Algoritmy pro pathfinding

Lukáš Bajer
MFF UK
bajeluk (zavináč) matfyz.cz
XII.2005

Přehled

1. **Pathfinding, jednoduché algoritmy**
2. A*
3. Hierarchické plánování (HPA*, PRA*)
4. Incremental heuristic search methods

Pathfinding

„Definice“. Pathfinding je schopnost najít pro daný objekt *dobrou cestu* z jednoho místa na druhé.

Dobrá cesta

- nejkratší
- nejlevnější
- nejjednodušší (např. v dopravě)
- nejbezpečnější
- ...

Přidružené problémy

- nalezení cest ze všech míst prostoru do cíle
- hledání cesty v (předem) neznámém prostoru
- mapování světa
- pohyb skupin postav
- steering, vyhýbání překážkám

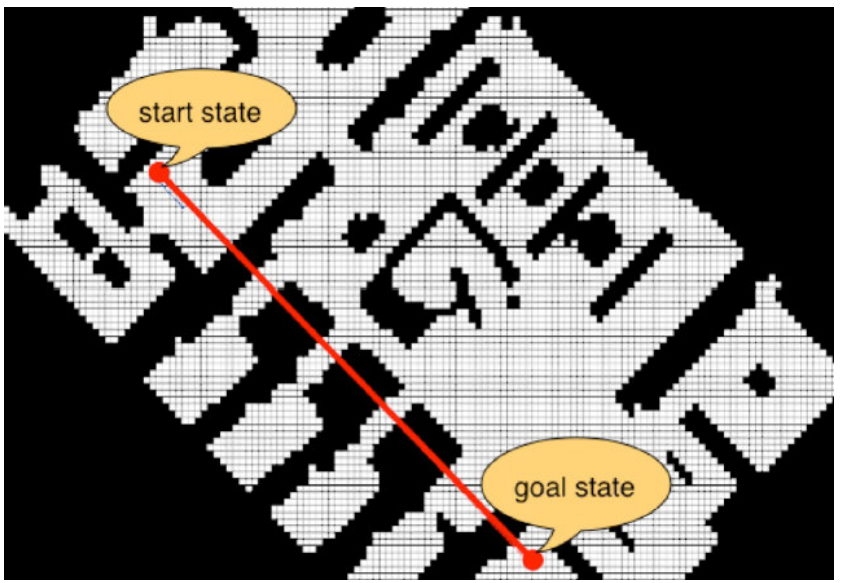
Pathfinding

Aplikace

- počítačové hry (strategie, akční, RPG,...)
- filmová animace
- doprava
- robotika

Problém

Najděte cestu pro autonomního agenta ze zadané startovní pozice do cíle v dopředu známém prostředí.



Implementace prostředí

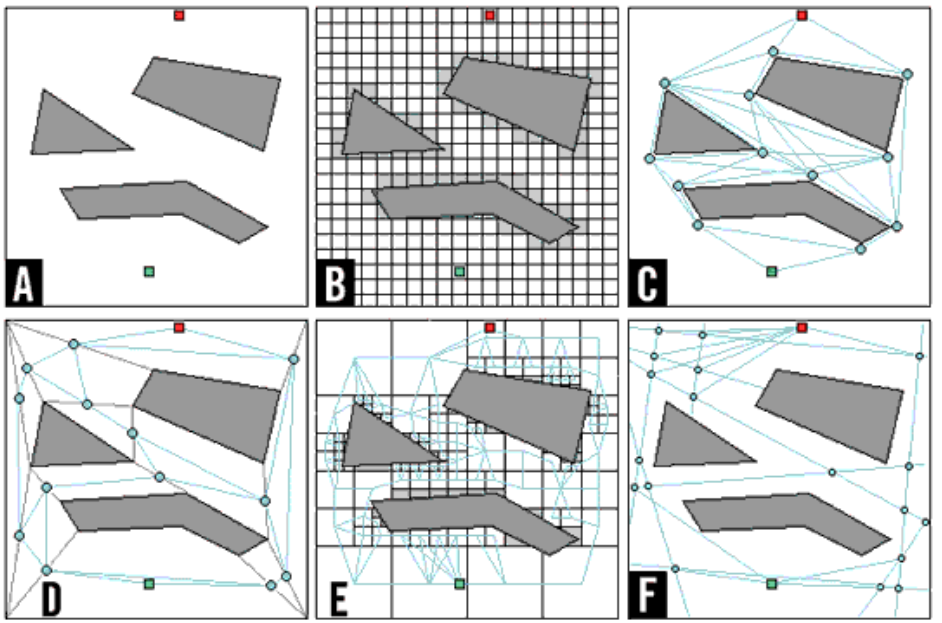
- Dlaždice
  - prostor je překryt čtvercovou sítí
  - dlaždice velikostí odpovídají nejmenší postavě

  + překážky a postavy lze snadno lokalizovat a tak se jim vyhnout
  + jednoduchost implementace
  + snadno jdou zohlednit různé druhy terénů
  + možnost hierarchického plánování

  - typicky velký prohledávaný prostor
  - nevhodné pro 3D svět

Implementace prostředí

- Dlaždice [B]
  - druhy dlaždic:
    - osmisměrné
    - čtyřsměrné
    - šestiúhelníkové
    - texes
- Body viditelnosti [C]
- Triangulace
- Konvexní polygony (NavMesh) [D]
- Potenční pole



Jednoduchoučké algoritmy

Vyhýbání se překážkám po cestě

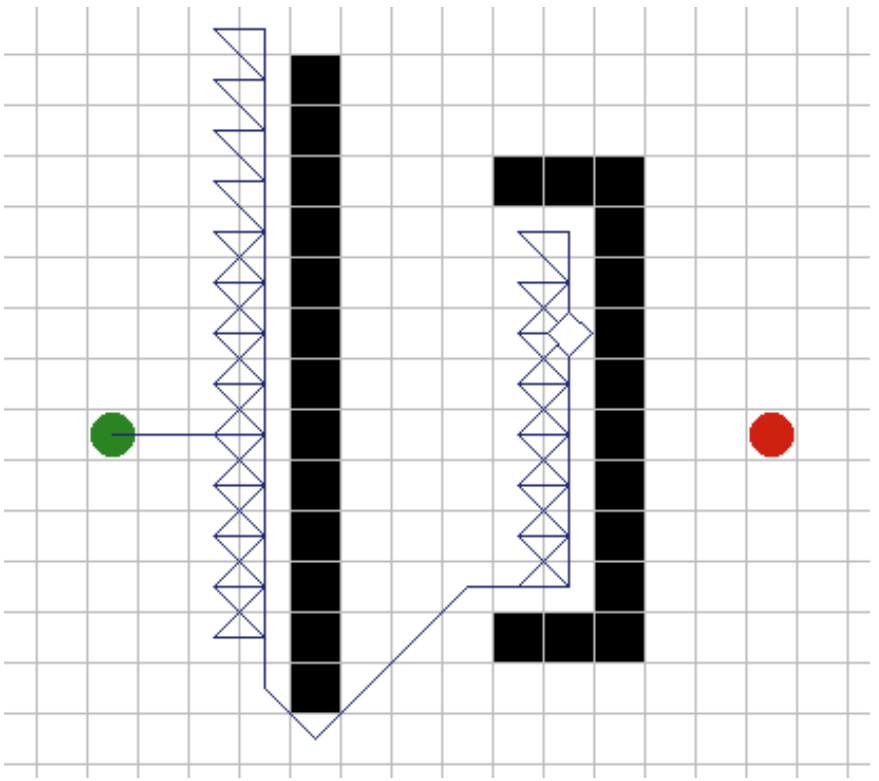
```
dokud nejsi v cíli:
  vyber dlaždici směrem k cíli
  když je dlaždice volná
    jdi na ni
  jinak
    vyber "jinou" dlaždici podle své "strategie"
```

Možné strategie

- pohyb náhodným směrem
- těsné obcházení překážek
- "robustní" obcházení překážek

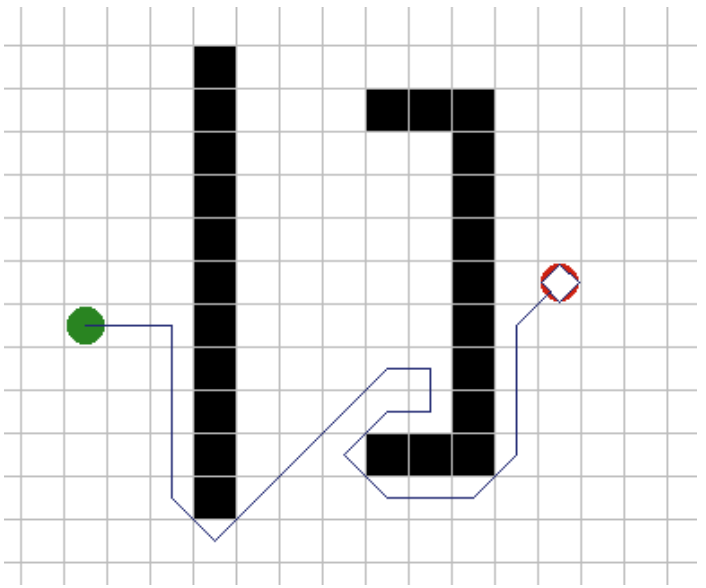
Jednoduchoučké algoritmy

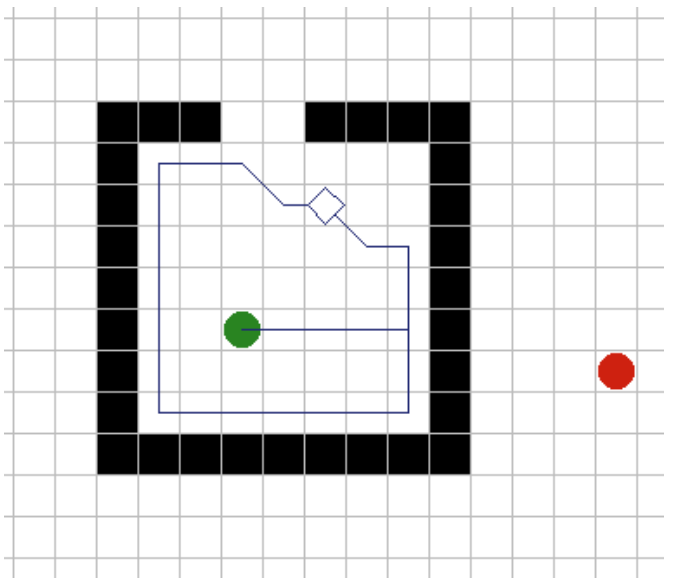
Pohyb náhodným směrem



Jednoduchoučké algoritmy

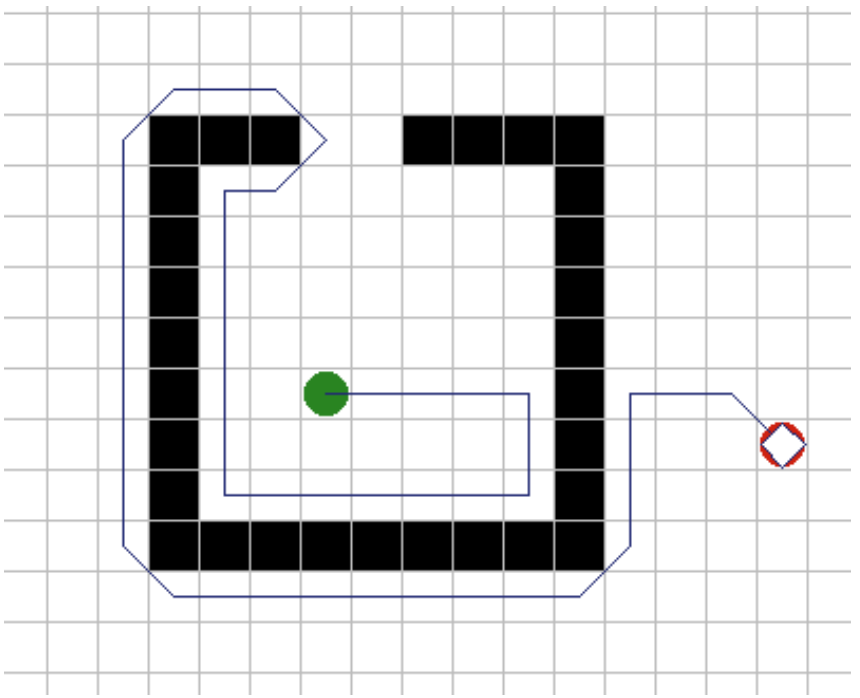
Těsné obcházení překážek





Jednoduchoučké algoritmy

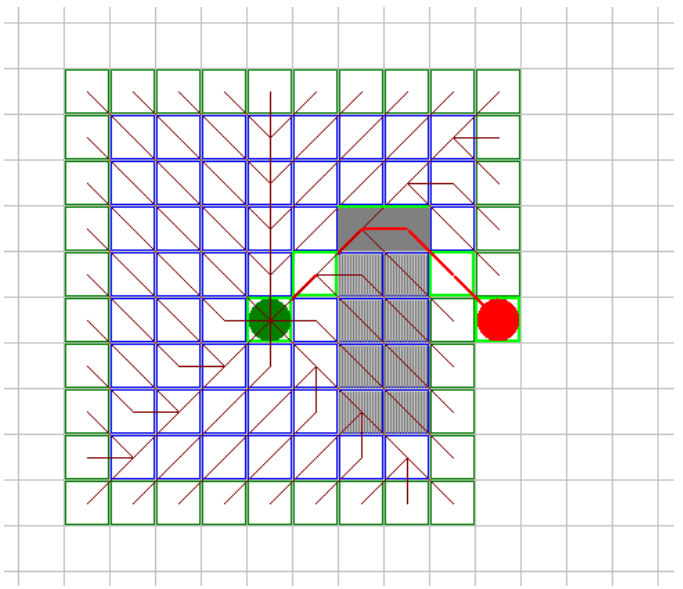
„Robustní“ obcházení překážek



Grafové algoritmy

Prohledávání do šířky

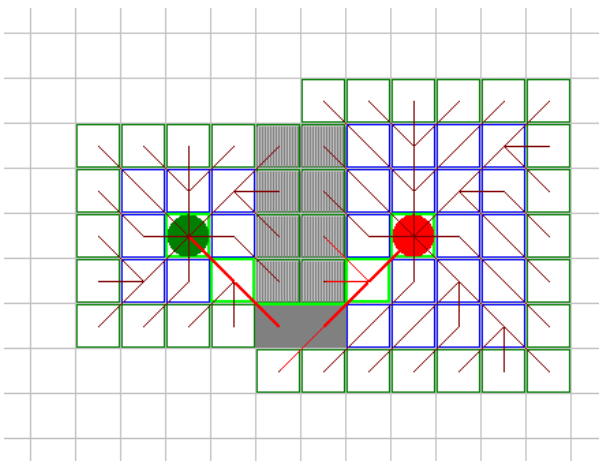
- (breadth-first search)
- Open list: FIFO, fronta
- postup všemy směry stejně rychle



Grafové algoritmy

Dvousměrné prohledávání do šířky

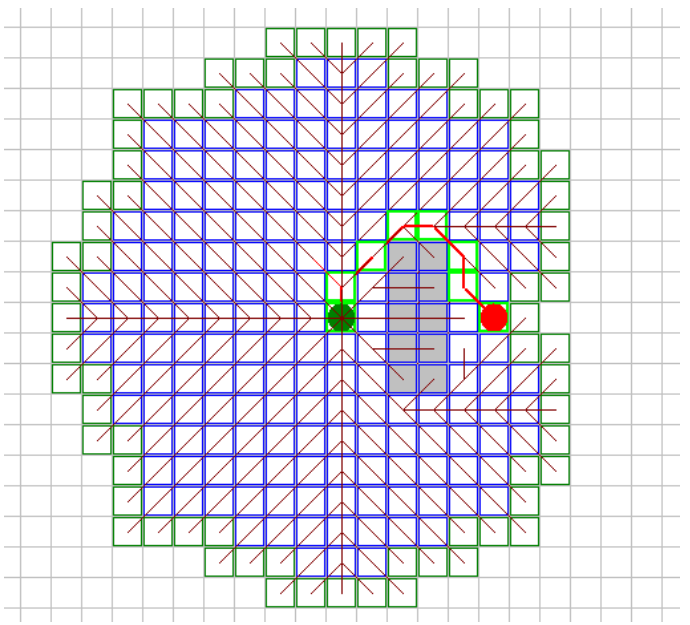
- může ušetřit prohledávání (asi 1/2)



Grafové algoritmy

Dijkstrův algoritmus

- Open list: prioritní fronta tříděná podle vzdálenosti ze startu
- bere v úvahu ceny hran

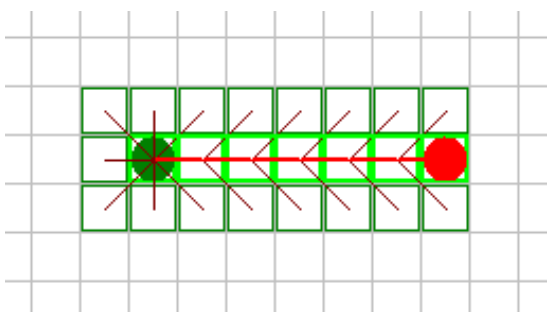


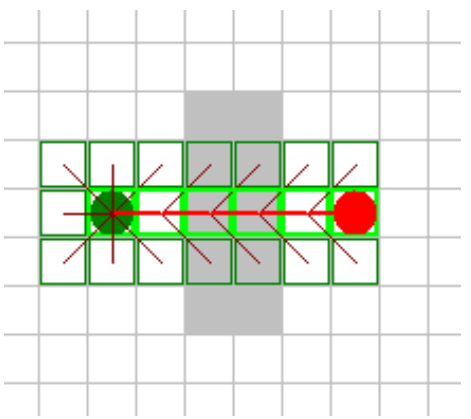
[Dijkstra, E.W. (1959)]

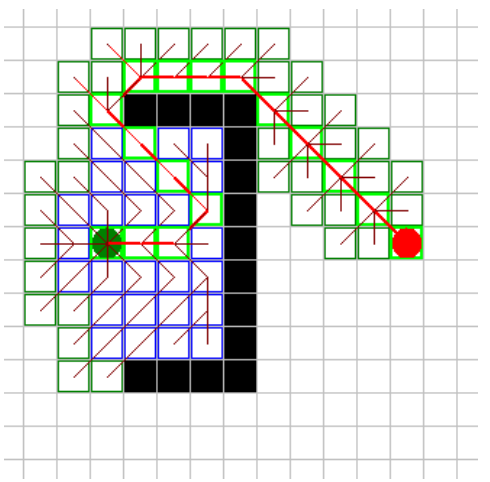
Grafové algoritmy

Best-first search

- používá heuristiku
- podobný Dijkstrovu algoritmu, vrcholy ale třídí podle odhadu do cíle
- nejrychlejší v jednoduchém terénu
- není zaručeno nalezení nejkratší cesty







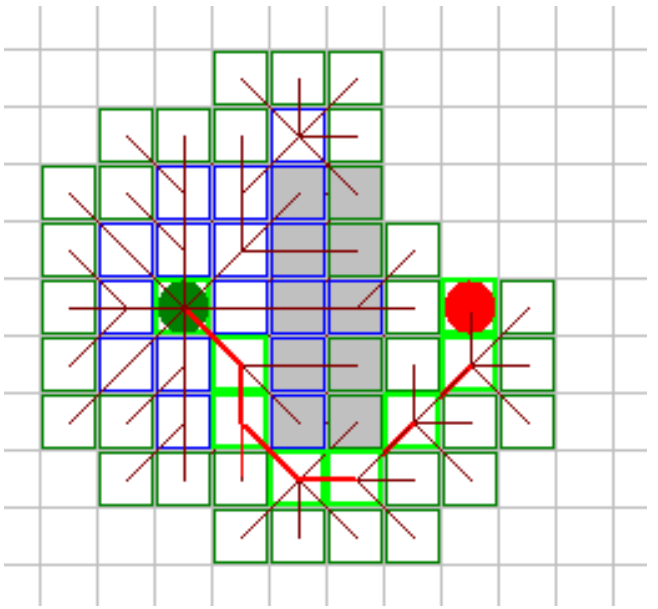
Přehled

1. Pathfinding, jednoduché algoritmy
2. **A***
3. Hierarchické plánování (HPA*, PRA*)
4. Incremental heuristic search methods

Algoritmus s hvězdičkou – A*

„AI research often focuses in a direction that is less useful for games. A is the most successful technique that AI research has come up with—and nearly the only one applied in computer games.“* (A. Nareyek, 2004)

- hledá za pomoci heuristiky
- zkouší vždy nejslibnější neprobraný stav



[Hart, P., Nilsson, N., & Raphael, B. (1968)]

Algoritmus s hvězdičkou – A*

Ohodnocení a heurisitka

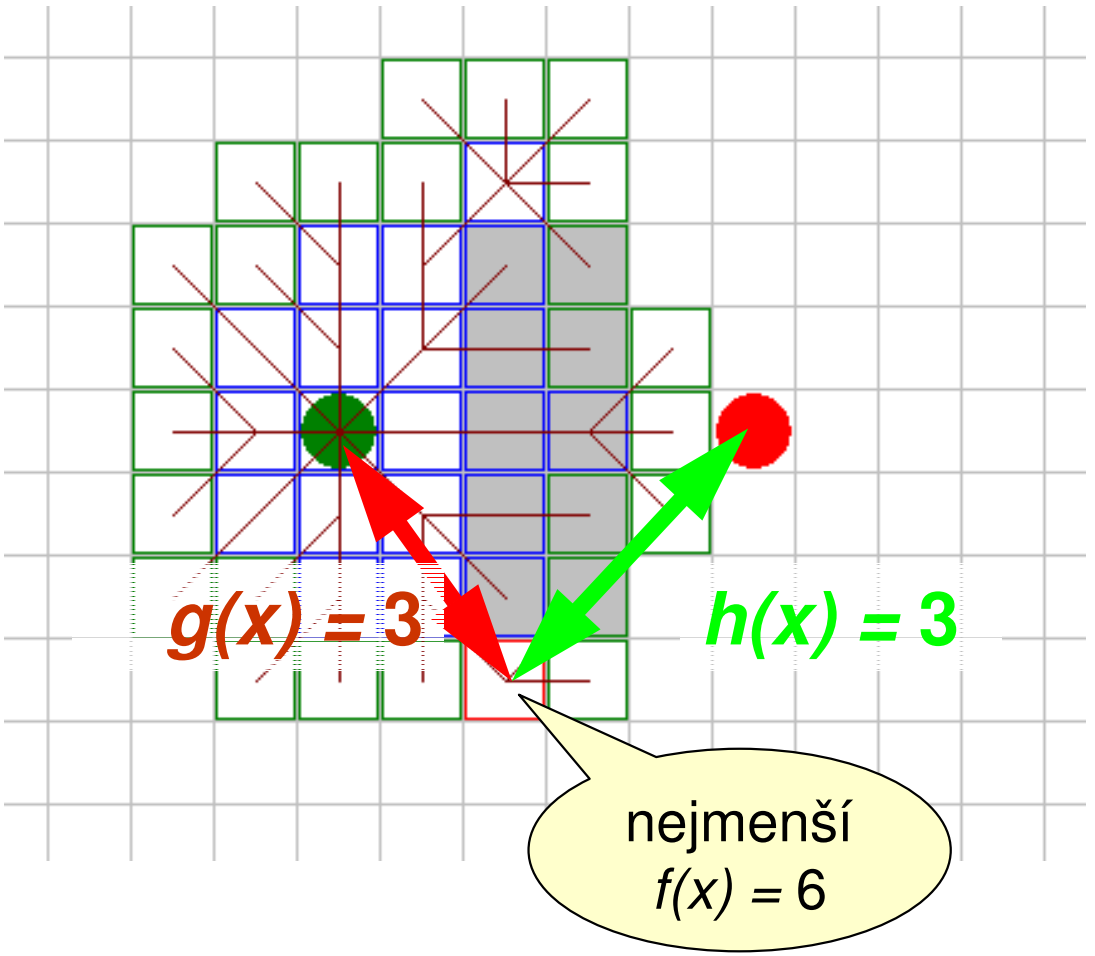
- Pro každý navštívený vrchol *x* se spočítá ohodnocení nejlepší cesty *f(x)*, která přes něj vede podle (složitého ;) vzorce

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

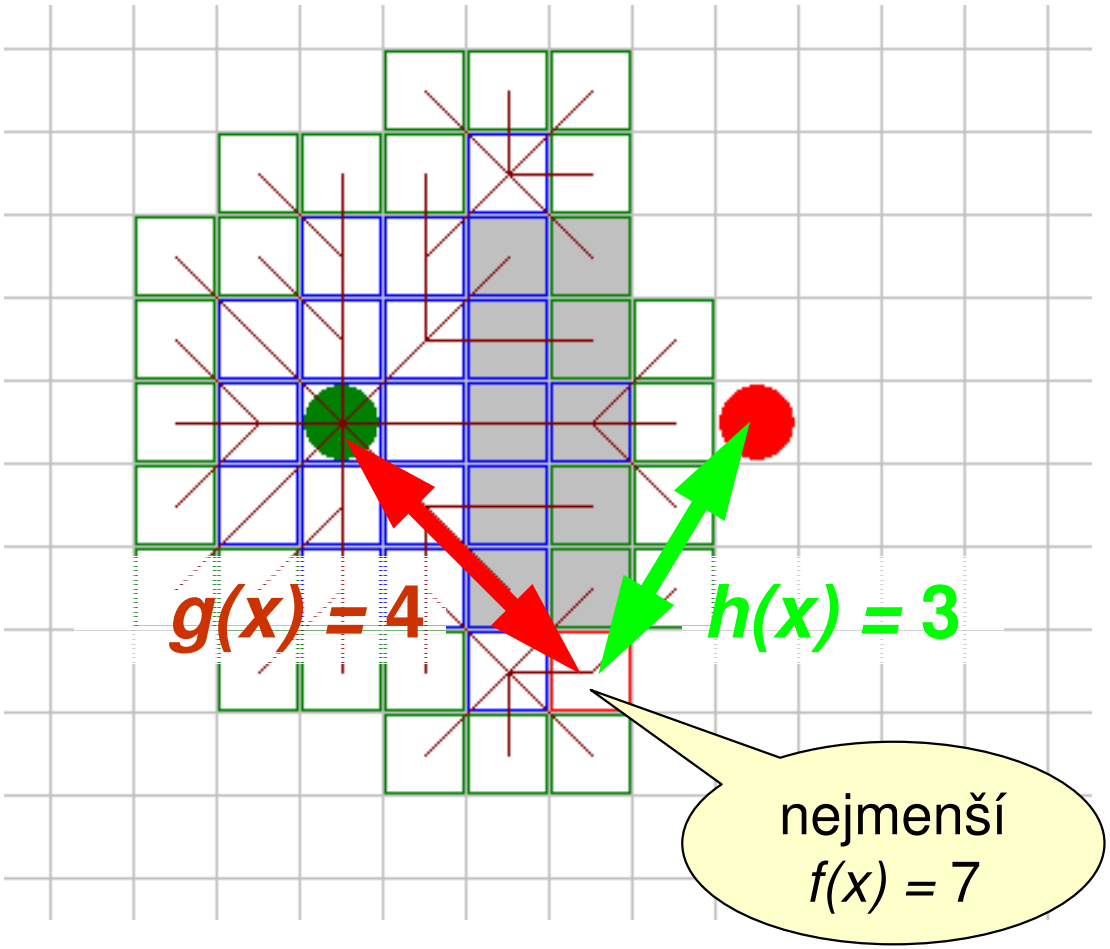
kde

- *g(x)* je součet ohodnocení nejkratší cesty ze startu přes dosud navštívené (a probrané) vrcholy do *x* a
- *h(x)* je odhad délky cesty z *x* do cílového stavu
- pokud je odhad vždy menší nebo rovný skutečné vzdálenosti do cíle, heuristika je "přípustná" a algoritmus vždy najde nejkratší cestu.

Algoritmus s hvězdičkou – A*



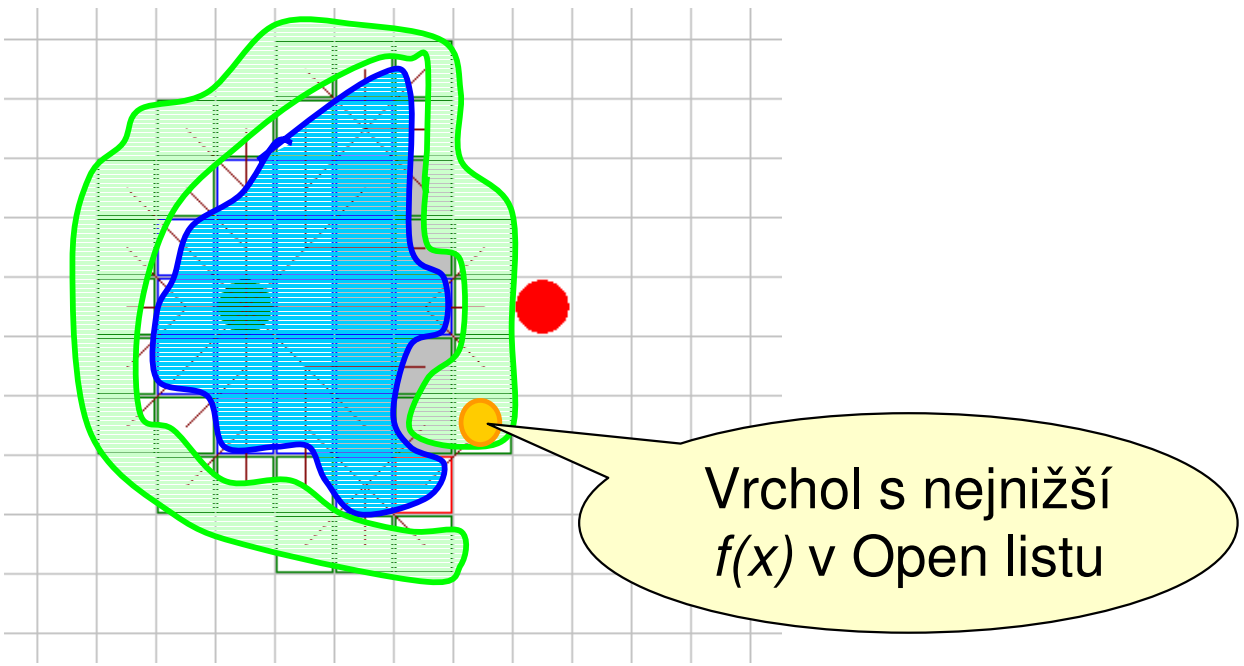
Algoritmus s hvězdičkou – A*



Algoritmus s hvězdičkou – A*

Datové struktury

- **Open**, seznam navštívených vrcholů, které je potřeba probrat
- **Closed**, seznam již probraných vrcholů



Algoritmus s hvězdičkou – A*

Pseudokód

```
Open.insert(start, 0)
while (Open.not_empty())
  (s, fs) = Open.pop()
  if (s == goal) return „úspěch“
  Closed.insert(s)
  for (all x in Succ(s) && x not in Closed)
    if (! Open.find(x, open_fx))
      Open.insert(x, f(x))
    else
      if (f(x) < open_fx)
        Open.update(x, f(x))
return „neúspěch“
```

Open.insert(x, fx)
vloží do **Open** listu vrchol *x* s ohodnocením *fx*

Open.pop()
vrátí vrchol s nejnižším ohodnocením

Open.find(x, fx)
vrátí *true* a v parametru *fx* ohodnocení, pokud *x* je v **Open**, jinak vrátí *false*

Open.update(x, fx)
opraví ohodnocení u vrcholu *x* v **Open** listu

A* – vlastnosti

+ cesta je **vždy** nalezená a je **nejkratší** možná
+ **volnost v ohodnocování** polí – nejrůznější druhy zvýhodnění/penalizace terénu
+ použití s libovolnou reprezentací terénu
+ jednoduchost implementace

– **počet vrcholů** v **Open** nebo Closed listu může být v řádu stovek nebo tisíců
– velká ***First move delay*** – cesta se musí spočítat celá najednou
– pokud cesta neexistuje, prohledá se **celý prostor**, což může trvat extrémně dlouho
– pokud se změní prostředí, je nutné **celou** cestu **přeplánovat**
– není možné dopředu počítat s **dalšími postavami**

A* – optimalizace (heuristika)

Na dlaždicových mapách existují jednoduché způsoby jak zajistit, aby byla heuristika přípustná:

- **maximová metrika** (pro 8–směrné dlaždice) ($h(x) = \max(dx, dy)$)
- **Manhattonská metrika** (pro 4–směrné dlaždice) ($h(x) = dx + dy$)
- **Euklidovská vzdálenost** ($h(x) = \text{sqrt}(dx\text{^}2 + dy\text{^}2)$)

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

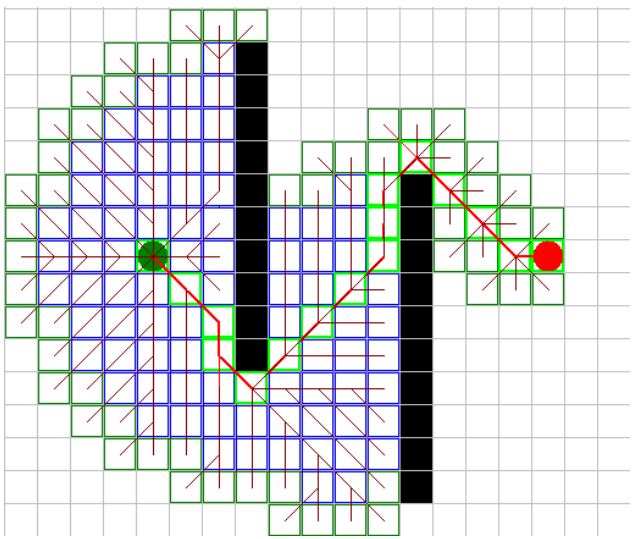
Extrémní případy heuristiky

- **Dijkstrův** algoritmus: $h(x) == 0$
  - většinou prohledává zbytečně mnoho
- **Best-first** algoritmus: $g(x) == 0$
  - často nalézá špatné cesty

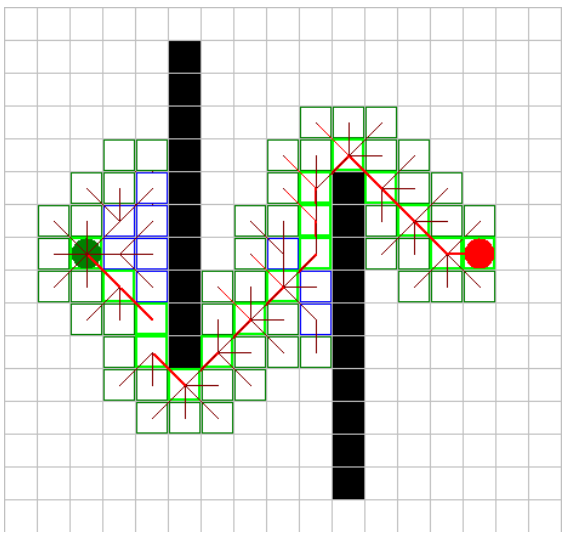
A* – optimalizace (heuristika)

Nadhodnocená (tzn. nepřípustná) heuristika tlačí prohledávání blíženi k cíli.

Pokud cesta nevyžaduje backtracking, dosahuje lepších výsledků.



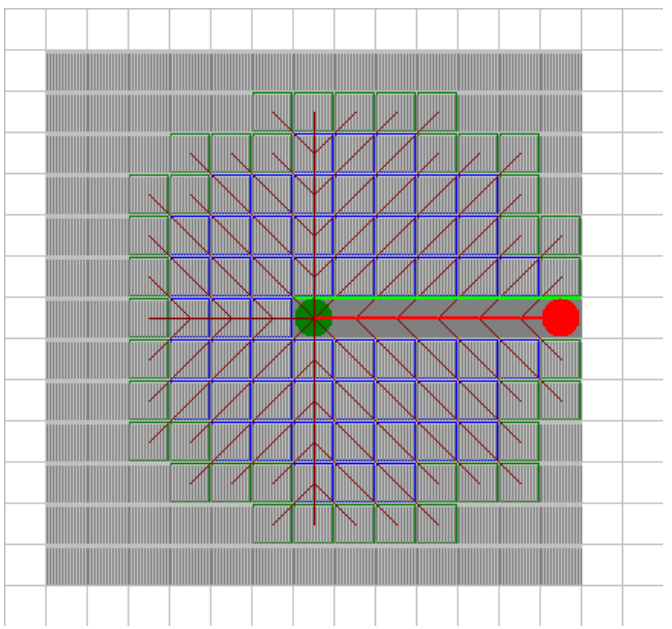
f(x) = g(x) + h(x)



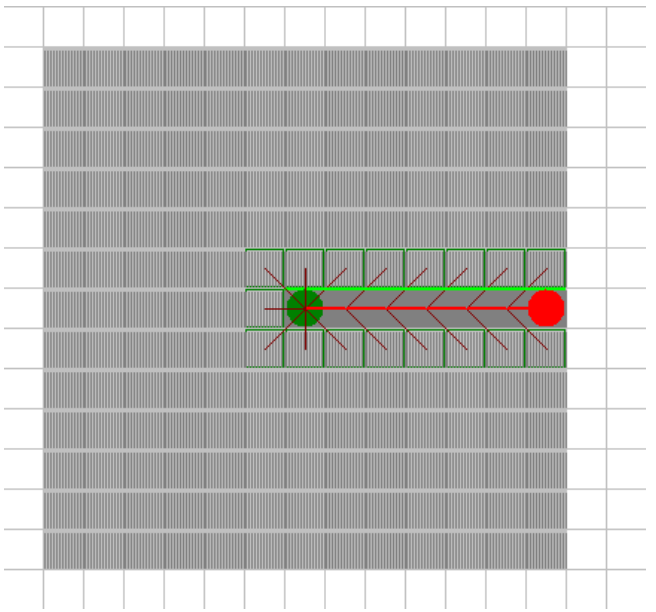
*f(x) = g(x) + 5 * h(x)*

A* – optimalizace (heuristika)

Přesná heuristika dává v „náročném“ terénu značně lepší výsledky než podhodnocená heuristika



$c(x, y) = 5,\ h(x, y) = 1$



$c(x, y) = 5,\ h(x, y) = 5$

Přehled

1. Pathfinding, jednoduché algoritmy
2. A*
3. **Hierarchické plánování (HPA*, PRA*)**
4. Incremental heuristic search methods

Hierarchické plánování

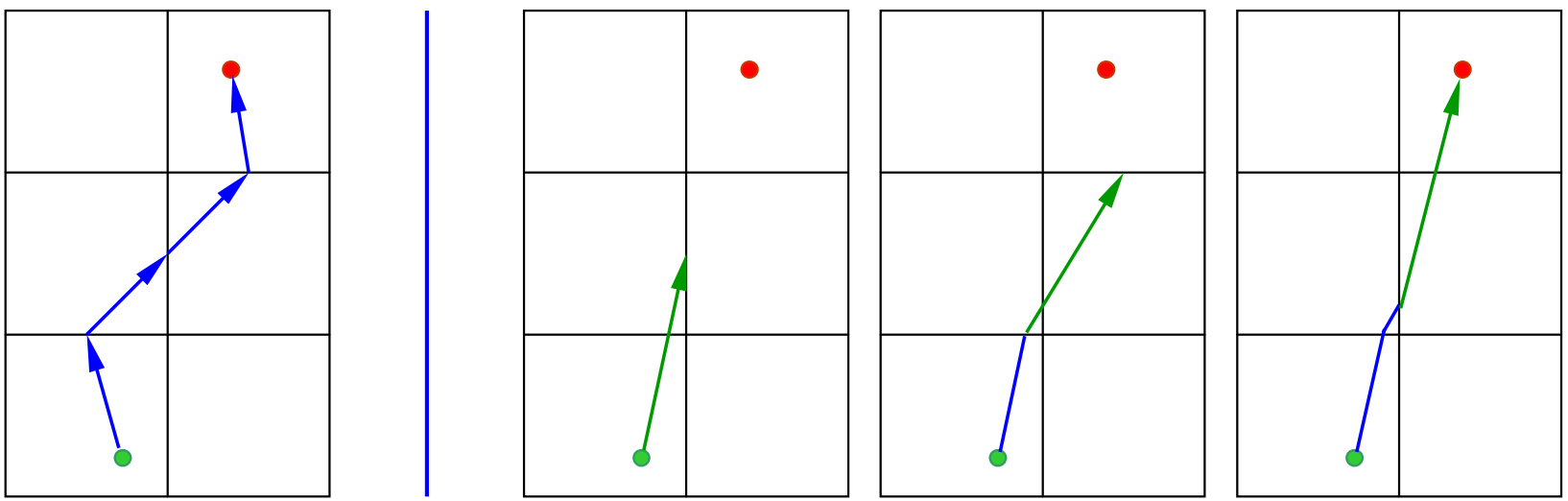
Vychází z přístupu podobného lidskému chování. Nejdříve je potřeba najít cestu "zhruba" a pak postupně zjemňovat.

+ výrazná **úspora zdrojů**
+ rozdíl výkonu roste s velikostí původního prostoru

– výsledná cesta **nemusí být optimální**
– složitější implementace
– může být závislé na konkrétní mapě, může být vyžadována podpora od návrháře konkrétní mapy (rozdělení na místnosti, souvislé oblasti stejného terénu atd.)

Hledání za roh

- cesta se plánuje mezi středy „průchodů“ mezi jednotlivými oblastmi
- vyhlazení prodloužením prohledávání o oblast dál



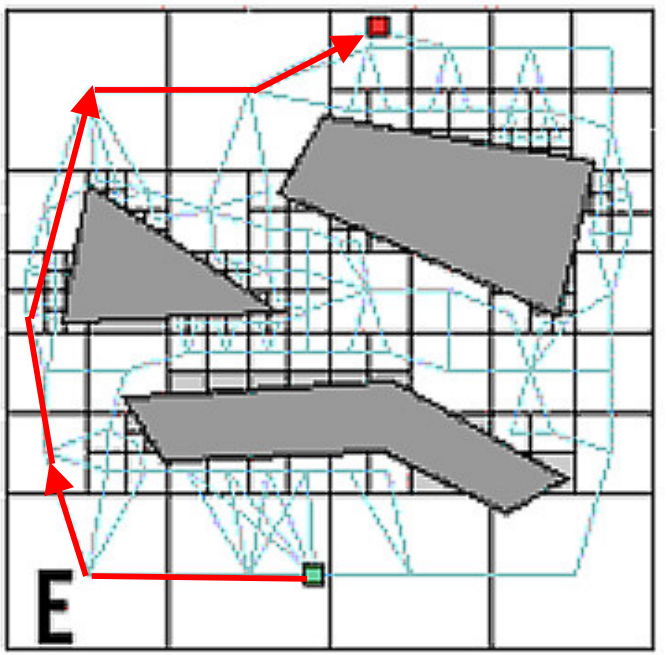
[Rabin, S. (2000)]

Quadtrees

Cesty se plánují mezi středy jednotlivých čtverců.

+ rychlé
+ nenáročné na paměť
+ jednoduchá dekompozice

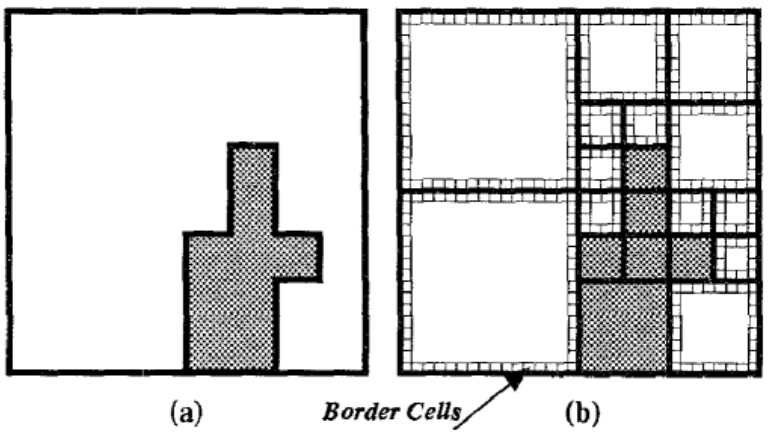
– výsledná cesta bývá poměrně neoptimální



Framed quadtrees

Každý z výsledných čtverců "obsahuje" i všechny hraniční dlaždice z původní mapy, cesta se plánuje mezi nimi.

+ znatelné zlepšení kvality cesty oproti normálním quadtrees
– větší spotřeba paměti

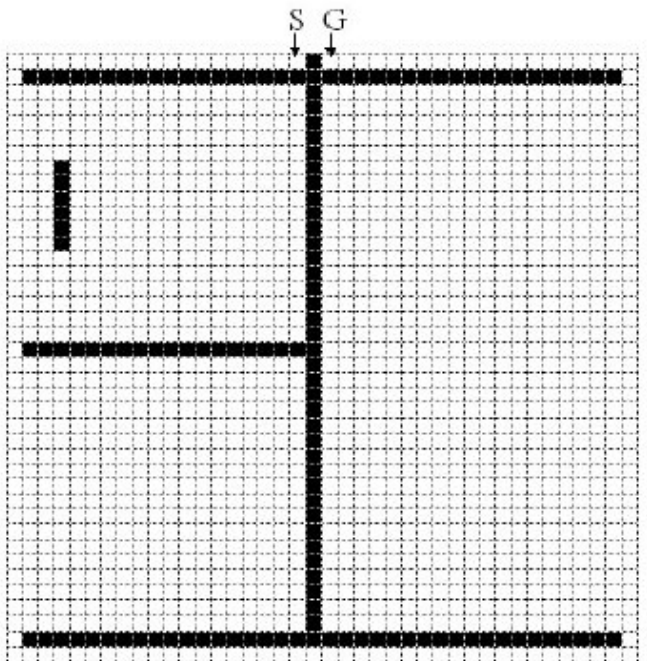


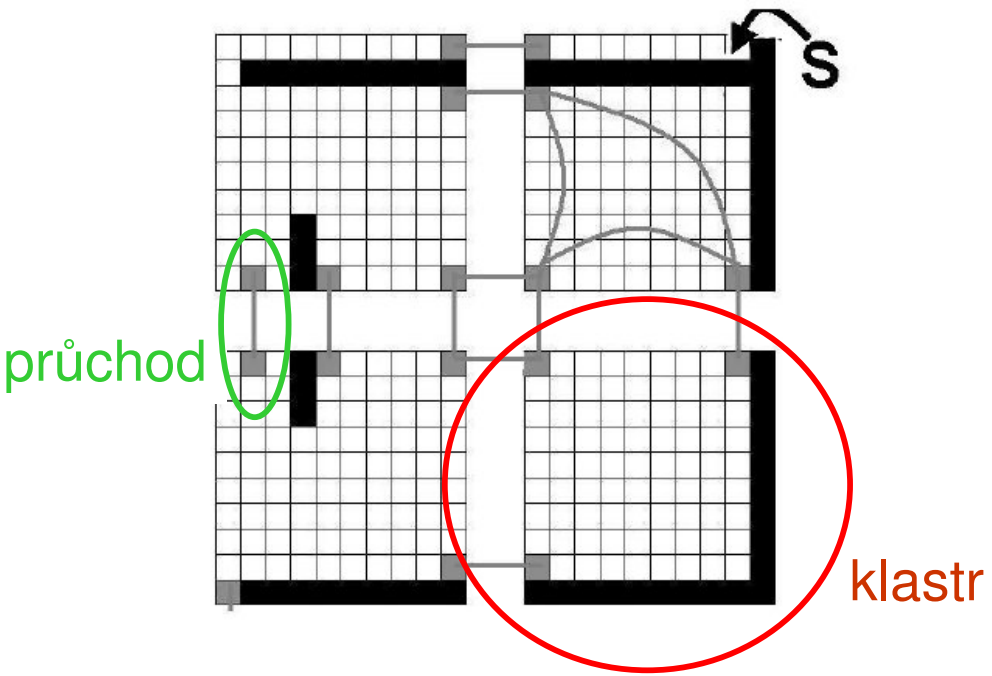
[D. Z. Chen, R. J. Szczerba, and J. J. Urhan Jr. (1995)]

Hierarchical Path–finding A*

rozdělení na *klastry*, cca 10 x 10 políček

určení *průchodů* mezi klastry (vybraná políčka, která sousedí s volnými políčky sousedních klastrů)

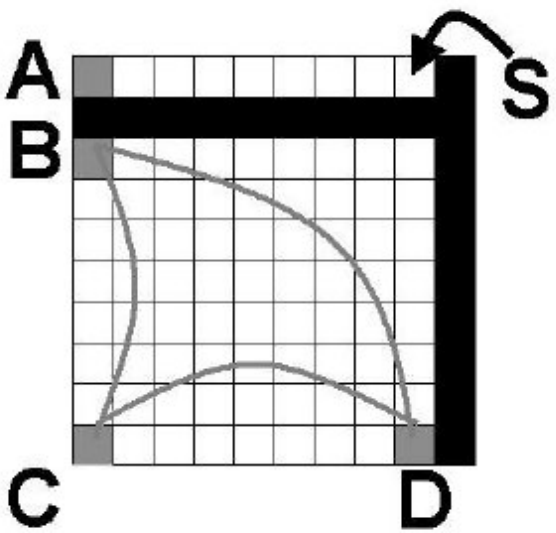


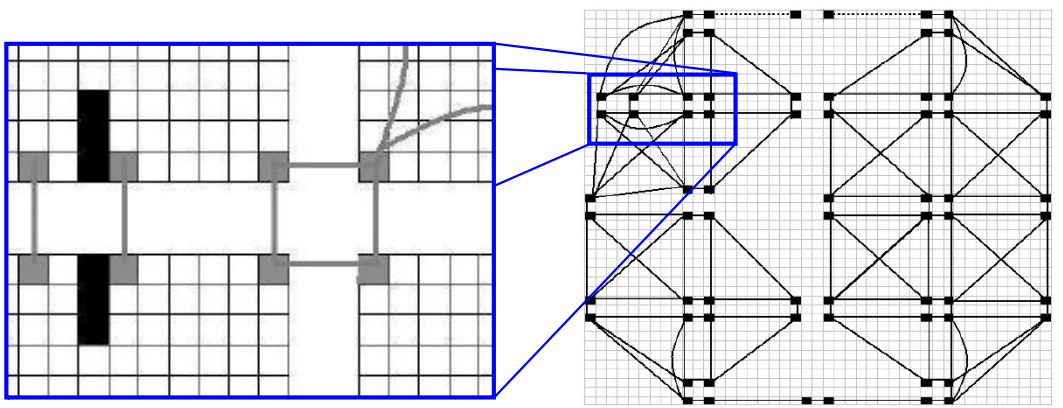


Hierarchical Path–finding A*

vytvoření *abstraktního grafu*

- hrany mezi průchody v rámci jednoho klastru (s uložením délky cest)
- hrany mezi políčky v průchodech sousedních klastrů



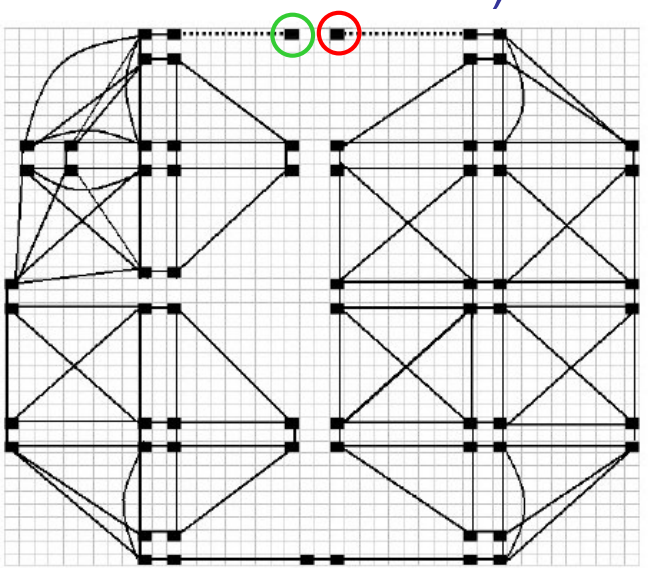


[Botea, A., Müller, M. & Schaeffer, J. (2004)]

Hierarchical Path–finding A*

Algoritumus

1. Začleň *start* a *cíl* do *abstraktního grafu*, což vyžaduje rožšíření grafu o hrany mezi přidávaným políčkem (*startem* resp. *cílem*) a průchody do sousedních klastrů (vyznačeno čárkovaně).

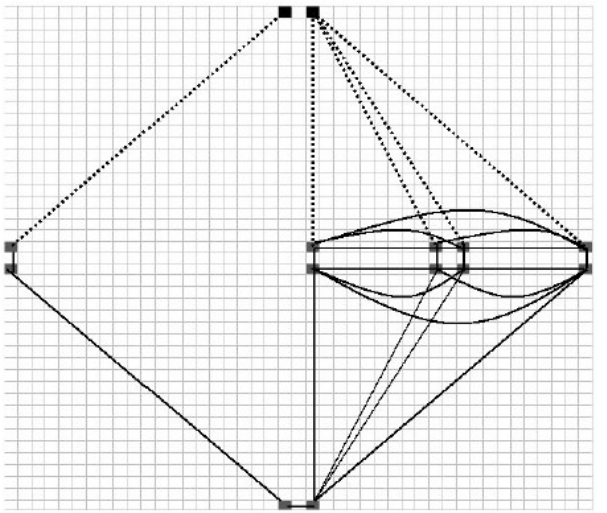


Hierarchical Path–finding A*

Algoritumus (pokračování)

2. Aplikuj standardní A* algorimus na *abstraktní graf* (rozšířený o *start* a *cíl*).
3. Zjemni abstraktní cestu na posloupnost pohybů po původní mapě. To je možné dělat postupně (cestou).
4. (nepovinně) Vyhlaď cestu získanou v bodě 3.

Hierarchical Path–finding A*



+ lze rozšířit do vícevrstvé hierarchie
+ nezávislý na původní mapě
+ desetkrát rychlejší (na reálných mapách)
+ cca 1% odchylka od optimální cesty
+ minimalizuje *first move delay* – cestu je možné dopočítávat postupně

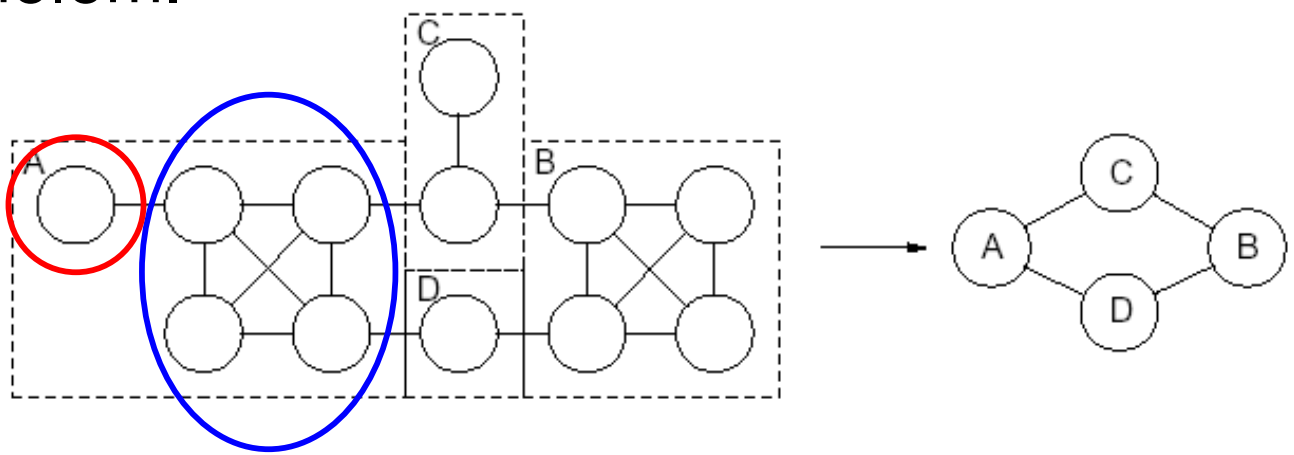
– při skupinovém pathfindingu dostanou všechny jednotky z téže oblasti v podstatě stejné cesty
– vyhlazování fuguje jen na nepříliš hustých mapách

Partial–Refinement A*

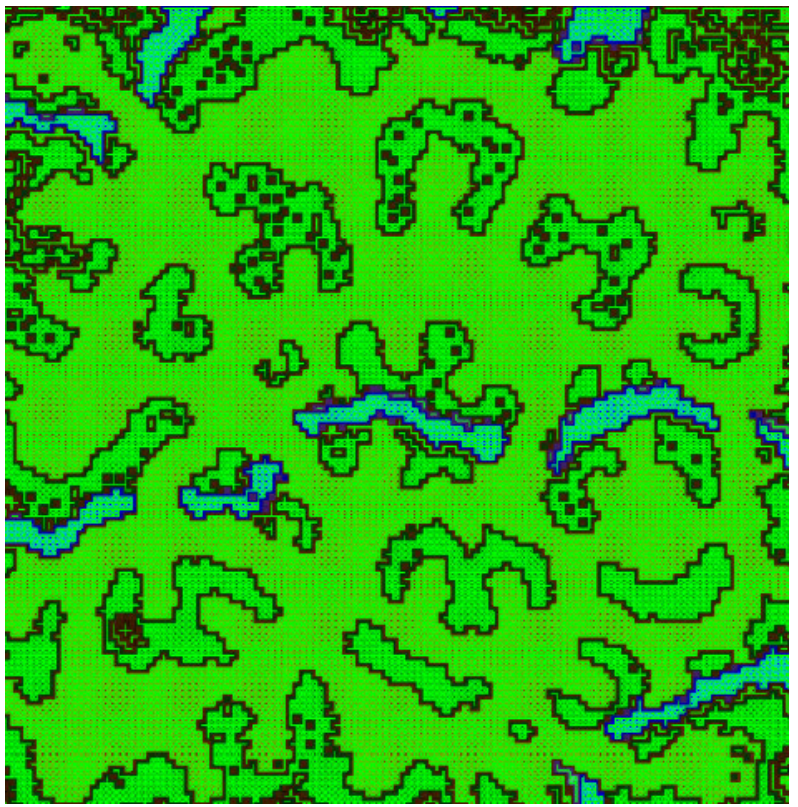
používá hierarchii abstraktních grafů postavených na základě lokálních vlastností grafů nižších vrstev

hledají se *kliky*, které se spolu se sousedícími *sirotky* abstrahují do jednoho vrcholu.

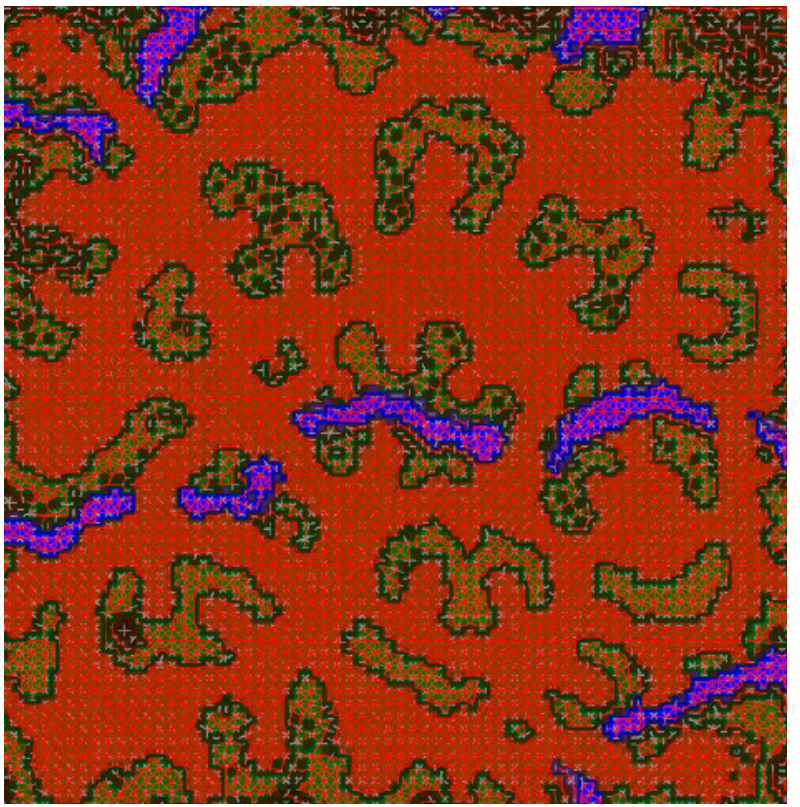
postupně vzniká hierarchie, která končí jediným vrcholem.



Partial–Refinement A*

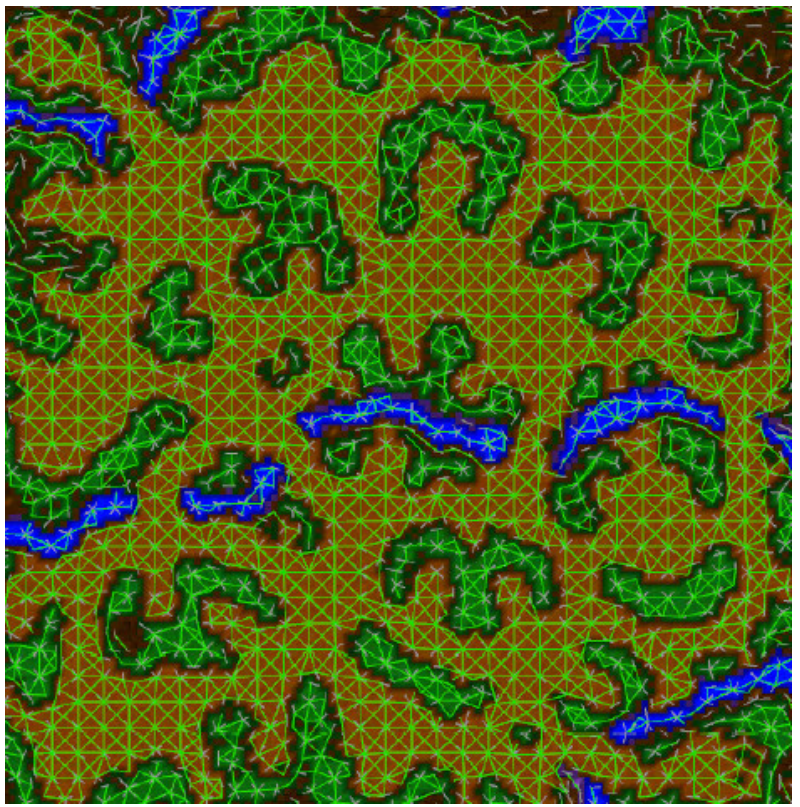


level 0, 16 807 vrcholů

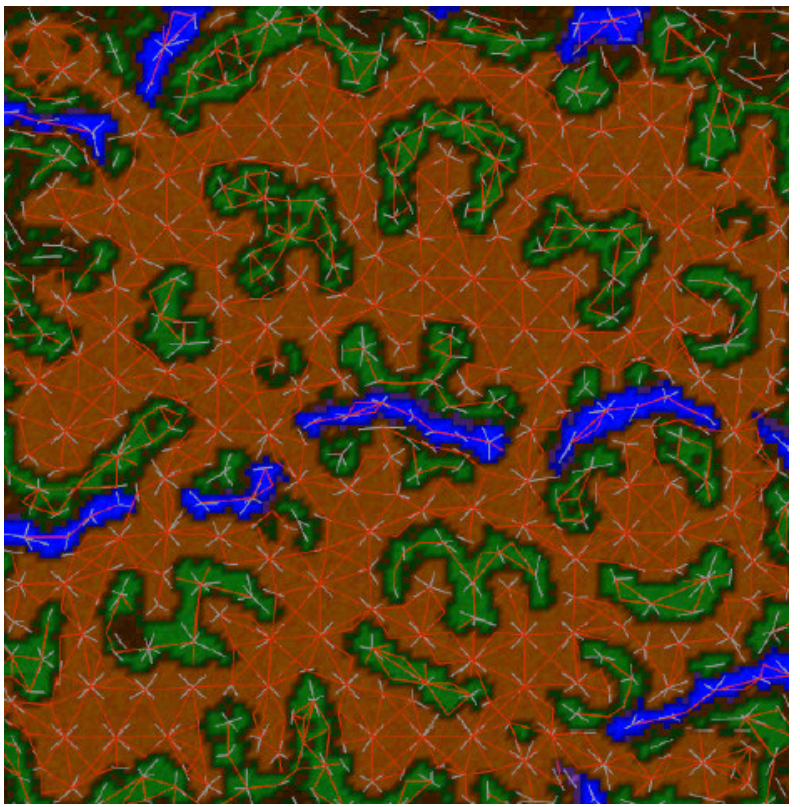


level 1, 5 212 vrcholů

Partial–Refinement A*



level 2, 1 919 vrcholů



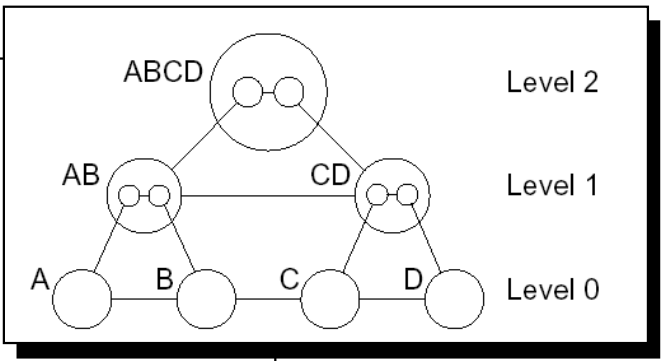
level 3, 771 vrcholů

Partial–Refinement A*

Algoritmus (pseudocode)

```
PRA*(start, goal, k)
  GetAbstractionHierarchy(start, goal)
  s = GetStartLevel(start, goal)
  empty path
  for each level i = s..1
    path = RefinePath(path, start[i], tail(path))
    truncate path to length k
  return path

RefinePath(path, start, goal)
  return aStar(start, goal) subject to nodes in path
```



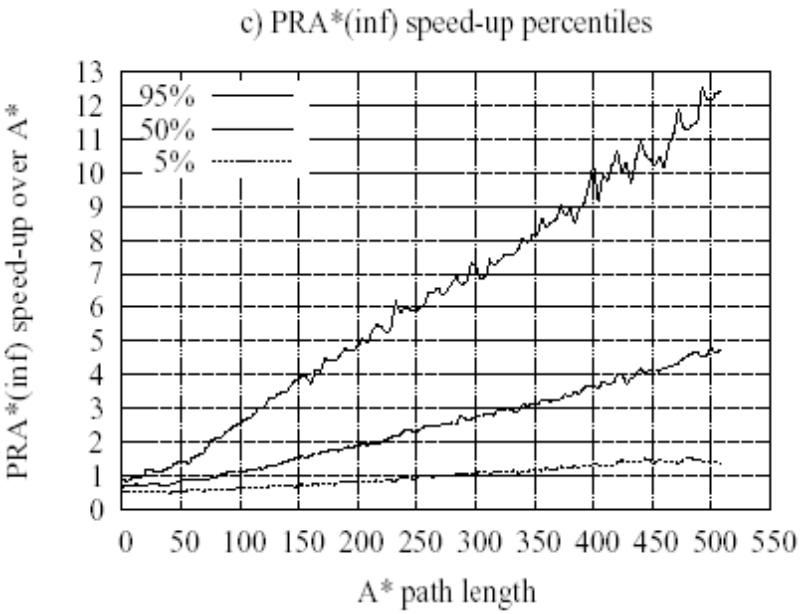
[Sturtevant, N., Buro, M. (2005)]

Partial–Refinement A*

Vlastnosti

+ dokáže pracovat s (víceméně) libovolným grafem
+ cca desetinásobné zrychlení
+ větší možnost parametrizace

– poměrně složitý
– abstrakce fungují špatně pro čtyřsměrné dlaždice



Přehled

1. Pathfinding, jednoduché algoritmy
2. A*
3. Hierarchické plánování (HPA*, PRA*)
4. **Incremental heuristic search methods**

Incremental heuritic search methods

"Incremental heuristic search methods use heuristics to focus their search and reuse information from previous searches to find solutions to series of similar search tasks much faster than is possible by solving each search task from scratch." [S. Koenig (2002)]

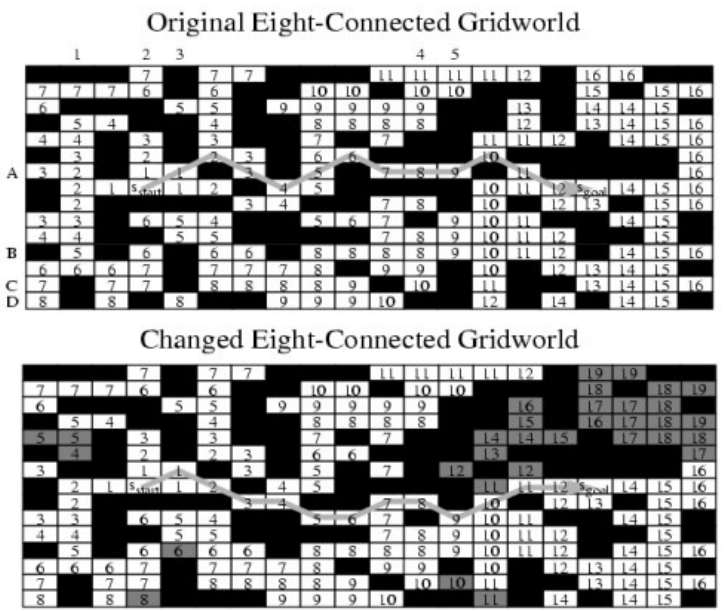
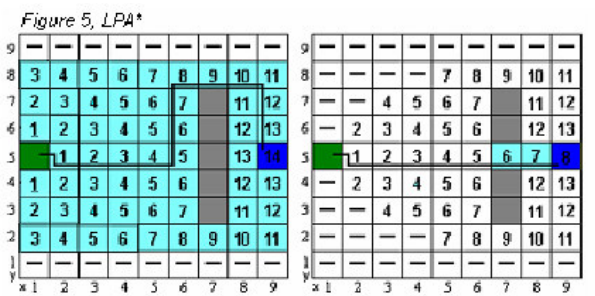


Figure 1: Simple Gridworld

Lifelong Plannig A* (LPA*)

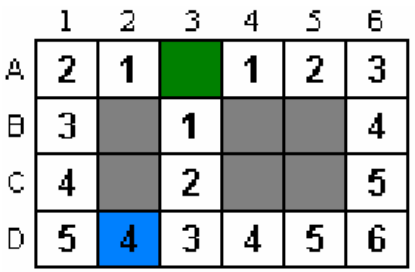
První hledání LPA* je identické s A*. Další však využívá již spočítané hodnoty k dalším hledáním a opravuje jen ty části, které se změnily.

+ Incremental search zaručuje stejné (tj. optimální) výsledeky jako plánování od začátku (tj. normální A*).
+ LPA* je dobře prozkoumaný a matematicky odůvodněný
– je potřeba nadhled nad celou mapou
– změny v mapě přeci jen vyžadují podstatný výpočetní výkon na přepočítání

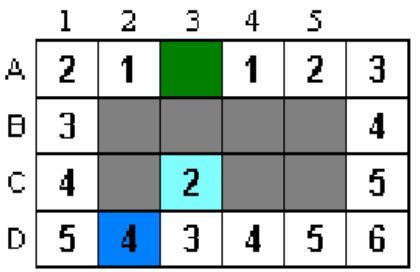


[Koenig, S., and Likhachev, M. (2001)]

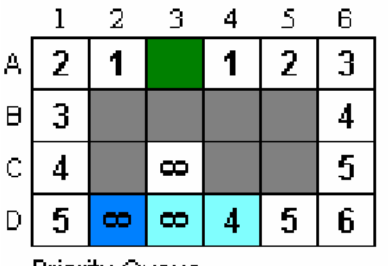
Lifelong Plannig A* (LPA*)



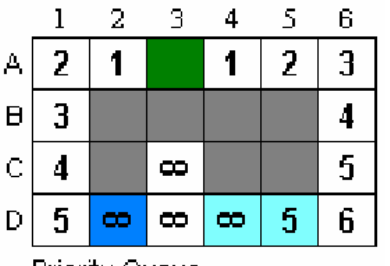
Priority Queue



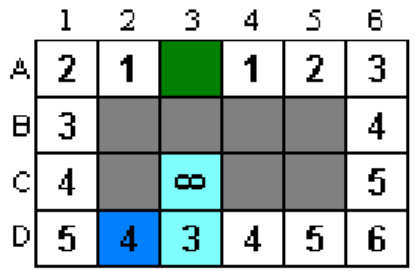
Priority Queue = C3:[4;2]



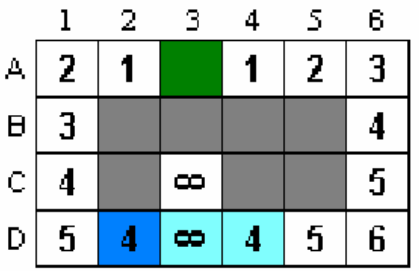
Priority Queue
D4:[6;4], D3:[6;5], D2:[6;6]



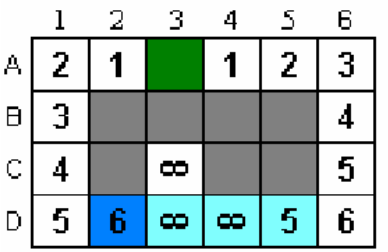
Priority Queue
D2:[6;6],D5: [8;5],D4:[8;6]



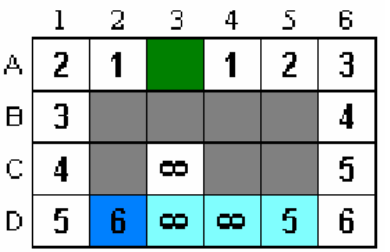
Priority Queue
D3:[4;3], C3:[6;4]



Priority Queue
D2:[4;4], D4:[6;4], D3:[6;5]



Priority Queue
D5: [8;5],D4[8;6], D3: [8,7]



Priority Queue
D5: [8;5], D4: [8;6], D3:[8, 7]

Lifelong Plannig A* (LPA*)

procedure CalculateKey(s)
{01} return $[\min(g(s), rhs(s)) + h(s, s_{goal}); \min(g(s), rhs(s))]$;

procedure Initialize()
{02} $U = \emptyset$;
{03} for all $s \in S$ $rhs(s) = g(s) = \infty$;
{04} $rhs(s_{start}) = 0$;
{05} U.Insert(s_{start}, CalculateKey(s_{start}));

procedure UpdateVertex(u)
{06} if $(u \neq s_{start})$ $rhs(u) = \min_{s' \in Pred(u)}(g(s') + c(s', u))$;
{07} if $(u \in U)$ U.Remove(u);
{08} if $(g(u) \neq rhs(u))$ U.Insert$(u,$ CalculateKey$(u))$;

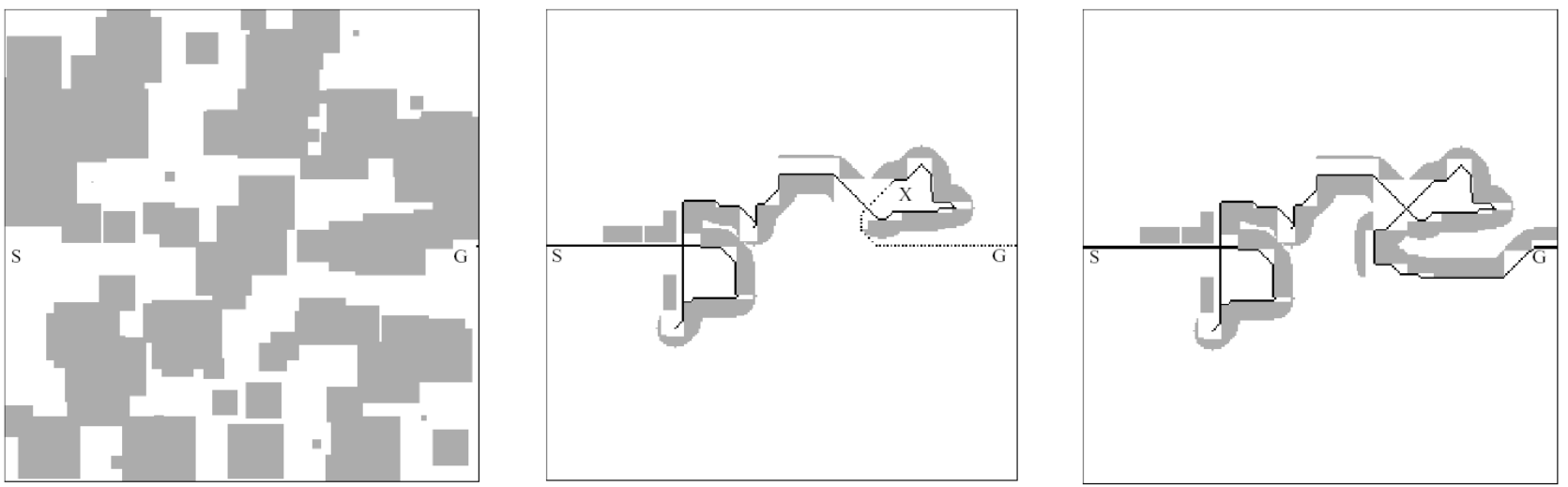
procedure ComputeShortestPath()
{09} while (U.TopKey() $\dot{<}$ CalculateKey(s_{goal}) OR $rhs(s_{goal}) \neq g(s_{goal})$)
{10} $u =$ U.Pop();
{11} if $(g(u) > rhs(u))$
{12} $g(u) = rhs(u)$;
{13} for all $s \in Succ(u)$ UpdateVertex(s);
{14} else
{15} $g(u) = \infty$;
{16} for all $s \in Succ(u) \cup \{u\}$ UpdateVertex(s);

procedure Main()
{17} Initialize();
{18} forever
{19} ComputeShortestPath();
{20} Wait for changes in edge costs;
{21} for all directed edges (u, v) with changed edge costs
{22} Update the edge cost $c(u, v)$;
{23} UpdateVertex(v);

Dynamic A* (D*)

Problém

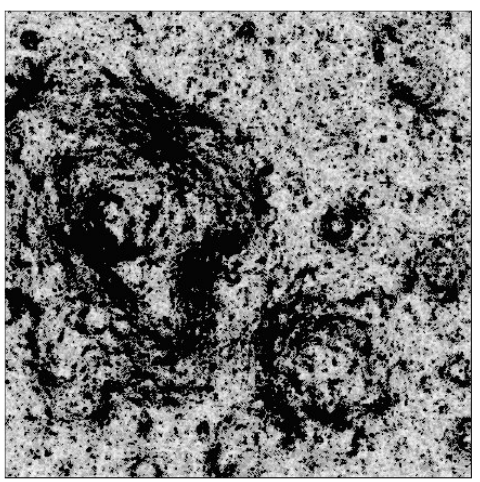
- Agent/robot se pohybuje k cíli a cestou zjišťuje změny v prostředí oproti své mapě. Jesliže takovou zjistí, přepočítá si cestu od své aktuální pozice do cíle.
- Mapa je dopředu buď úplně neznámá, nebo nepřesná.

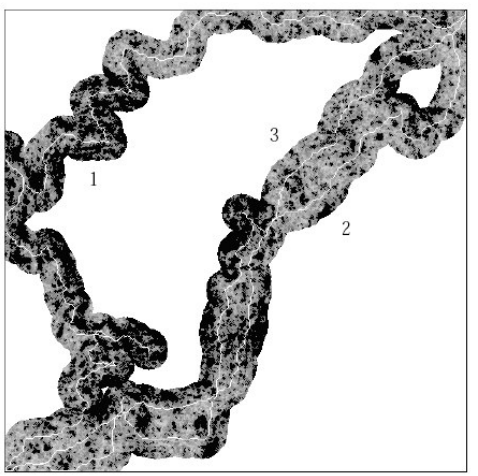


Dynamic A* (D*)

nasazen na reálných robotech (Stentz & Herbert, 1995), mj. v „Mars Rover prototypes“ a „tactical mobile robot prototypes for urban reconnaissance“ (Matthies et al. 2000; Thayer et al. 2000)

není zaručeno, že cesta je nejkratší, je však zaručeno, že je nejkratší možná z dostupných informací





[Stentz, A. (1994)]

D* Lite

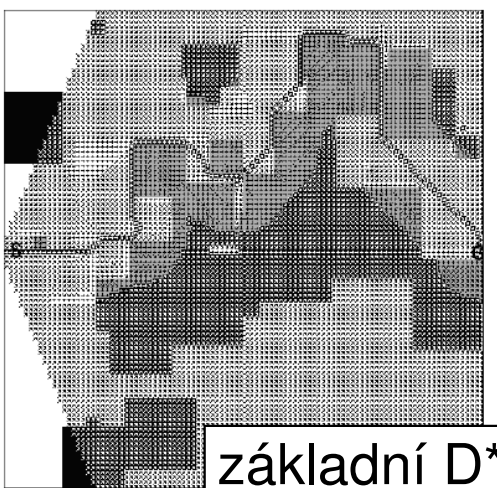
základ je v D* a LPA*

algoritmicky jednodušší než Focussed D* (vylepšená verze D*), ale na stejném principu

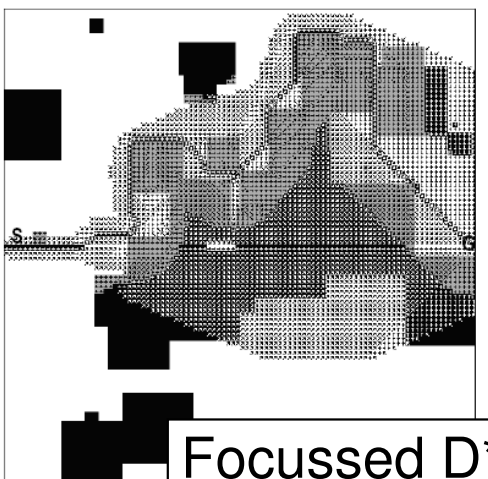
prohledává z cílového vrcholu do aktuální pozice

hledání „zaměřuje“ pomocí heuristiky

používá techniku pro minimalizaci přeuspořádávání prioritní fronty



základní D*



Focussed D*

D* Lite

procedure CalculateKey(s)
{01'} return $[\min(g(s), rhs(s)) + h(s_{start}, s) + k_m; \min(g(s), rhs(s))]$;

procedure Initialize()
{02'} $U = \emptyset$;
{03'} $k_m = 0$;
{04'} for all $s \in S\ rhs(s) = g(s) = \infty$;
{05'} $rhs(s_{goal}) = 0$;
{06'} U.Insert(s_{goal}, CalculateKey (s_{goal}));

procedure UpdateVertex(u)
{07'} if $(u \neq s_{goal})\ rhs(u) = \min_{s' \in Succ(u)}(c(u, s') + g(s'))$;
{08'} if $(u \in U)$ U.Remove(u);
{09'} if $(g(u) \neq rhs(u))$ U.Insert$(u$, CalculateKey$(u))$;

D* Lite

procedure ComputeShortestPath()
{10'} while (U.TopKey() $\dot{<}$ CalculateKey(s_{start}) OR $rhs(s_{start}) \neq g(s_{start})$)
{11'} $k_{old} = $ U.TopKey();
{12'} $u = $ U.Pop();
{13'} if ($k_{old} \dot{<}$ CalculateKey(u))
{14'} U.Insert(u, CalculateKey(u));
{15'} else if ($g(u) > rhs(u)$)
{16'} $g(u) = rhs(u)$;
{17'} for all $s \in Pred(u)$ UpdateVertex(s);
{18'} else
{19'} $g(u) = \infty$;
{20'} for all $s \in Pred(u) \cup \{u\}$ UpdateVertex(s);

procedure Main()
{21'} $s_{last} = s_{start}$;
{22'} Initialize();
{23'} ComputeShortestPath();
{24'} while ($s_{start} \neq s_{goal}$)
{25'} /* if ($g(s_{start}) = \infty$) then there is no known path */
{26'} $s_{start} = \arg\min_{s' \in Succ(s_{start})}(c(s_{start}, s') + g(s'))$;
{27'} Move to s_{start};
{28'} Scan graph for changed edge costs;
{29'} if any edge costs changed
{30'} $k_m = k_m + h(s_{last}, s_{start})$;
{31'} $s_{last} = s_{start}$;
{32'} for all directed edges (u, v) with changed edge costs
{33'} Update the edge cost $c(u, v)$;
{34'} UpdateVertex(u);
{35'} ComputeShortestPath();

References (A*)

[1] Yap, P. (2002): Grid-based Path-finding. *University of Alberta, Edmonton, Canada.*

- porovnání dlaždicových reprezentací (ocitles, tiles, texes)

[2] Stout, B. (1997). Smart Moves: Intelligent Pathfinding. *Game Developer Magazine*.

- porovnání „jednoduchých algoritmů“, vysvětlení A*

[3] Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik,* 1, 269--271.

- Dijkstrův algoritmus

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

- A* algoritmus

[5] Higgins, D. F. (2002). Pathfinding Design Architecture, How to Achieve Lightning-Fast A*, *AI Game Programming Wisdom. Charles River Media*.

- optimalizace základního A*

References (hierarchické plánování)

[6] Rabin, S. (2000). A* Aesthetic Optimizations. *In Mark Deloura, editor, Game Programming Gems*, pages 264--271. Charles River Media.

- vylepšení A*, např. „hledání za roh“ (nejjednodušší hierarchický přístup) a další

[7] D. Z. Chen, R. J. Szczerba, and J. J. Urhan Jr. (1995). Planning Conditional Shortest Paths Through an Unknown Environment: A Framed-Quadtree Approach. *In Proceedings of the 1995 IEEE/RSJ Int. Conf. on Intelligent Robots etc.*, vol. 3, pages 33--38.

- Framed Quadtrees

[8] Botea, A., Müller, M., Schaeffer, J. (2004). Near Optimal Hierarchical Path-Finding. *Dep. of Comp. Science, University of Alberta*.

- HPA*, Hierarchical Path-Finding A*, velmi pěkný a srozumitelný článek

[9] Sturtevant, N., Buro, M. (2005). Partial Pathnding Using Map Abstraction and Refinement. *Amer. Assoc. for AI* (www.aaai.org).

- PRA*, Partial–Refinement A*, poněkud náročnější, ale má co říci

[10] Bajer, L. (2005). Hierarchical– and Partial–Refinement Pathfinding. *MFF UK, Praha*.

- Porovnání HPA* a PRA*

References (incremental heur. search)

[11] Koenig, S., Likhachev, M. (2001). Incremental A*. *Georgia Inst. of Techn., College of Computing, Atlanta*.

- LPA*, Lifelong planning A*

[12] Manley, K. (2004). Pathfinding: From A* to LPA*. *University of Minesota, Moris.*

- pěkné vysvětlení LPA*

[13] Stenz, A. (1994). Optimal and Efficient Path Planning for Partially-Known Environments. *The Robotics Inst., Mellon University, Pittsburgh*.

- D*, základní verze

[14] Stenz, A. (1995). The Focussed D* Algorithm for Real-Time Replanning. *The Robotics Inst., Mellon University, Pittsburgh*.

- Focussed D*, rychlejší verze (přesnější zaměření hledání)

[15] Koenig, S., Likhachev, M. (2002). D* Lite. *Amer. Assoc. for AI* (www.aaai.org).

- D* Lite, alespoň tak dobrý jako D*, srovnatelný s Focussed D*, ale jednodušší, aplikace LPA* jako alg. na aktualizaci změn v prostředí, nejlepší článek o D* z výše uv.