Metody informovaného hledání

Podstatou metod informovaného vyhledávání je snaha zabránit hledacím algoritmům, aby zabloudily.

Neinformované hledání, diskutované dříve, je schopno najít řešení problémů systematickým generováním nových stavů a testováním, zda splňují cílové podmínky. Potíž je v tom, že ve většině případů jsou vysoce neefektivní (čas, prostor, optimalita, ...).

Informované hledání je založeno na strategii, která využívá *znalost* specifickou pro daný problém, a nalezení cíle je mnohem efektivnější, včetně lepší možnosti dosáhnout optimálního řešení.

Hledání prvního nejlepšího

Doposud bylo obecně možné, po formulaci problému v termínech stavů a operátorů, nějakým způsobem určitou znalost aplikovat pro hledání cíle. Avšak pokud je problém tzv. dobře definovaný, volby jsou omezené. Např. při použití algoritmu `General-Search` se dá znalost použít na jediném místě—ve funkci zařazování do fronty, kde se určuje, který uzel má být expandován jako příští.

Obvykle se znalost toho, jak určit uzel, stanovuje **vyhodnocovací funkcí** (evaluation function), která vrací číslo určující míru (nebo nedostatek) potřeby expandovat uzel.

Pokud jsou uzly uspořádány tak, že ty s nejlepším ohodnocením jsou expandovány jako první, nazývá se tato strategie jako **hledání prvního nejlepšího** (best-first search).

Ilustrace možné implementace této funkce je na následujícím obrázku.

```
function Best-First-Search(problem, Eval-Fn) returns sekvence
    řešení

    inputs: problem // problém
            Eval-Fn // vyhodnocovací funkce

    Queuing-Fn ← funkce seřazující uzly pomocí Eval-Fn
    return General-Search(problem, Queuing-Fn)
```

Pozn.: Název "vyhledej první nejlepší" ve skutečnosti znamená jen to, že získáme uzel, ketrý se *zdá* být nejlepším (jinak by cesta k řešení byla snadná přímočará, což obecně neexistuje). Vzhledem k tomu, že funkce hledání nejlepšího uzlu není vševědoucí, může být hledání samozřejmě svedeno z cesty. Správný název by tedy měl být spíše *hledání zdánlivě prvního nejlepšího*.

Obdobně, jako existuje skupina algoritmů `General-Search` pro různé funkce zařazování do fronty, je k dispozici skupina `Best-First-Search` algoritmů s různými vyhodnocovacími funkcemi.

Tyto `Best-First-Search` algoritmy se snaží najít *nenákladná* řešení, typicky používají nějakou odhadovací míru pro cenu řešení a snaží se ji minimalizovat.

Jednou z již ukázaných možností byla cena cesty *g* k rozhodnutí, kterou cestu prodloužit (viz předchozí diskuse k tzv. dobře definovaným problémům a řešením). Míra *g* však *přímo nesměřuje k cíli*.

Aby bylo hledání přímo zaměřeno na cíl, musí použitá míra v sobě zahrnovat nějaký *odhad ceny cesty z nějakého stavu do nejbližšího cílového stavu*.

Lze použít nejméně dva základní přístupy k uvedenému řešení: pokus expandovat *uzel nejbližší k cíli* a pokus expandovat *uzel na nejlevnější cestě k řešení*.

Lačné vyhledávání minimalizující odhadovanou cenu dosažení cíle

Jednou z nejjednodušších strategií hledání prvního nejlepšího je minimalizace odhadované ceny dosažení cíle. Znamená to, že vždy je prvně expandován uzel, který se zdá být nejblíže cíli. Pro většinu (reálných) problémů lze náklady na dosažení cíle z nějakého okamžitého stavu jen *odhadnout*, nikoliv určit přesně.

Funkce, která počítá takové odhady nákladů, se nazývá **heuristická funkce**, obyčejně označovaná jako *h:*

> $h(n)$ = odhadnutá cena nejlevnější cesty ze stavu v uzlu *n* do stavu cílového.

Hledání prvního nejlepšího, které používá *h* k výběru dalšího expandovaného uzlu, se nazývá **lačné hledání** (greedy search).

Symbolický kód pro lačné hledání s danou funkcí *h:*

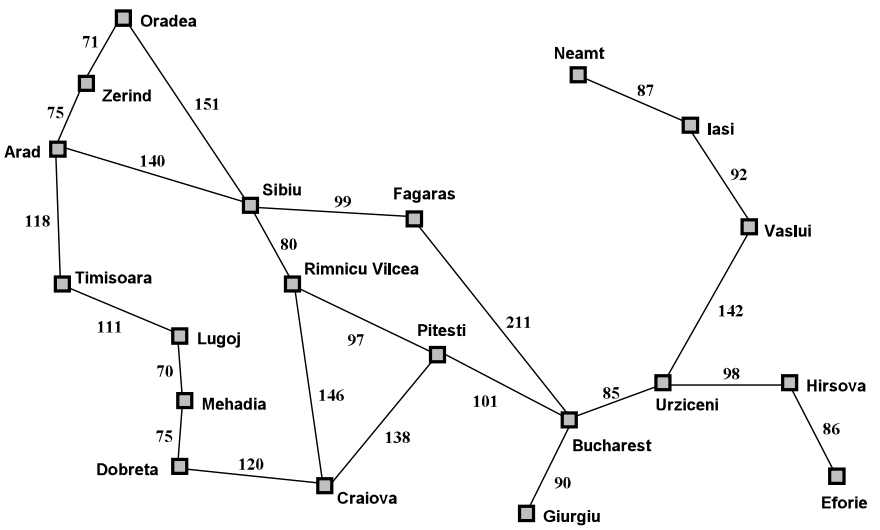
```
function Greedy-Search(problem) returns řešení nebo neúspěch

     return Best-First-Search(problem, h)
```

Formálně vzato, *h* může být jakoukoliv funkcí. Jediným požadavkem je, aby $h(n) = 0$ jestliže *n* je cíl.

Heuristické funkce jsou vždy specifické pro daný problém, tj. aplikačně závislé. Pro příklad uvažme opět cestování z města **Arad** do **Bucharest**. Mapka je stejná jako dříve, avšak přídavná informace je zde k dispozici ve formě kilometrové vzdálenosti tras mezi jednotlivými městy.

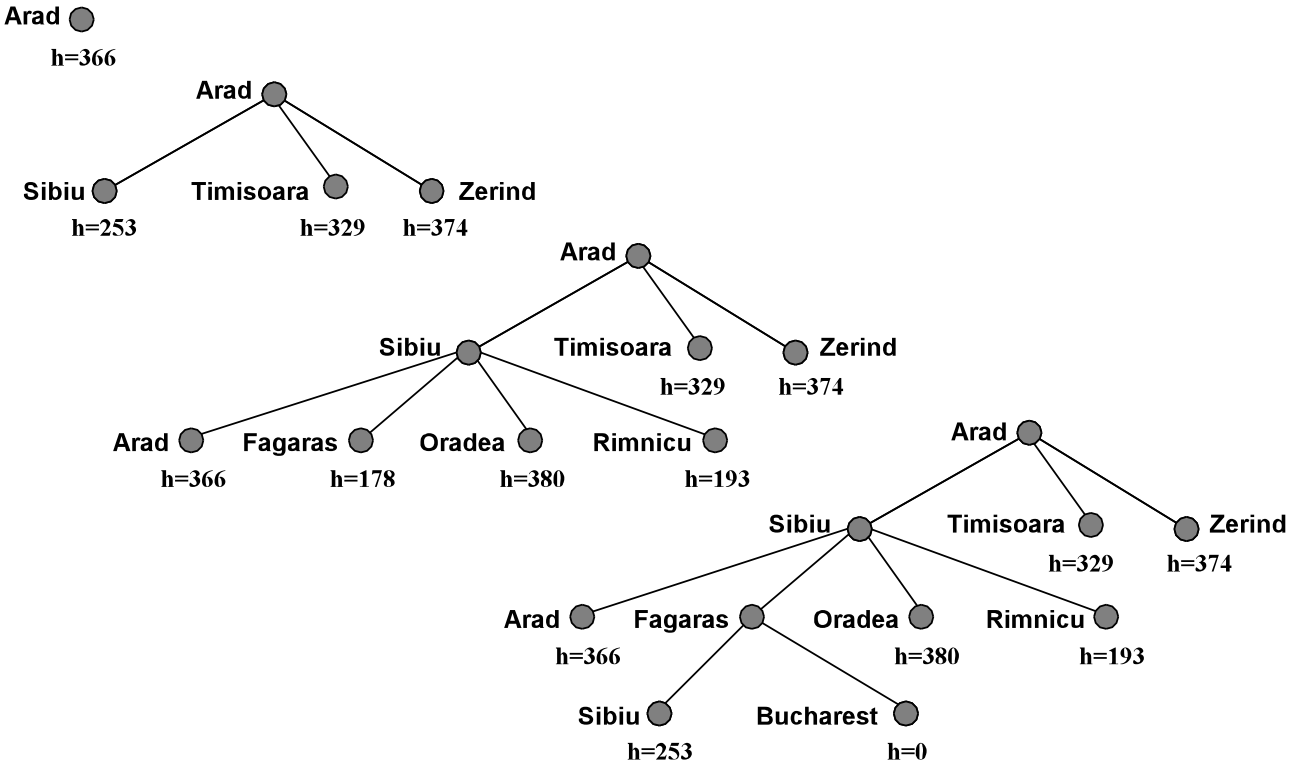
Pro obdobné problémy je dobrou heuristickou funkcí **přímočará vzdálenost v řadě** k cíli, neboli *SLD* (Straight-Line Distance):

> $h_{SLD}(n)$ = přímočará řadová vzdálenost mezi *n* a cílovým místem.



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

h_{SLD} lze zde spočítat jen tehdy, jsou-li známy souřadnice měst v Rumunsku. Kromě toho zde platí, že cesta z **A** do **B** je převážně vždy správným směrem, takže tento druh přídavné informace umožňuje heuristice pomoci v redukci ceny vyhledávání. Další obrázek ukazuje pokrok lačného vyhledávání při hledání cesty z Aradu do Bucharesti:



Pomocí heuristiky *SLD* se expanduje jako první uzel z Aradu do Sibiu, protože je bližší (dle přídavné informace) Bucharesti než Zerind nebo Timisoara (viz vzdálenosti různých měst od Bucharesti). Dalším uzlem je ze stejného důvodu Fagaras, a odtud už to je přímo do Bucharesti (ta je cílem, protože její vzdálenost od sebe sama je nula). V tomto příkladě funguje heuristika skvěle, protože *našla řešení* bez expanse uzlů, které nejsou na cestě. Je však cesta *optimální*?

Optimální není, protože nalezená cesta je o 32 km delší než cesta přes Rimnicu Vilcea a Pitesti. Optimální cesta nebyla heuristikou nalezena, protože Faragas je Buchuresti blíže z hlediska *SLD* než Rimnicu Vilcea, proto cesta přes Faragas byla expandována jako první.

Ukázaná heuristická strategie preferuje největší nalezený kus cesty k cíli vzhledem k tomu, že tento velký kus odebere největší část ze zbývající cesty do cíle (Fagaras–Bucharest = 211 km, Rimnicu Vilcea–Pitesti–Bucharest = 97+101 km, a 97 odebere méně ze zbytku než 211), aniž by se starala o to, zda je to nejlepší vzhledem k délce cesty—proto je název hledání "lačné, hltavé" (greedy). Přesto praxe ukazuje, že lačnost přináší velmi dobré výsledky (i když se jedná o jeden ze sedmi smrtelných hříchů). Lačnost přináší jako positivum rychlé nalezení cíle, i když ne vždy optimální. Optimalita by vyžadovala důkladnější analýzu z hlediska uvažování o celé dlouhé vzdálenosti, nikoliv pouze bezprostřední nejlepší výběr.

Lačné hledání je citlivé na chybný počátek. Uvažme problém cesty z **Iasi** do **Fagarasu**: heuristika nabízí první expansi uzlu **Neamt**, což je ovšem slepá ulička. Řešením je expanze **Vaslui**, což je ale z hlediska heuristiky do cíle dále. Z **Vaslui** pak do **Urziceni**, **Bucharesti** a **Fagarasu**, což z hlediska heuristiky jsou zbytečné expanse uzlů, pokud je okamžitý stav v **Neamtu**. Navíc, při zanedbání opatrnosti vzhledem k detekci opakovaných stavů, může dojít k nekonečným oscilacím mezi **Neamtem** a **Iasi**.

Lačné hledání připomíná hledání prvně do hloubky v tom, že preferuje jednu cestu do cíle, ale při objevení slepé uličky se vrací zpět. Rovněž má stejné neduhy: není optimální a není kompletní (může se vydat na nekonečnou cestu a nikdy se nevrátit k vyzkoušení odlišné možnosti). Nejhorší časová složitost je $O(b^m)$, kde m je max. hloubka hledání. Stejná je i prostorová složitost (uzly jsou uchovávány v paměti). Podstatná redukce složitosti vyžaduje kvalitní heuristiku h a také závisí na konkrétním problému.

Minimalizace celkové ceny cesty: hledání A*

Lačné hledání *h(n)* má uvedené přednosti a nedostatky. Uniformně-cenové hledání *g(n)* minimalizuje cenu cesty do daného okamžiku; je optimální a kompletní, ale může být velmi neefektivní. Kombinací výhod obou strategií lze získat alternativní funkci, přičemž postačuje prostě jejich hodnoty sečíst:

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

Protože *g(n)* dává cenu cesty od startu do *n* a *h(n)* odhaduje cenu nejlevnější cesty z *n* do cíle, pak platí *f(n)* = odhadnutá cena nejlevnějšího řešení přes *n*.

Při hledání nejlevnějšího řešení je tedy rozumné prvně zkusit uzel s nejmenší hodnotou *f*. Existuje dokonce důkaz, že tato strategie je kompletní a optimální za předpokladu určitého omezení pro *h*:

> Omezením je výběr funkce *h* takové, která *nikdy nepřecení* cenu dosažení cíle. Taková funkce *h* se nazývá **přijatelná heuristika**.

Přijatelné heuristiky jsou v principu optimistické, protože předpokládají, že cena řešení problému je menší než je ve skutečnosti. Tento předpoklad se přenáší do funkce *f*:

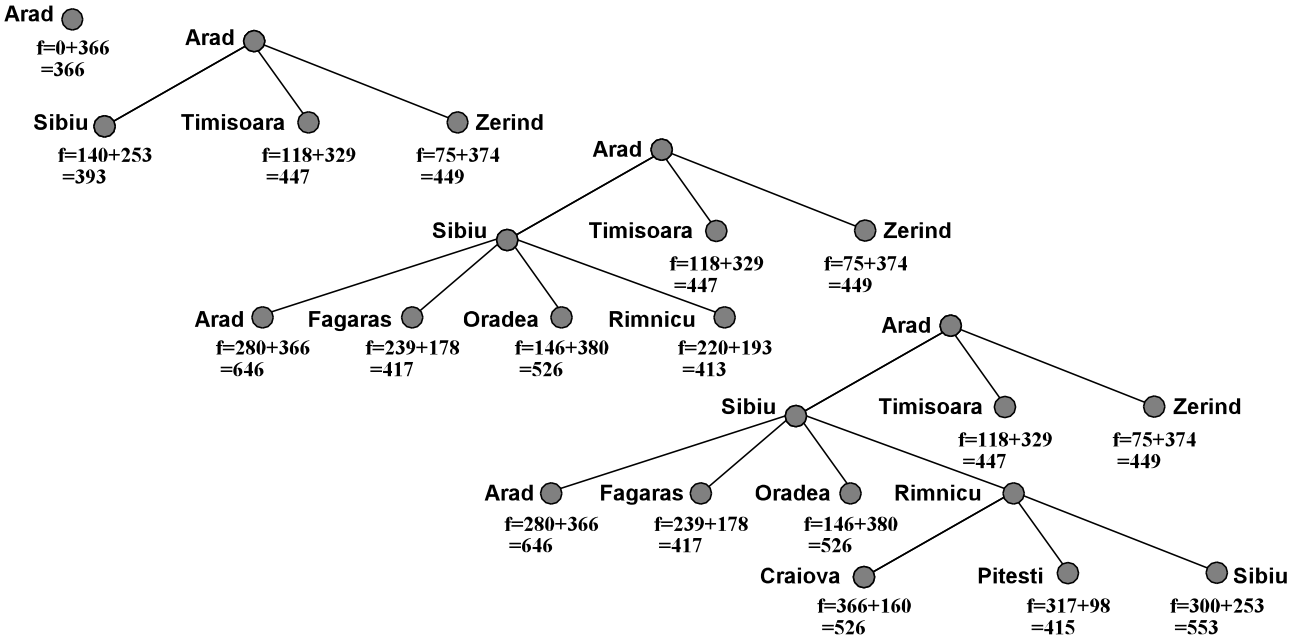
> Je-li *h* přijatelná, pak *f(n)* nikdy nepřecení skutečnou cestu nejlepšího řešení přes *n* (při *přecenění* ceny by pak cesta nebyla vybrána, i když je správným řešením).

Hledání prvního nejlepšího s použitím *f* jako vyhodnocovací funkce a *h* jako přijatelné funkce se nazývá **hledání A***. Symbolický zápis funkce:

```
function A*-Search(problem) returns řešení nebo neúspěch

     return Best-First-Search(problem, g+h)
```

Příkladem přijatelné heuristiky je výše uvedená h_{SLD} (nejkratší cesta mezi dvěma body je přímá cesta—úsečka).

Obrázek ukazuje několik prvních kroků hledání A* při použití heuristiky h_{SLD}. Za povšimnutí stojí, že A* preferuje expansi z Rimnicu Vilcea před expansí z Fagarasu. Přestože Fagaras je blíže Bucharesti, cesta do Fagarasu není tak efektivní v přiblížení se Bucharesti jako cesta do Rimnicu Vilcea (uzly mají označení $f = g+h$, přičemž hodnoty h jsou *SLD* vzdálenosti do Bucharesti z dříve uvedeného obrázku):



Vlastnosti hledání A*

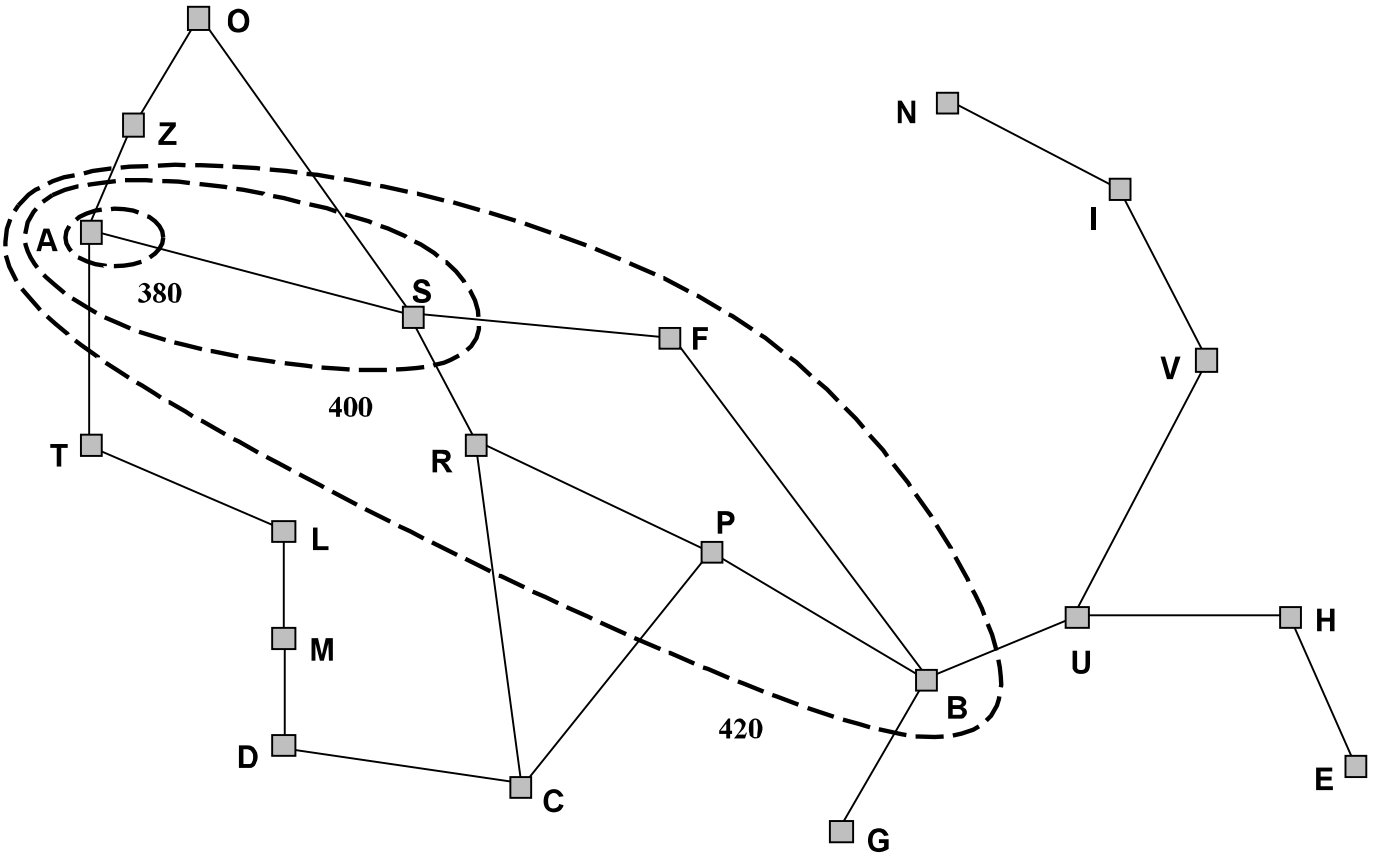
Obrázek hledání pomocí A* demonstruje jednu základní věc: podél libovolné cesty z kořene, *f*-cena nikdy neklesá. To není náhodou a heuristiky, které tuto vlastnost nenarušují, se nazývají **monotónní** (v r. 1984 bylo dokázáno, že heuristika je monotónní tehdy a jen tehdy, pokud splňuje pravidlo tzv. *trojúhelníkové nerovnosti*—přímočaré vzdálenosti tuto podmínku samozřejmě splňují, takže *SLD* je monotónní).

Pokud by heuristika nebyla monotónní, lze ji upravit na monotónnost:

Uvažujme dva uzly *n* a *n'*, kde *n* je rodičovský uzel *n'*. Dále nechť např. *g(n)* = 3 a *h(n)* = 4, takže *f(n)* = *g(n)*+*h(n)* = 7. Víme tedy, že skutečná cena cesty k řešení je nejméně 7. Předpokládejme ještě, že *g(n')* = 4 a *h(n')* = 2, takže *f(n')* = *g(n')*+*h(n')* = 6. Z uvedeného je zřejmé, že se jedná o nemonotónní heuristiku *h*. Je ale zřejmé, že *libovolná cesta přes n' je také cestou přes n*, tedy hodnota 6 je bezvýznamná, neboť již víme, že skutečná cena musí být nejméně 7. Musíme tedy kontrolovat při generování nového uzlu, zda jeho *f*-cena je menší než *f*-cena jeho rodiče; pokud je cena u potomka menší, použije se prostě cena rodiče:

$$f(n') = max[f(n), g(n')+h(n')].$$

Takto se ignorují hodnoty, které se mohou vyskytnout s nemonotónními heuristikami a uvedený stav se nazývá rovnice **max-cesty** (pathmax equation). Pokud vztah max-cesty použijeme, pak *f* bude vždy neklesající podél každé cesty z kořene, za předpokladu přijatelnosti *h*. Z toho vyplývá další vlastnost A*, tj. pokud *f* nikdy cestou z kořene neklesá, lze ve stavovém prostoru vytvořit tzv. **obrysy**:



Na mapce (stavovém prostoru) jsou obrysy pro $f = 380$, $f = 400$ a $f = 420$, kde **Arad** je počáteční stav. Uzly uvnitř daného obrysu mají f-ceny nižší než je hodnota daného obrysu.

A^* expanduje uzly s nejnižší f, tedy hledání postupuje koncentricky v pásmech podle narůstání f.

Při hledání s uniformní cenou (A^*-hledání za použití $h = 0$) jsou pásma "cirkulární" kolem počátečního stavu. Se vzrůstající přesností heuristiky se pásma začínají protahovat směrem k cílovému stavu a zužují se kolem optimální cesty.

Nechť f^* je cena optimální cesty k řešení. Pak lze o A^* říci následující:

- A^* expanduje všechny uzly, pro něž platí $f(n) < f^*$.
- A^* může dále expandovat některé uzly přímo na "cílovém obrysu", kde platí $f(n) = f^*$, před výběrem cílového uzlu.

Intuitivně je zřejmé, že první řešení musí být optimální, protože uzly ve všech následujících obrysech mají vyšší f-cenu a tudíž vyšší g-cenu (důvodem je, že všechny cílové stavy mají $h(n) = 0$, takže zdražení způsobí g).

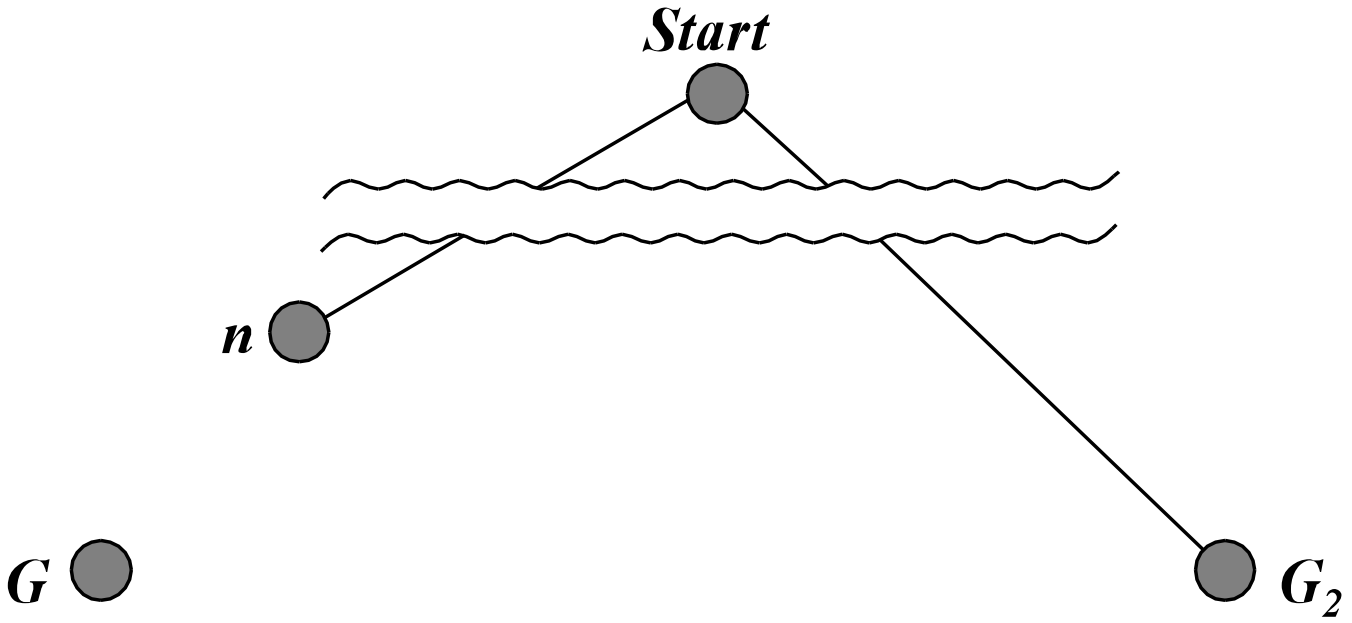
Dále je intuitivně zřejmé, že A^*-hledání je kompletní. Přidáváním pásem s rostoucími hodnotami f se nakonec dostaneme do pásma, kde f je rovno ceně cesty do cílového stavu.

Kromě uvedeného stojí za zmínku, že mezi optimálními algoritmy tohoto typu—tj. algoritmy, které prodlužují vyhledávací cesty z kořene—je A^* **optimálně účinný** pro jakoukoliv danou heuristickou funkci:

> Libovolný algoritmus, který *neexpanduje* všechny uzly v pásmech mezi kořenovým a cílovým pásmem, musí počítat s rizikem opominutí optimálního řešení (rozsáhlý důkaz byl podán v r. 1985).

Důkaz optimality A*

Nechť G je optimální cílový stav s cenou cesty f^*. Nechť G_2 je suboptimální cílové řešení, tj. cílový stav s cenou cesty $g(G_2) > f^*$. Uvažujme situaci, kdy A* vybral G_2 z fronty. Protože G_2 je cílovým stavem, ukončí hledání se suboptimálním řešením (viz obrázek—uzel n je uzel na cestě k optimálnímu řešení G).



Ukážeme, že k uvedenému závěru—suboptimálnímu výběru—nemůže dojít. Předpokládejme, že uzel n je v daném okamžiku listem na optimální cestě do G (nějaký takový uzel musí existovat, ledaže by již cesta byla zcela expandována—zde by ovšem bylo vráceno G). Protože h je přijatelná heuristika, pak:

$$f^* \geq f(n).$$

Dále, pokud n není vybrán pro expansi přes G_2, platí:

$$f(n) \geq f(G_2).$$

Z toho plyne:

$$f^* \geq f(G_2).$$

Protože G_2 je cílovým stavem, platí:

$h(G_2) = 0$, z čehož zjevně plyne $f(G_2) = g(G_2)$.

Tím bylo s použitím předpokladů dokázáno, že

$f^* \geq g(G_2)$.

To je ale v rozporu s předpokladem, že G_2 je suboptimální, takže A* nikdy nevybere pro expansi suboptimální cíl. Protože vrací jedině řešení vybrané pro expansi, musí A* být optimálním algoritmem. □

Důkaz úplnosti A*

A* expanduje uzly v pořadí vzrůstající *f*, takže nakonec musí dojít k expansi k dosažení cílového stavu. To platí s výjimkou případu, kdy je nekonečně mnoho uzlů s $f(n) < f^*$. Důvod k existenci nekonečného počtu uzlů je buď: *a) existuje uzel s nekonečně velkým počtem větvení*, nebo *b) existuje cesta s konečnou cenou, ale s nekonečným počtem uzlů podél ní* (starý paradox typu "zajíc nikdy nedožene želvu"). Z toho plyne, že A* je kompletní v **lokálně konečných grafech** (tj. grafech s konečným faktorem větvení) za předpokladu, že existuje nějaká kladná konstanta δ taková, že každý operátor stojí nejméně δ.

Složitost A*

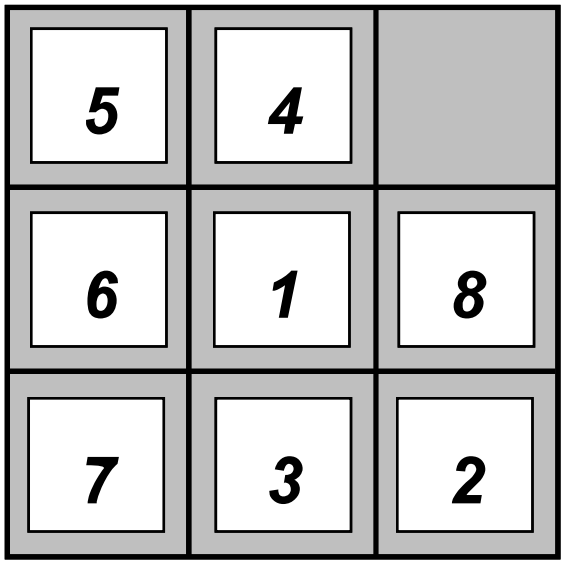
Z hlediska kompletnosti, optimálnosti a optimální efektivnosti A* není odpovědí na všechny požadavky hledání—pro většinu problémů je totiž počet cílových uzlů uvnitř obrysu, vymezujícího cílový prostor vyhledávání, stále exponenciální vzhledem k délce řešení. Komplikovaný důkaz byl však vytvořen pro to, že exponenciální nárůst se objevuje, pokud ovšem chyba heuristické funkce neroste rychleji než logaritmus skutečné ceny cesty. Podmínka pro sub-exponenciální nárůst je:

$$|h(n) - h^*(n)| \leq O(\log h^*(n)),$$

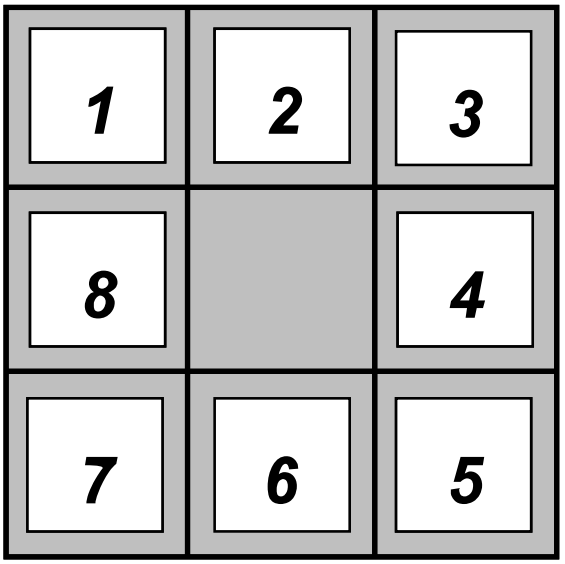
kde $h^*(n)$ je *skutečná* cena cesty z *n* do cíle. Pozn.: pro A* není rozhodující čas, nýbrž paměť, protože udržuje v paměti všechny generované uzly.

Heuristické funkce

Hlavolam s posouváním osmi čtverečků v devíti polích patří k nejstarším problémům s heuristickým vyhledáváním:



Start State



Goal State

Typické řešení má zhruba 20 kroků (závisí to na počátečním stavu). Faktor větvení je cca 3 (je-li prázdné pole uprostřed, pak $b = 4$; v rohu je $b = 2$; podél hran je $b = 3$). Prohledávání vyčerpávající všechny možnosti do hloubky 20 zkoumá přibližně $3^{20} = 3.5 \times 10^9$ stavů.

Pokud by se důsledně testovaly stavy opakování, lze redukovat počet stavů radikálně, neboť existuje pouze 9! = 362 880 různých rozmístění pro 9 polí. To je stále velmi mnoho, proto je vhodné najít dobrou heuristiku. Pokud je cílem nalezení nejkratšího řešení, je zapotřebí mít heuristickou funkci, která nikdy nepřecení počet kroků do cíle. Dvě možnosti:

- h_1 = počet čtverečků na chybných pozicích. Na obrázku je umístěno 7 z 8 chybně, takže počáteční stav má $h_1 = 7$; je to přijatelná heuristika, protože každý chybně umístěný čtvereček musí být alespoň jednou posunut.

- h_2 = součet vzdáleností čtverečků od jejich koncových pozicí. Čtverečky se nemohou pohybovat diagonálně, takže se sečtou vzdálenosti vertikální a horizontální (tento typ výpočtu vzdáleností je znám jako **vzdálenost městských bloků** nebo **manhattanská vzdálenost**). h_2 je rovněž přijatelná, protože jakýkoliv posun pohne pouze jedním čtverečkem o jeden krok směrem k cíli. Čtverečky 1 až 8 z počátečního stavu na obrázku mají celkovou délku manhattanské cesty $h_2 = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18$.

Vliv přesnosti heuristiky na efektivitu

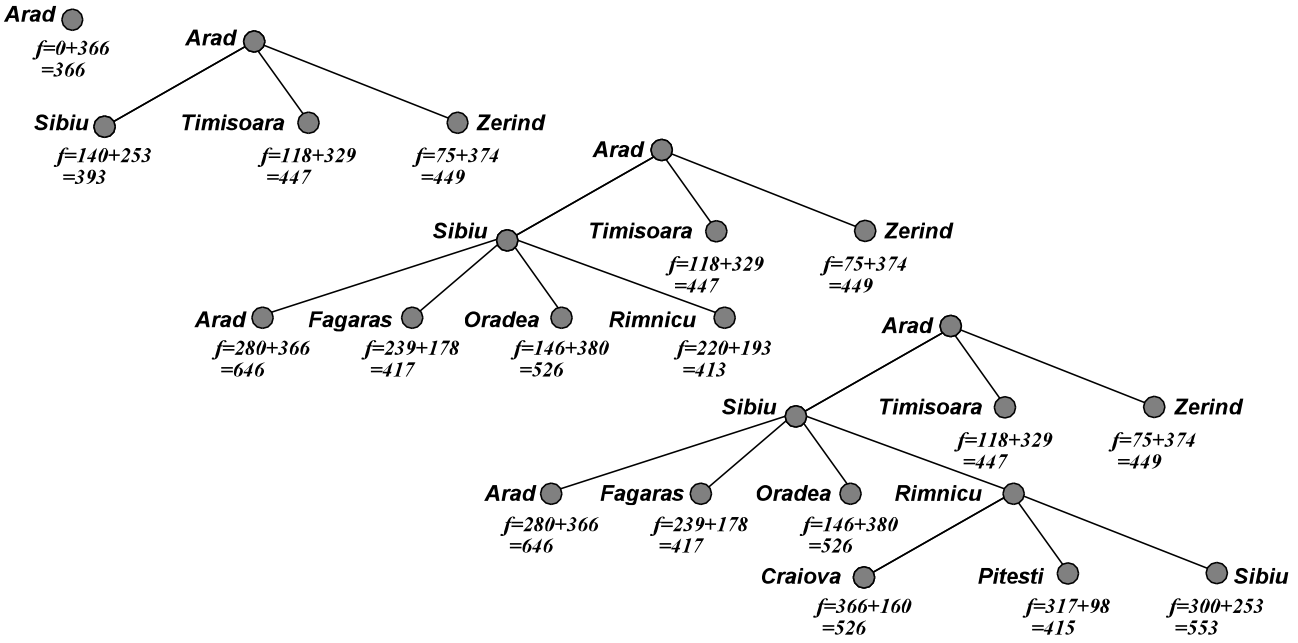
Jednou možností pro stanovení kvality heuristiky je **efektivní faktor větvení**, označovaný b^*. Je-li N celkový počet uzlů expandovaných pomocí A* a hloubka řešení je d, pak b^* je faktor větvení, který by měl rovnoměrně vyvážený strom hloubky d, aby obsahoval N uzlů:

$$N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d.$$

Např. když A* najde řešení v hloubce 5 za použití 52 uzlů, pak $b^* = 1.91$. Dobrá heuristika by se měla blížit $b^* = 1$. Pro 100 náhodně vygenerovaných pozic hlavolamu s 8 čtverečky (a za použití iterativního hloubkového hledání `Iterative-Deepening-Search` IDS s A*, h_1 a h_2) lze získat srovnání pro průměrné hodnoty počtu expandovaných uzlů:

d	cena hledání			efektivní faktor větvení		
	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

Otázkou může být, zda h_2 je *vždy* lepší než h_1. Protože z definic obou heuristik platí pro libovolný uzel, že $h_2(n) \geq h_1(n)$, pak odpověď zní ano, tedy h_2 **dominuje** h_1. Dominace je převedena do efektivnosti přímočaře, protože A* s h_2 expanduje průměrně méně uzlů než s h_1. Např. pro již uvedený problém:



Bylo již dříve uvedeno, že expandovány budou uzly, pro něž platí $f(n) < f^*$. To je totéž jako tvrzení, že budou expandovány uzly s $h(n) < f^* - g(n)$ vzhledem k definici $f(n)$. Protože h_2 je přinejmenším tak velká jako h_1 pro všechny uzly, pak každý uzel expandovaný pomocí hledání A* s h_2 bude rovněž expandován s h_1 a h_1 může ještě způsobit expanzi jiných uzlů. Z toho plyne určitý závěr či doporučení: *Je lépe použít heuristiku s vyššími hodnotami, neboť nedojde k přecenění* (konečný výsledek jednoduše může být takový nebo lepší).

Hledání vhodné heuristiky vyžaduje velmi často experimentování, např. náhodné generování počátečních konfigurací, z nichž se vyzkouší vyhledávání s různými možnými heuristikami, a výsledná statistika doporučí nejvhodnější z nich. Vzhledem k zavedení pravděpodobnosti výběru ovšem dochází ke ztrátě záruky na přijatelnost heuristiky ve výše zmíněném smyslu, výhodou je šance expandovat méně uzlů, tj. levnější řešení.

Poměrně často je možné nalézt vlastnosti nutné pro řešení, což může přispět k heuristické vyhodnocovací funkci (v šachu k dosažení matu je zapotřebí mít určitý počet určitých figur apod.).

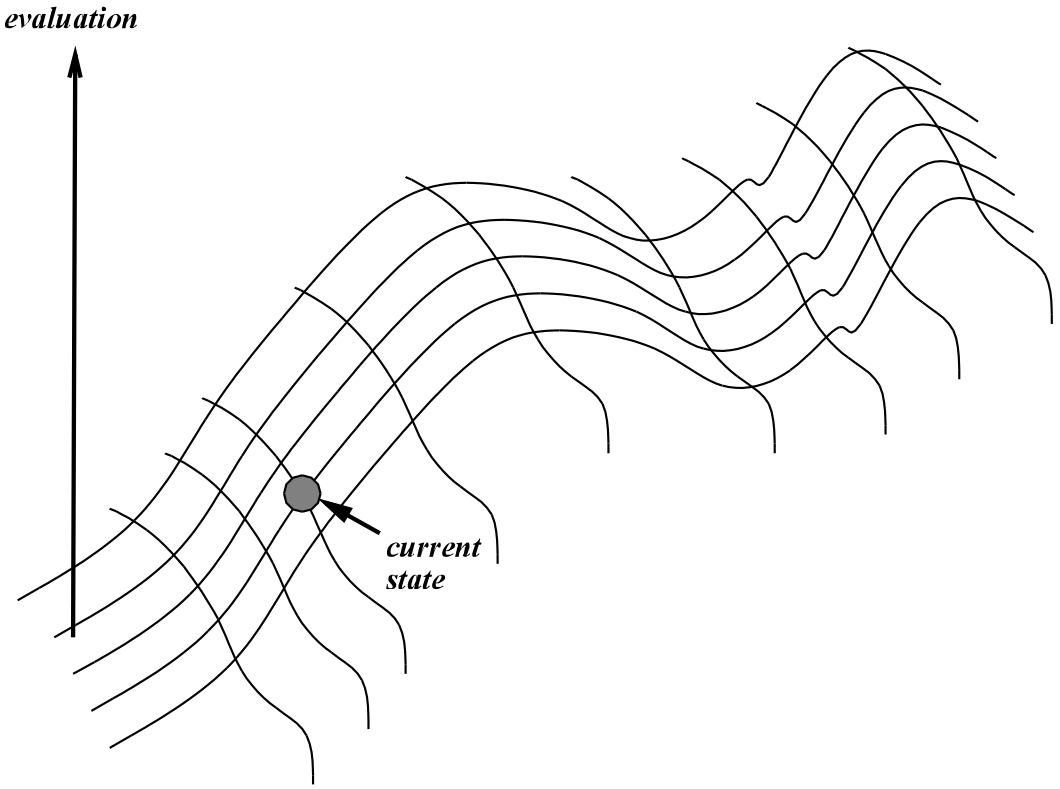
Algoritmy iterativního vylepšování

Pro řadu tzv. dobře definovaných problémů (8 dam, VLSI) platí, že popis stavu sám o sobě obsahuje veškerou informaci potřebnou pro řešení. Konkrétní cesta k řešení (zda je řešení dosaženo tak či jinak) nemusí být (a často není) relevantní. V těchto případech poskytují algoritmy s **iterativním vylepšováním** nejpraktičtější přístup.

Např. lze začít se všemi 8 dámami na šachovnici nebo všemi vodiči v konkrétních spojovacích kanálech řešeného obvodu. Pak můžeme pohybovat dámami ve snaze redukovat počet vzájemného napadání, nebo přemísťovat vodič z jednoho kanálu do druhého ve snaze redukovat zahlcení.

Obecnou ideou je začít s kompletní konfigurací a modifikacemi zlepšit kvalitu.

Pro představu lze uvažovat o stavech umístěných na povrchu nějaké krajiny. Výška povrchu v nějakém místě odpovídá vyhodnocovací funkci (evaluation) pro stav v daném bodě (current state):



Podle myšlenky iterativního vylepšování je vhodné se v krajině pohybovat a hledat nejvyšší vrcholek, tj. optimální řešení. Iterativní vylepšování obvykle udržuje údaje pouze o cestě aktuálního uzlu a nehledí dále než k bezprostředním sousedům daného stavu (hledání vrcholku hory v mlze zároveň s postižením zapomnětlivostí). Přes zmíněné nedostaky je tento postup v řadě velmi obtížných případů praktický—typickou ukázkou použití zmíněného postupu jsou umělé neuronové sítě.

Jednou ze skupin zmíněných algoritmů je tzv. **slézání kopce** (v originále *hill-climbing*, někdy pojmenované také jako *gradient descent*, tj. **gradientní sestup**). Další skupinou jsou algoritmy tzv. **simulovaného žíhání** (*simmulated annealing*). Zde se zmíníme o lezení po kopcích.

Slézání kopce (gradientní sestup)

Algoritmus je symbolicky popsán následovně:

```
function Hill-Climbing(problem) returns stav řešení

     inputs: problem // problém
     local variables:   current // aktuální stav-uzel
                        next // další stav-uzel

     current ← Make-Node(Initial-State[problem])
     loop do
          next ← následný uzel s nejvyšší hodnotou
          if Value[next] < Value[current] then return current
          current ← next
     end
```

Ve smyčce se hledající jednoduše pohybuje směrem ke vzrůstající hodnotě (do kopce). Algoritmus nepracuje s vyhledávacím stromem, takže datová struktura pro uzel musí pouze zaznamenávat stav a jeho hodnotu `Value`. Je-li na cestě vzhůru několik ekvivalentně nejlepších uzlů, pak se obvykle vybírá náhodně jeden z nich.

Uvedený jednoduchý postup má tři známé nedostatky:

- **Lokální maxima:** po dosažení lokálního maxima se algoritmus zastaví, přestože globální maximum (neznámo kde v krajině je) je mnohem lepší.
- **Roviny:** ve všech směrech poskytuje vyhodnocovací funkce stejnou hodnotu, takže postup krajinou vede k náhodné cestě.
- **Hřebeny:** hřeben má strmě klesající strany, takže se na něj lze dostat snadno, ale další stoupání hřebenu k vrcholku je velmi mírné a dochází k oscilaci hledání ze strany na stranu se zanedbatelným pokrokem v přibližování se k maximu.

V každém případě se algoritmus dostane do bodu, odkud se nelze dostat výš. V takových případech se lze pokusit znovu z jiného startovacího bodu—k tomu se používá metoda zvaná **slézání kopce s náhodným počátkem**, takže dochází k sérii hledání z náhodně vybraných různých míst, a nejlepší dosažené řešení se uchovává. Ukončení vyhledávání pak závisí na tom, zda byl stanoven maximální předem stanovený počet těchto hledání nebo zda doposud dosažený nejlepší výsledek nebyl po nějaký počet iterací zlepšen.

Pro dostatečný počet iterací může (ovšem i nemusí) hledání s náhodným restartem nakonec najít optimální řešení.

Úspěch zmíněného typu hledání velmi silně závisí na tvaru povrchu, který je dán prostorem uvažovaných stavů. S malým počtem lokálních maxim je řešení nalezeno rychle. Realistické problémy však představují spíše podobu povrchu dikobraza.

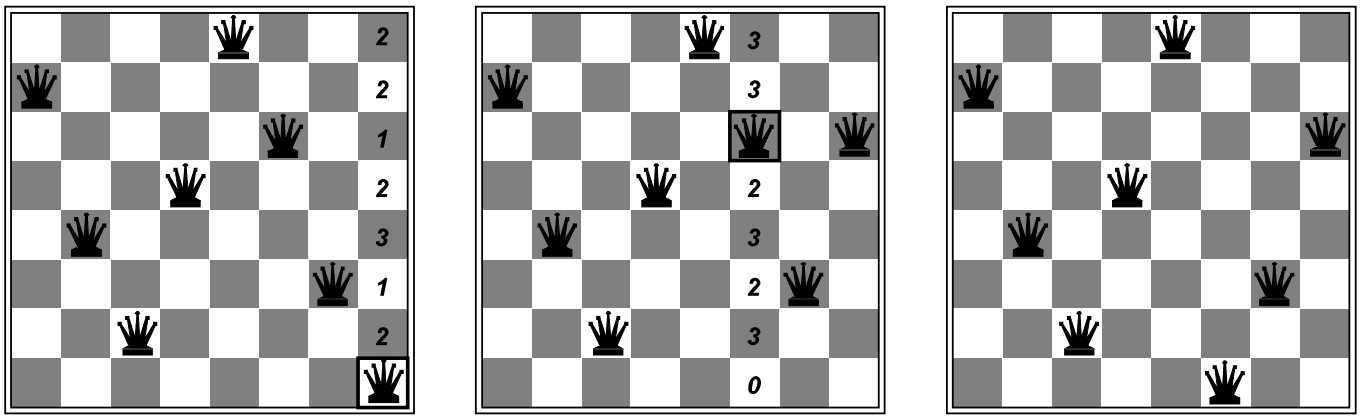
Pro NP-kompletní problémy nelze dosáhnout lepší než exponenciální časové závislosti. Přesto jsou takto založené metody velmi často úspěšné proto, že poskytují velmi dobré řešení, i když to nemusí být zrovna optimum (a často není možné ani zjistit, jaké to optimum vlastně je). Proto se použije přijatelný výsledek dosažený po nepříliš velkém počtu iterací—praktické hledisko hraje často výraznou roli.

Aplikace pro problémy vyhovující omezením (CSP)

Problémy vyhovující omezením (Constraint Satisfaction Problems) mohou být řešeny metodami iterativního zlepšování tak, že napřed jsou všem proměnným dosazeny nějaké hodnoty a pak za použití modifikačních operátorů se konfigurace pohybuje směrem k řešení. Modifikační operátory jednoduše přiřazují jiné hodnoty proměnným.

Např. pro problém 8 dam je počátečním stavem 8 dam nějak rozestavených na šachovnici a modifikační operátor může dámou pohnout o políčko vedle.

Tyto algoritmy se nazývají **heuristické opravování**, neboť napravují nekonsistence aktuální konfigurace. Při **výběru nové hodnoty proměnné** se (heuristicky) vybírají takové hodnoty, které jako výsledek dávají minimální konflikty s ostatními proměnnými—tzv. heuristika **min-konflikt**:



Problém je zde vyřešen ve dvou krocích. Čísla ve sloupci s dámou, jejíž postavení se vylepšuje (**Dh1** v pravém dolním rohu), udávají, s kolika jinými dámami je v konfliktu. Dáma je přesunuta na pole **h6** (prostřední diagram), kde má konflikt s **Df6**. Dáma z pole **f6** našla nekonfliktní místo na poli **f1** a protože další konflikty nejsou, úloha je vyřešena. Překvapivě je metoda min-konflikt úspěšná pro mnoho problémů s omezením (CPS) , např. problém s milionem dam má průměrné řešení v méně než 50 krocích.

V nedávné době byla min-konflikt použita pro plánování pozorování vesmírného teleskopu Hubble (HST, Hubble Space Telescope), kde redukovala čas nutný pro naplánování týdenního pozorování *ze tří týdnů na přibližně 10 minut* (astronomové si mohou podávat žádosti o pozorování pomocí HST a velké množství velmi různých žádostí vyžaduje velmi dobrý rozvrh).