějaké funkce, která závisí na mnoha proměnných

SIMULATED ANNEALING (SIMULOVANÉ ŽÍHÁNÍ)

Pokud nějaký tekutý materiál ochlazujeme moc rychle, pak tento materiál přejde do pevné fáze se suboptimální konfigurací.

Pokud bude ochlazování dostatečně pomalé, konfigurace atomů v materiálu bude mít minimální energii. Tento stav s minimální energií odpovídá globálnímu optimu.



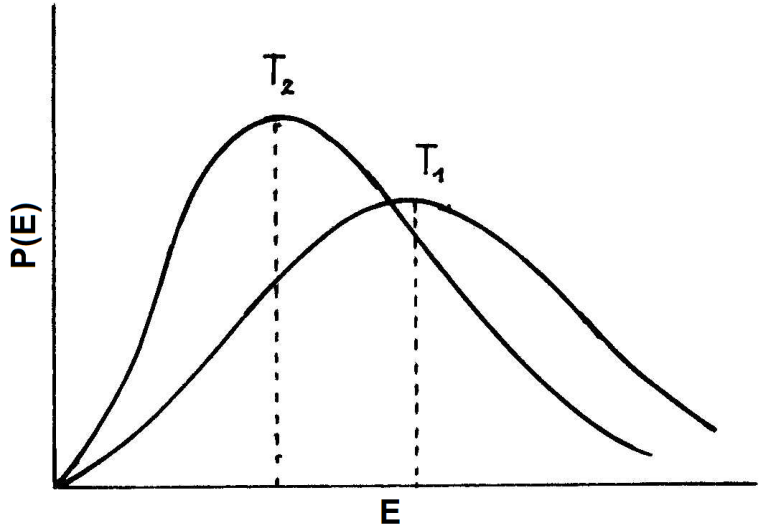
SIMULATED ANNEALING (SIMULOVANÉ ŽÍHÁNÍ)

Boltzmannovo pravděpodobnostní rozložení:

$$P(x) = \frac{1}{Z} e^{-\frac{E(x)}{kT}}$$

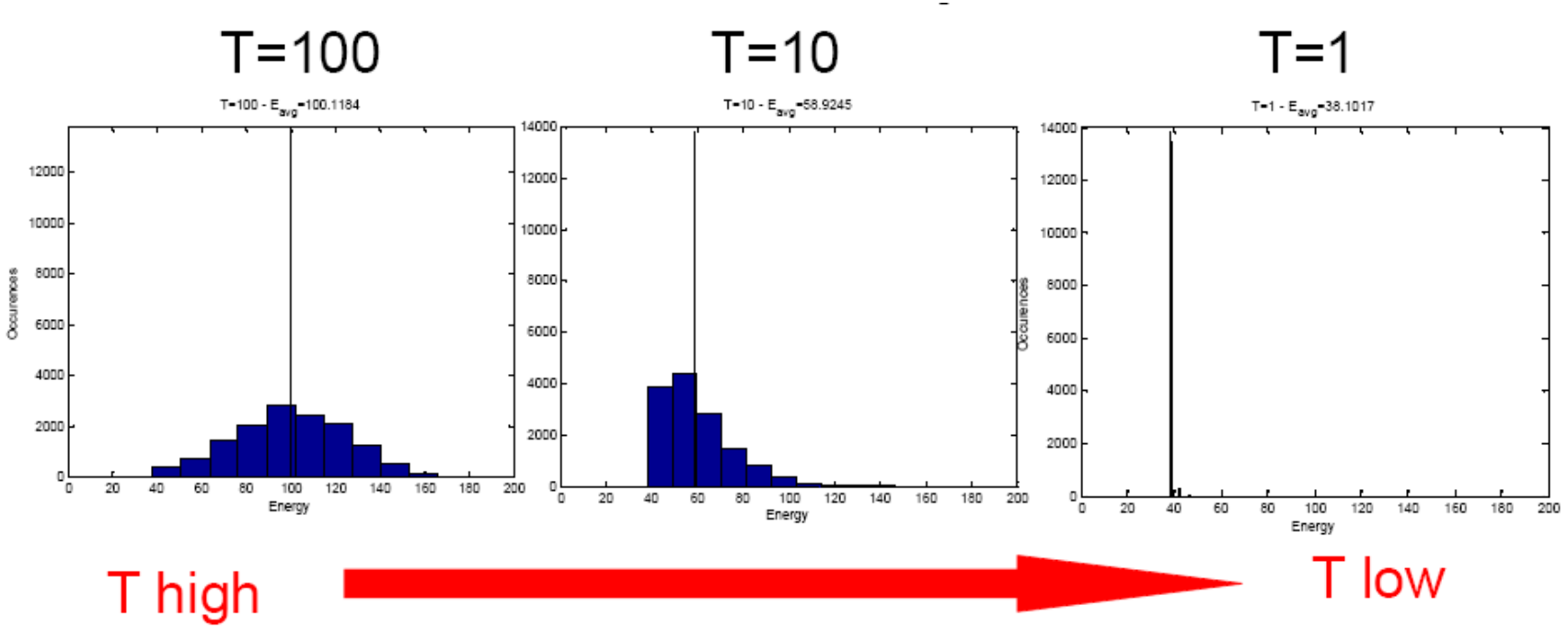
pravděpodobnost stavu **x**

E(x) je **energie** daného stavu
k Boltzmannova konstanta
T termodynamická teplota systému
Z normovací konstanta



SIMULATED ANNEALING (SIMULOVANÉ ŽÍHÁNÍ)

- Boltzmannův kolaps



SIMULATED ANNEALING (SIMULOVANÉ ŽÍHÁNÍ)

- MINIMALIZACE ZTRÁTOVÉ FUNKCE~MINIMALIZACE ENERGIE
- Využita simulace pomocí Metropolisova algoritmu
- V každé iteraci naruš lokálně aktuální stav
- Pokud je nový stav lepší nebo stejný ($f_{perturbed} <= f_{current}$),

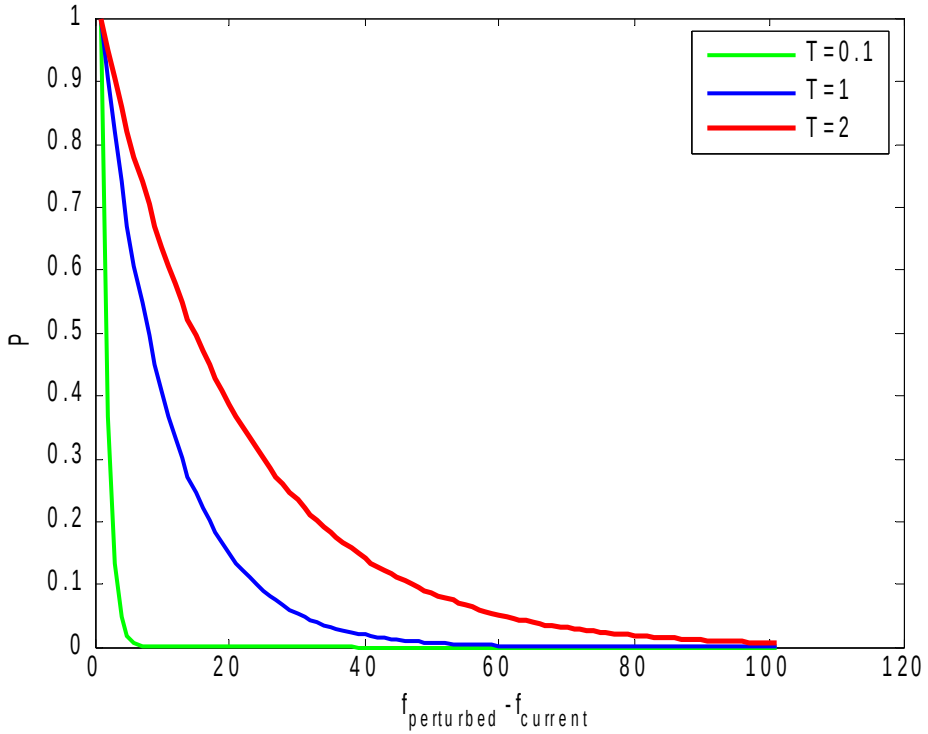
  nahraď jím aktuální stav s ***P=1***

- V opačném případě

  nahraď jím aktuální stav s

$$P = e^{-\frac{f(x_{perturbed}) - f(x_{current})}{T}}$$

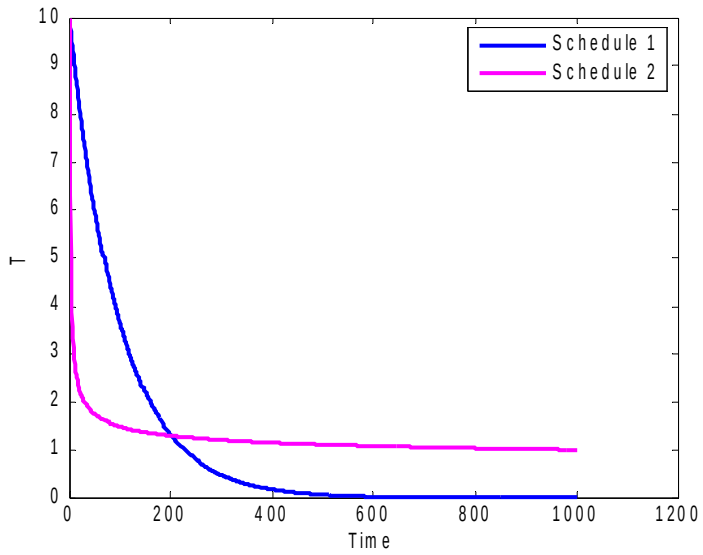
SIMULATED ANNEALING (SIMULOVANÉ ŽÍHÁNÍ)



$$P = e^{-\frac{f(x_{perturbed}) - f(x_{current})}{T}}$$

SIMULATED ANNEALING (SIMULOVANÉ ŽÍHÁNÍ)

- Teplota by se měla postupně snižovat
- Ochlazování může být
  - lineární
  - schodovité
  - exponenciální
  - ...



- $T(t) = R/\log(t)$. Convergence guaranteed, but known to be slow empirically.

- Exponential Schedule: $T(t) = T(0)a^n$ with $a < 1$ and very close to 1 (a=0.95 or 0.99) commonly used.