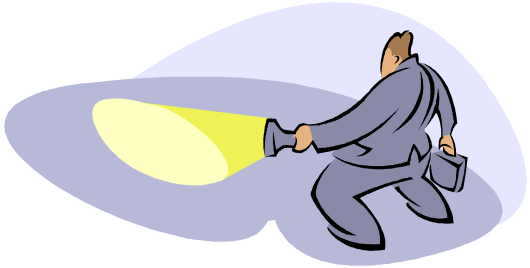
Obousměrné prohledávání

Tomáš Hřebejk



Motivační úlohy

- Jak řešit úlohy typu:
  - Nalezení nejkratší trasy (po silnici, po železnici…) z Prahy do Vídně.
  - Najít nejkratší řešení Loydovy patnáctky.
- Pomocí prohledávání!
- Nešlo by využít speciálních vlastností těchto úloh?
  - Řešení je možné hledat i od cíle.
  - Umíme odhadnout délku řešení.

2	14	8	6
13	5	3	9
12	15		4
11	7	1	10

Formální popis úlohy

- Je dán počáteční stav *s* a cílový stav *t*.
- Z každého stavu se můžeme dostat pomocí daných akcí do dalších stavů („následníků“).
- Každá akce má (nezápornou) cenu.
- Úkolem je najít nejlevnější posloupnost akcí, která vede ze stavu *s* do stavu *t*.
- Jedná se vlastně o nelezení nejkratší cesty mezi dvěma vrcholy v ohodnoceném grafu $G=(V, E)$, kde:
  - Vrcholy V jsou stavy.
  - Hrany E představují přechody mezi stavy pomocí akcí.

Dijkstrův algoritmus?

- Nešlo by použít Dijkstrův algoritmus?
  - Klasická implementace DA předpokládá, že je celý graf nějak uložen v paměti.
  - V našem případě je většinou dán implicitně.
  - Celý graf by se často do paměti ani nevešel!
- Uniform Cost Search (UCS)
  - Modifikovaná verze DA.
  - Pracuje i s implicitním grafem.
  - Speciální případ best-first search. (Bude později…)

Prohledávání – datové struktury

- V paměti je uložena jen část grafu v okolí *s* (*t*).
- Data jsou reprezentována stromem prohledávání, který je tvořen uzly.
- Uzel je určen stavem a cestou do tohoto stavu z *s* (*t*).
- Datová struktura pro uzel typicky obsahuje:
  - Stav
  - Ukazatel na rodiče
  - Délku cesty z kořene - $g(v)$
  - Další pomocné údaje (např. použitou akci)

Prohledávání – expanze uzlu

- Prohledávací strom se zvětšuje tak, že se postupně expandují listy.
- Při expanzi uzlu se vytvoří nový uzel pro každého následníka.
- Nové uzly se přidají do prohledávacího stromu jako potomci expandovaného uzlu.
- Expandované uzly – uzavřené
- Neexpandované uzly – otevřené
  - Tvoří „okraj“ (fringe, frontier).
- Pro jeden stav může být ve stromě více uzlů.
  - Řešení: Necháme si jen uzel s nejkratší cestou z kořene, ostatní zahodíme.

Best-first search

- Obecná třída prohledávácích algoritmů.
- Pro expanzi se volí „nejlepší“ uzel.
  - Uzel u s nejmenší hodnotou $f(u)$.
  - Jednotlivé algoritmy se liší (hlavně) ve funkci f.
    - Pro $f(u) = g(u)$ dostaneme Uniform-Cost Search.

Best-first search - pseudokód

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.

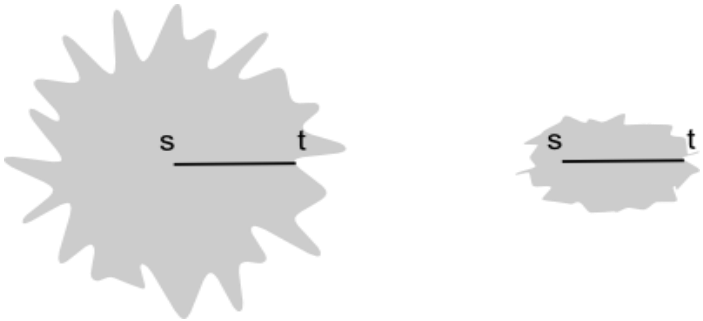
[P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, SCC-4(2):100*–107, 1968]

Požadavky na „dobrou“ $f(n)$

- Intuitivně: $f(n)$ by měla dávat (dolní) odhad na délku řešení přes uzel n.
- Chtěli bychom, aby platilo:
  1. Pro uzel n, který odpovídá cílovému stavu, je $f(n)$ rovno $g(n)$.
  2. Pokud n a n' jsou dva uzly pro **stejný stav** tak
     $$g(n') < g(n) \Rightarrow f(n') < f(n)$$
  3. Pokud je n' **potomek** uzlu n, tak
     $$f(n') \geq f(n)$$
- Pokud jsou tyto podmínky splněny, tak platí:
  - Do uzavřených uzlů jsou nalezeny nejkratší cesty.
  - Každý stav je expandován nejvýše jednou.
  - Algoritmus nalezne nejkratší cestu do cílového stavu. (Pokud cesta existuje a stavový prostor je konečný…)
- Podmínky jsou splněny například pro
  - $f(n) = g(n)$ (Uniform-cost search)
  - $f(n) = g(n) + h(n)$, kde h je konzistentní přípustná heuristika (**A***)
- Je možné použít i „ošklivější“ f

A*

- $f(n) = g(n) + h(n)$
- Heuristika h dává odhad na délku cesty do cílového stavu.
  - Přípustnost: $\forall n\ h(n) \leq h^*(n)$
    - $h^*(n)$ je délka optimální cesty do cílového stavu
  - Konzistence: $\forall n,n'\ h(n') \geq h(n) - d(n, n')$
    - $d(n, n')$ je délka (nejkratší) cesty z n do n'
    - Stačí, když to platí pro hrany (ekvivalentní definice).
- Pokud je použita přípustná heuristika, tak nalezené řešení je optimální.
- Pokud je navíc heuristika konzistentní, tak je každý stav expandován maximálně jednou (viz předchozí slide).



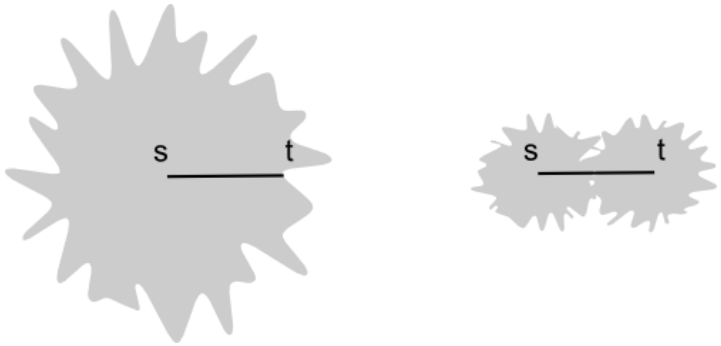
IDA*

- Problémem A* je velká paměťová náročnost.
- Řešením může být IDA*
  - Používá heuristiku stejně jako A*
  - Prohledávání do hloubky (lze implementovat rekurzí).
  - Více iterací, v každé je stanovena maximální hloubka vzhledem k $f(n)$.
  - Limit na hloubku se postupně zvyšuje.
- IDA* je jednodušší než A*.
- Potřebuje mnohem méně paměti.
- Stavy může expandovat opakovaně.
  - Jednak kvůli iteracím, ale hlavně proto, že si nepamatuje všechny navštívené stavy.
  - Exponenciální časová složitost v nejhorším případě (vzhledem k velikosti prozkoumané oblasti).
  - Lze částečně řešit.
    - Transpoziční tabulky.
    - Kombinace A* + IDA*

Obousměrné prohledávání

- Prohledávání probíhá „zároveň“ od počátku i od cíle.
- Cesta je nalezena, když se prohledávací stromy potkají.
- Graf může být orientovaný, ale musíme umět nalézt předchůdce.
- Proč obousměrně?
  - Představa: Počet uzlů roste s hloubkou víc než lineárně (často exponenciálně).
  - Je tedy lepší provést dvě prohledávání do poloviční hloubky.
- První nalezená cesta (délky L) nemusí být nejkratší.
  - Různá kritéria, kdy může prohledávání skončit, abychom měli jistotu, že bylo nalezeno optimální řešení.
    - Obousměrný UCS:

$$\min_{u \in O_1} g_1(u) + \min_{v \in O_2} g_2(v) \geq L$$

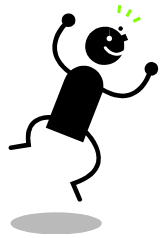


Obousměrné heuristické p. (1)

- Nešlo by využít heuristiku i při obousměrném prohledávání?
- BHPA – Ira Pohl, 1971
  - A* současně z obou směrů.
  - Uzel pro expanzi se vždy vybírá ze směru, kde je méně otevřených uzlů.
  - Téměř vždy pomalejší než A*…
- Poté různé experimenty s front-to-front vyhodnocováním.
  - BHFFA (1977), d-node retargeting (1984)
  - Pomalé, nebo nemusí najít optimum…
- BS* - James B.H. Kwa, 1989
  - Mírné vylepšení BHPA
    - Zahazování zbytečných uzlů.
    - Snaha najít (nějaké) řešení co nejdřív – horní odhad.

Obousměrné heuristické p. (2)

- BIDA* - Manzini, 1995; PS* - Dillenburg & Nelson, 1994
  - Perimetr okolo cíle; front-to-front vyhodnocování.
  - Lepší výsledky než jednosměrné prohledávání (IDA*) pro Loydovu patnáctku.
- Dynamická heuristika – Kaindl & Kainz, 1997
  - Vylepšení heuristiky pomocí prohledávání z druhého směru.
  - Souvisí s „vyváženými heuristikami“. (Ikeda et al., 1994)
- Mnoho dalších algoritmů…



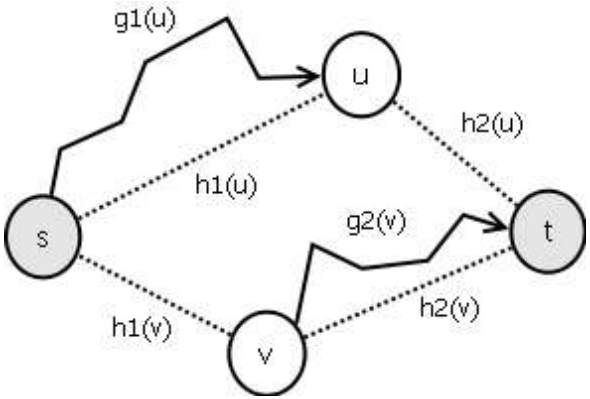
BS*

- A* z obou směrů (vylepšení BHPA)
- Požaduje konzistentní přípustnou heuristiku.
- Hlavní myšlenka: Je možné zahazovat uzly, přes které nemůže vést nejkratší (lepší) cesta.
  1. **nipping** – Pokud známe pro uzel nejkratší cestu do cílového stavu, tak ho není potřeba expandovat. Nejkratší cestu známe pro uzavřené uzly z druhého směru.
     - Je to ale komplikovanější - „bad nodes“
  2. **pruning** – V předchozím případě je možné zahodit potomky uzavřeného uzlu z druhého směru. („opožděný nipping“)
  3. **trimming, screening** – Je možné zahodit uzly, které nemají f-hodnotu menší než dosud nalezená nejkratší cesta (horní odhod délky cesty).
     - screening – při expanzi takové uzly ani nejsou vytvořeny
- Setkání stromů se kontroluje již pro otevřené uzly.
  - Je dobré získat co nejdřív horní odhad na délku cesty.
- Je možné použít jednodušší postup – bez pruningu.
  - Při expanzi uzlu se zahodí otevřený uzel z druhého stromu pro stejný stav (pokud existuje).
  - Nevytváří se nové uzly pro stavy, pro které již existuje uzavřený uzel.

Kaindl & Kainz (1997)

- Non-traditional Bidirectional search
  1. Napřed prohledávání z jednoho směru (best-first search). Prohledávácí strom je uložen v paměti.
  2. Poté prohledávání z druhého směru, při kterém se využívá prohledávací strom získaný během prvního prohledávání.
- Dynamická heuristika *h*‘
  - Při druhém prohledávání se používá heuristika *h*‘, která je vytvořena na základě dat získaných během prvního prohledávání.
  - Dává lepší odhady než *h* (***h*‘ dominuje *h***).
  - Původní heuristika ***h* musí být konzistentní** (a přípustná).
  - 2 metody: *Add* Method, *Max* Method
  - Lze dále vylepšit.

Značení

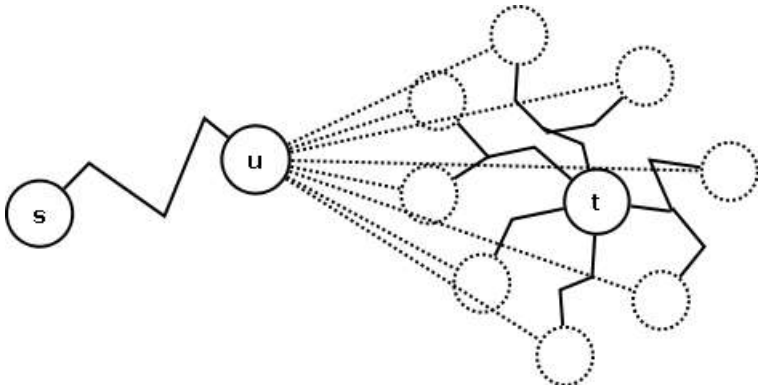


- O_1, O_2 – Množiny otevřených uzlů okolo s, resp. t.
- $g_1(u)$, $g_2(v)$ – Délka cesty z s do u, resp. z v do t v příslušném stromě prohledávání.
- $h_i(n)$– Heuristický odhad $g_i(n)$.
- $f_i(n) = g_i(n) + h_j(n)$, $j = 3 - i$
- $diff_i(n) = g_i(n) - h_i(n)$

Dynamická heuristika - principy

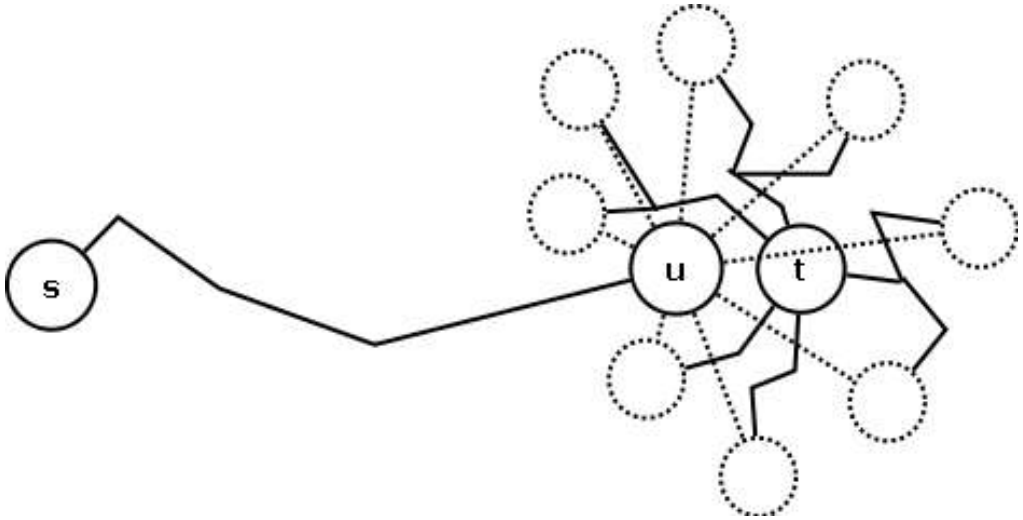
- BÚNO předpokládejme, že máme otevřený uzel $u \in O_1$ a chceme odhadnout vzdálenost do t. (Pro druhý směr je vše analogické.)
- Můžeme předpokládat, že u není uzavřený v druhém stromě (viz BS*).
- Nejkratší cesta z u do t musí vést přes nějaký otevřený uzel z O_2.
- Dolní odhad délky nejkratší cesty z u do t přes uzel $v \in O_2$ označme $h_{2,v}(u)$.
- Dynamická heuristika $h_2'(u)$ se spočítá jako:

$$h_2'(u) = \min_{v \in O_2} h_{2,v}(u)$$



Dynamická heuristika - I

- $h'_{\text{dist}}(u) = \max(h_2(u), mindist_2)$
  - $mindist_2 = \min\{ g_2(v) \mid v \in O_2 \}$
- Konzistentní přípustná heuristika (pokud se $mindist_2$ nemění).
- Opravuje příliš malé hodnoty heuristiky.

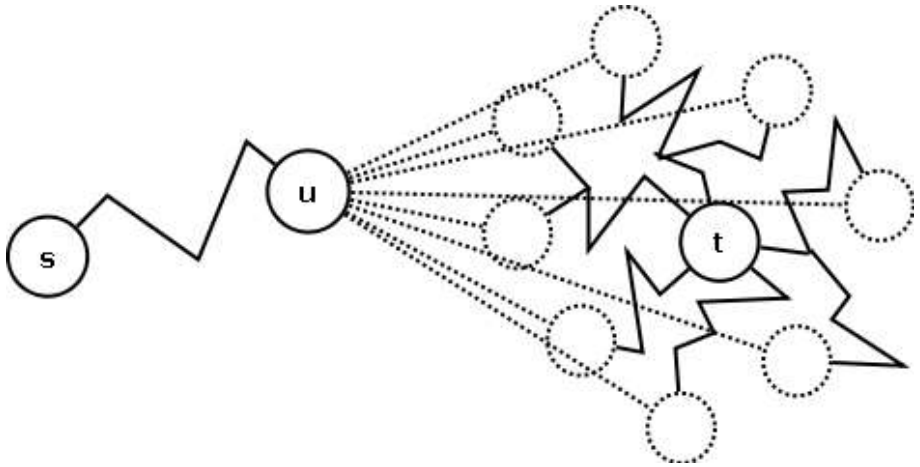


Diff

- $diff_i(n) = g_i(n) - h_i(n)$
  - Rozdíl mezi skutečnou vzdáleností a odhadem.
  - Pro přípustnou heuristiku je vždy nezáporný.
  - Pro konzistentní heuristiku $diff_2(n)$ směrem od t nikdy neklesne.
    - (Podobně pro druhý směr a $diff_1(n)$.)
- Pokud se prohledávací stromy setkají v uzlech u~u', tak je nalezena cesta délky L.
  - $L = g_1(u) + g_2(u')$
  - $L = f_1(u) + diff_2(u')$
  - $L = diff_1(u) + f_2(u')$

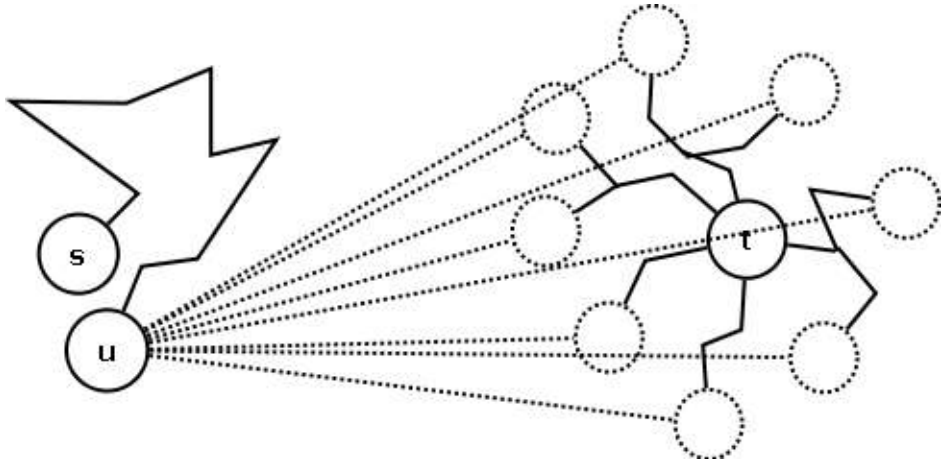
Dynamická heuristika - II

- „*Add* Method“
- $h'_{\text{add}}(u) = h_2(u) + mindiff_2$
  - $mindiff_2 = \min \{ diff_2(v) \mid v \in O_2 \}$
- Konzistentní přípustná heuristika (pokud se $mindiff_2$ nemění).
- Pokud je $mindiff_2$ nenulové, tak je lepší pro každý uzel.



Dynamická heuristika - III

- „*Max* Method“
- $\mathbf{h'_{max}(u) = max(h_2(u), minf_2 - h_1(u))}$
  - $minf_2 = \min \{ f_2(v) \mid v \in O_2 \}$
- Opět konzistentní přípustná heuristika (pokud se $minf_2$ nemění).
- Dává větší hodnoty pro uzly s velkým $diff_1(u)$.



Dynamická heuristika - závěr

- Dynamické heuristiky je možné zkombinovat
  - $h_2'(u) = \max(h'_{\text{dist}}(u), h'_{\text{add}}(u), h'_{\text{max}}(u)) =$

    $= \max(mindist_2, h_2(u) + mindiff_2, minf_2 - h_1(u))$
  - Vznikne opět přípustná konzistentní heuristika.
- Účelem prvního prohledávání je, aby výsledná dynamická heuristika byla co nejlepší. Jak to ale zajistit? (V jakém pořadí vybírat uzly pro expanzi?)
  - Podle *f*? Podle *diff*? Podle vzdálenosti? Kombinace?
  - Dočasné zahození uzlu s velkou f-hodnotou.
  - Střídavé prohledávání.
- Nešlo by najít ještě lepší dynamickou heuristiku?

$$h''_2(u) = \min_{v \in O_2} \max(g_2(v), h_2(u) + diff_2(v), f_2(v) - h_1(u))$$

Vyvážené heuristiky

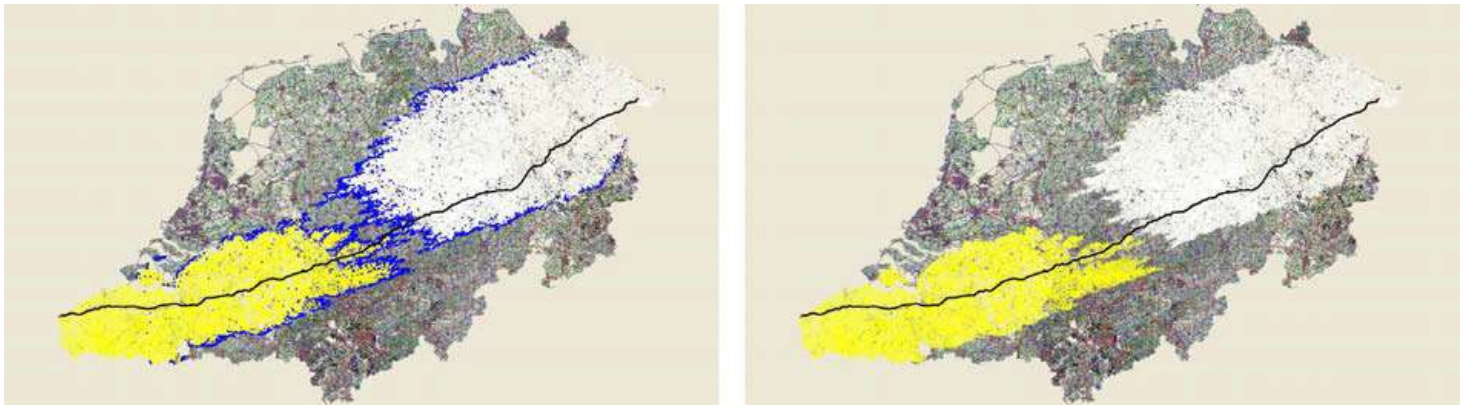
- Ikeda et al., 1994
- Jsou vyžadovány *konzistentní* heuristiky h_1 a h_2.
- Současné (best first) prohledávání z obou směrů.
  - $f_1(v) = g_1(v) + p_1(v)$
  - $f_2(v) = g_2(v) + p_2(v)$
- $\boldsymbol{p_1(v) + p_2(v) = C}$
  1. $p_1(v) = h_2(v)$; $p_2(v) = -h_2(v)$
  2. $p_1(v) = -h_1(v)$; $p_2(v) = h_1(v)$
  3. $\boldsymbol{p_1(v) = (h_2(v) - h_1(v))/2}$; $\boldsymbol{p_2(v) = (h_1(v) - h_2(v))/2}$
- Odpovídá klasickému obousměrnému prohledávání na grafu s redukovanými vahami.
  - $c'(u,v) = c(u,v) - h(u) + h(v)$
  - A* je ekvivalentní Dijkstrově algoritmu pracujícímu s redukovanými vahami.

BIDA* (Manzini, 1995)

- Zatím jsme vždy měli heuristiky, které odhadovaly vzdálenost jen do *t* a *s*.
- Často máme heuristiku $h(u,v)$, která dává dolní odhad na vzdálenost mezi libovolnými dvěma uzly *u* a *v*.
- BIDA*
  - Stejný přístup jako u dynamické heuristiky.
  - Opět 2 prohledávání.
    - V prvním se vytvoří perimetr okolo *t*.
    - Ve druhém se nalezne řešení pomocí IDA*.
  - Vzdálenost mezi uzlem $u \in O_1$ a $v \in O_2$ se odhadne pomocí $h(u,v)$.
  - $h'(u) = \min \{ h(u,v) + g_2(v) \mid v \in O_2 \}$
  - Při výpočtu $h'(u)$ je potřeba vyčíslovat *h* vícekrát.
    - V nejhorším případě pro každý uzel z O_2.
    - Ve skutečnosti je ale možné hodně výpočtů ušetřit.

Pijls & Post, 2008 (2010)

- „ A new bidirectional search algorithm with shortened postprocessing“ (European Journal of Operational Research)
- Obousměrný A* (jako BS*).
- Pamatuje si dosud nejlepší nalezenou cestu (délky *L*).
- main phase, postprocessing phase
- Po nalezení první cesty (postprocessing phase) jsou zahazovány uzly s velkým rozdílem.
  - $\boldsymbol{diff_i(u) + minf_j \geq L}$, $i \in \{1,2\}$, $j = 3 - i$

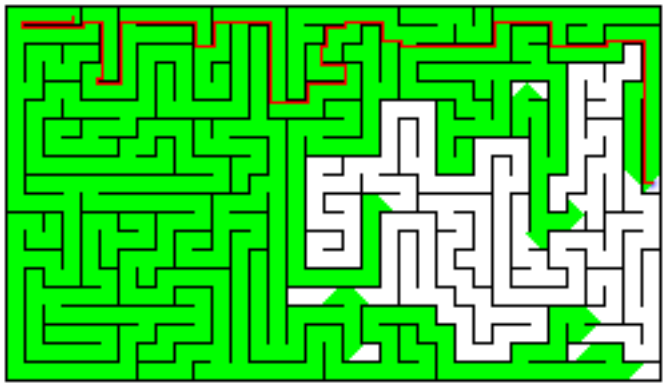


Search space on termination of the New-symmetric algorithm (left) and the Traditional algorithm (right)

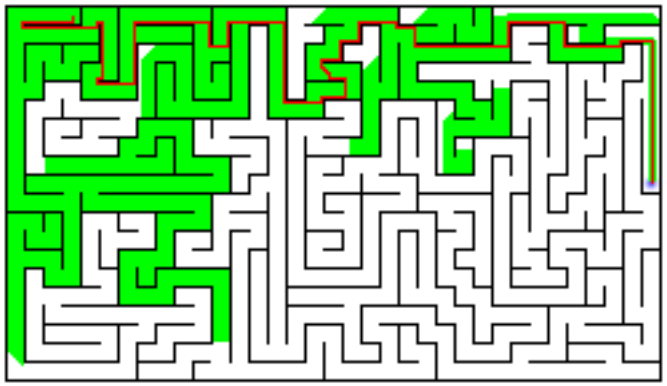
Frontier search

- Korf et al., 2005
- Prohledávání, při kterém se neudržují v paměti uzavřené uzly.
  - Menší paměťové nároky.
- Obecný postup, který lze aplikovat na různé algoritmy (jednosměrné i obousměrné).
- Hodí se především na neorientované grafy.
- Seznam zakázaných akcí u každého otevřeného uzlu.
  - Akce, které by vedly na uzavřený uzel.
- Řešení je nutné zrekonstruovat jiným způsobem.
  - Metoda rozděl a panuj.

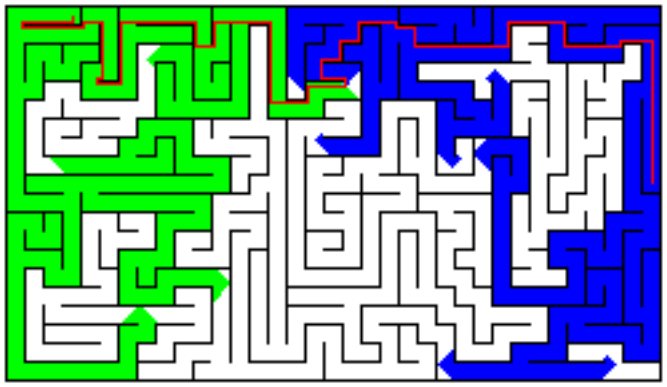
Experimenty - Labyrint



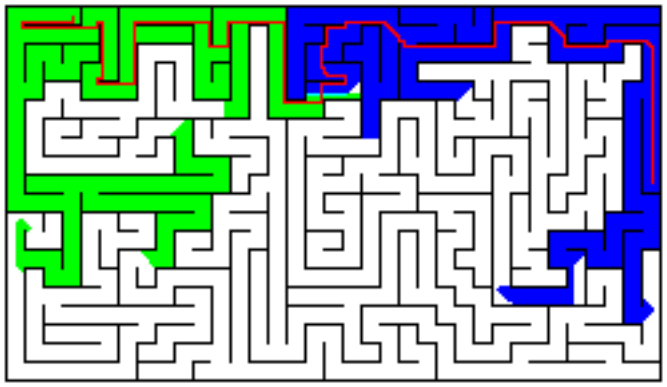
BFS



A*

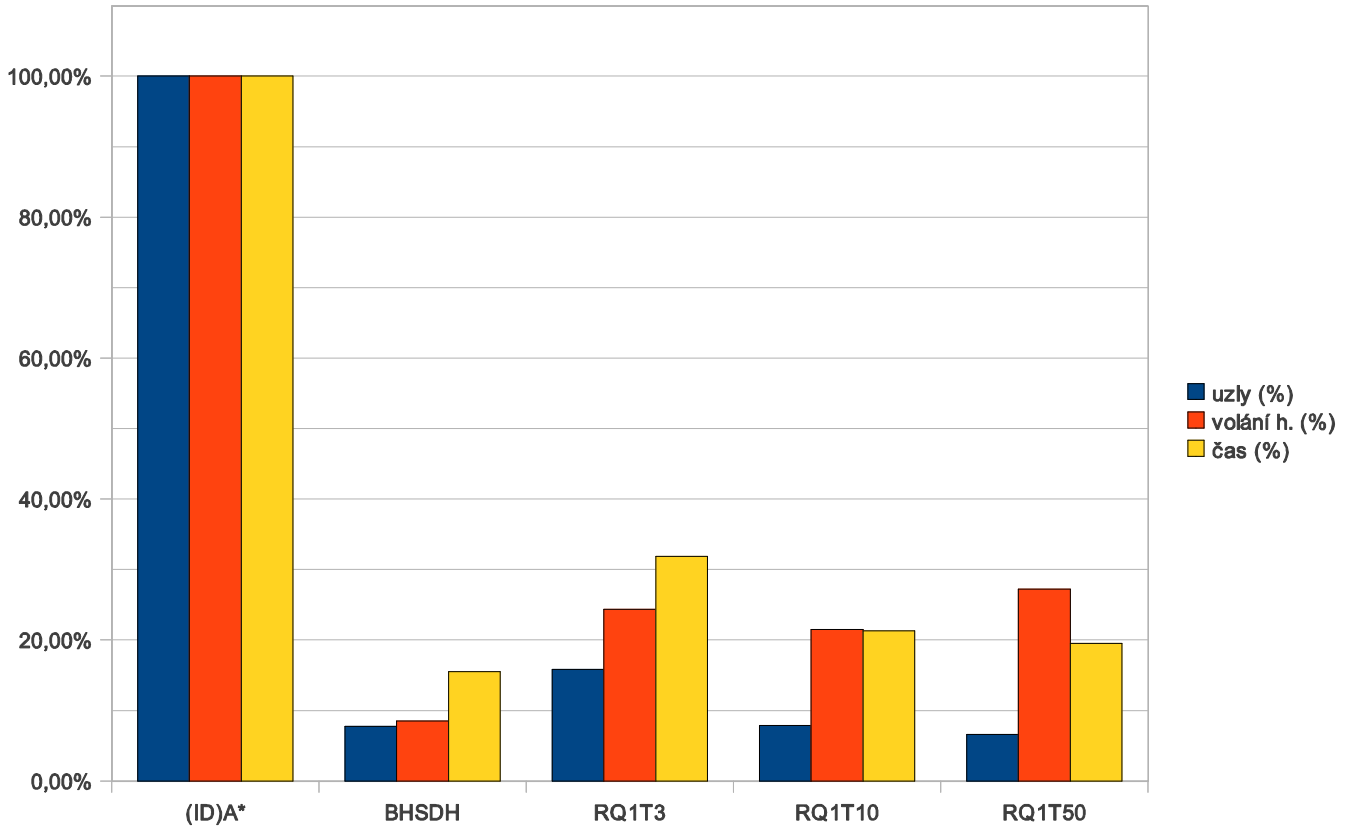


BiS

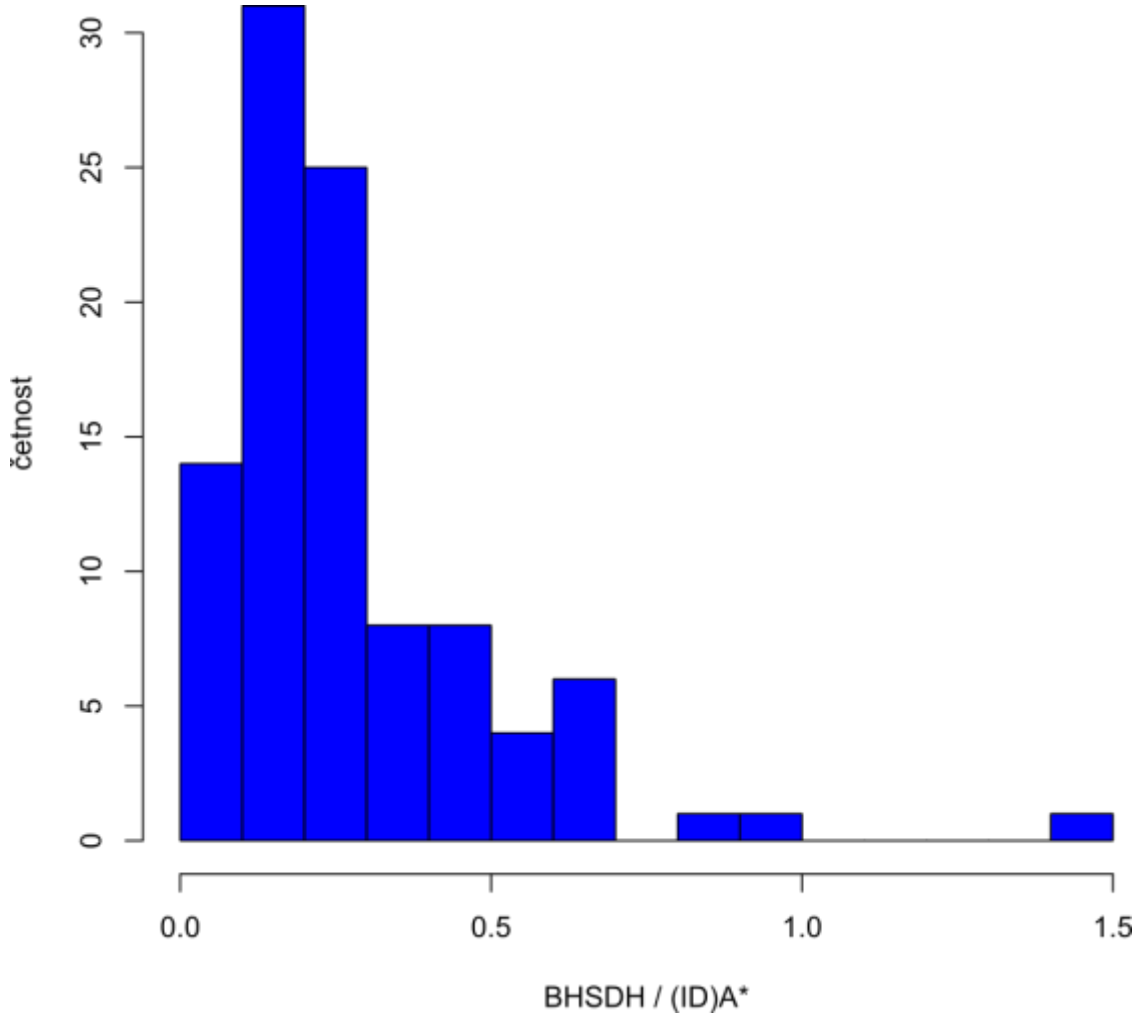


BHSDH

Experimenty – 15 puzzle



Experimenty – 15 puzzle



Jiné úlohy

Úloha	délka řešení	doba běhu (s)			relativně vůči (ID)A*	
		(ID)A*	RQ1T10	BHSDH	RQ1T10	BHSDH
15p_1.in	46	2	1	1	50,0%	50,0%
15p_2.in	52	1065	10	11	0,9%	1,0%
ch_2.in	16	29	6	5	20,7%	17,2%
mz_1.in	250	1	0	1	0,0%	100,0%
mz_2.in	810	26	21	16	80,8%	61,5%
sok_1.in	46	1	0	0	0,0%	0,0%
sok_3.in	142	19	0	0	0,0%	0,0%
sok_4.in	58	9	0	0	0,0%	0,0%
sok_6.in	154	>601	8	9	1,3%	1,5%
sok_9.in	76	>601	2	3	0,3%	0,5%
průměr	165	>235,4	4,8	4,6	2,0%	2,0%

Nevýhody obousměrného p.

- Ne vždy lze použít.
  - Musíme znát cílový stav (stavy).
  - Musíme vždy umět najít předchůdce.
- Všechny algoritmy jsou poměrně složité.
  - Je obtížné je implementovat.
  - Velká šance, že někde bude chyba.
- Není zaručeno, že bude obousměrné prohledávání rychlejší.
  - Nevhodná úloha.
  - Špatná implementace. (Pomalé datové struktury…)
- Je potřeba hodně paměti.
- Většina „obousměrných“ algoritmů vyžaduje konzistentní heuristiky.

Literatura

- **Pohl, I**.: Bi-directional Search. *Machine Intelligence*, ročník 6, 1971: s. 127-140
- **de Champeaux, D.; Sint**, L.: An Improved Bidirectional Heuristic Search Algorithm. *Journal of the Association for Computing Machinery*, ročník 24, č. 2,1977: s. 177-191
- **Politowski, G.; Pohl, I**.: D-node retargeting in bidirectional heuristic search. *In Proceedings of the Fourth National Conference on Articial Intelligence*, 1984
- **Kwa, J. B**.: BS*: An Admissible Bidirectional Staged Heuristic Search Algorithm. *Articial Intelligence*, ročník 38, 1989: s. 95-109.
- **Ikeda, T.; Hsu, M.-Y.; Imai, H.; aj**.: A fast algorithm for finding better routes by AI search techniques. In *1994 Vehicle Navigation and Information Systems Conference Proceedings*, 1994, s. 291-296.
- **Dillenburg, J. F.; Nelson**, P. C.: Perimeter search. *Articial Intelligence*, ročník 65, leden 1994: s. 165-178.
- **Manzini, G**.: BIDA*: an improved perimeter search algorithm. *Articial Intelligence*, ročník 75, 1995: s. 347-360.
- **Kaindl, H.; Kainz**, G.: Bidirectional heuristic search reconsidered. *Journal of Articial Intelligence Research*, ročník 7, 1997: s. 283-317.
- **Pijls, W.; Post, H.:** A new bidirectional search algorithm with shortened postprocessing. *European Journal of Operational Research*, ročník 198, č. 2, 2009: s.363-369.

Konec