Any-Angle Search

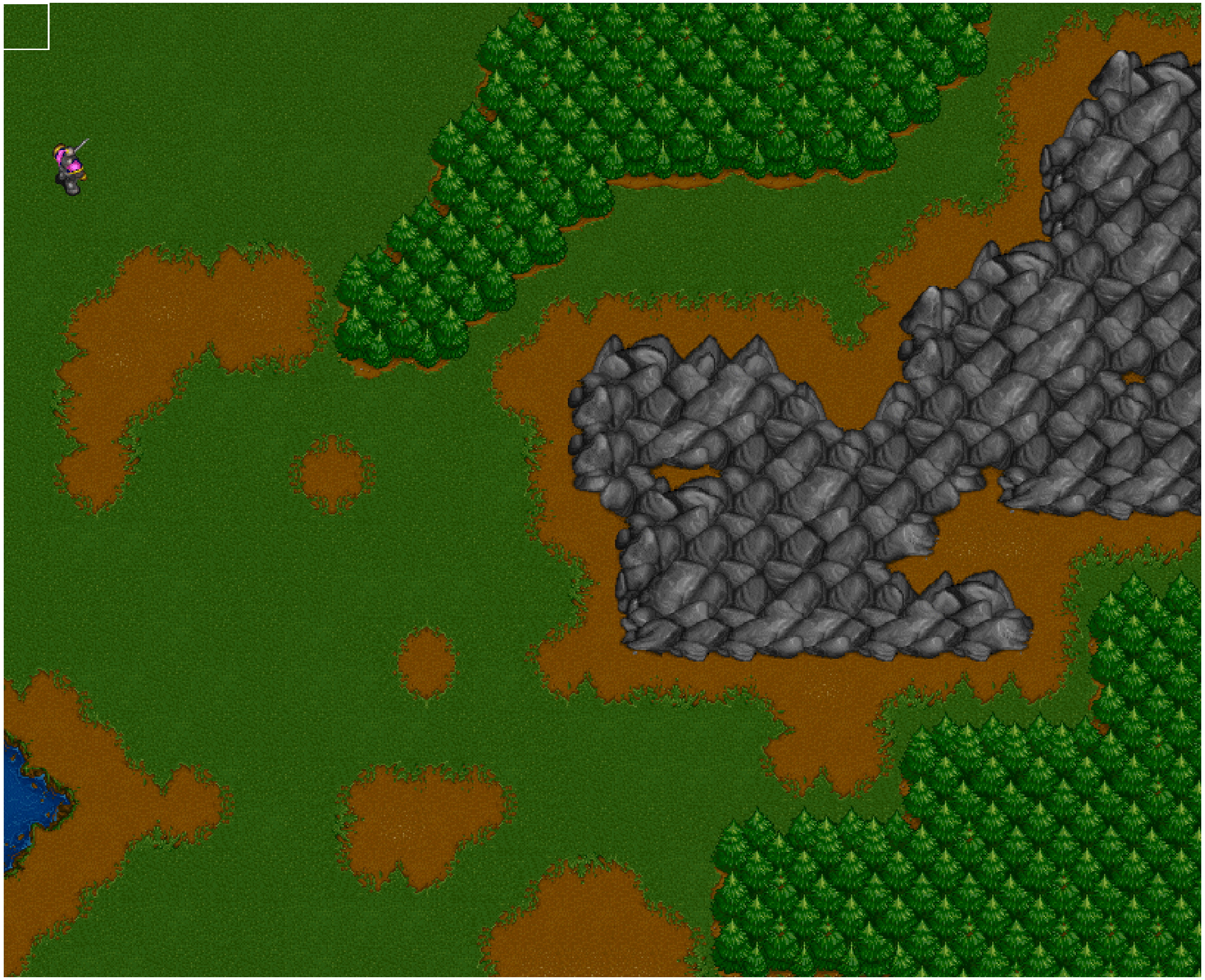
Case Study: Theta*

slides by Alex Nash
anash@usc.edu / alexwnash@gmail.com
with contributions by Sven Koenig
skoenig@usc.edu

Table of Contents

- Introduction
- Analysis of Path Lengths
- Any-Angle Search Methods
    - Known 2D Environments
    - Known 3D Environments
    - Unknown 2D Environments (1 slide only)
- Conclusion

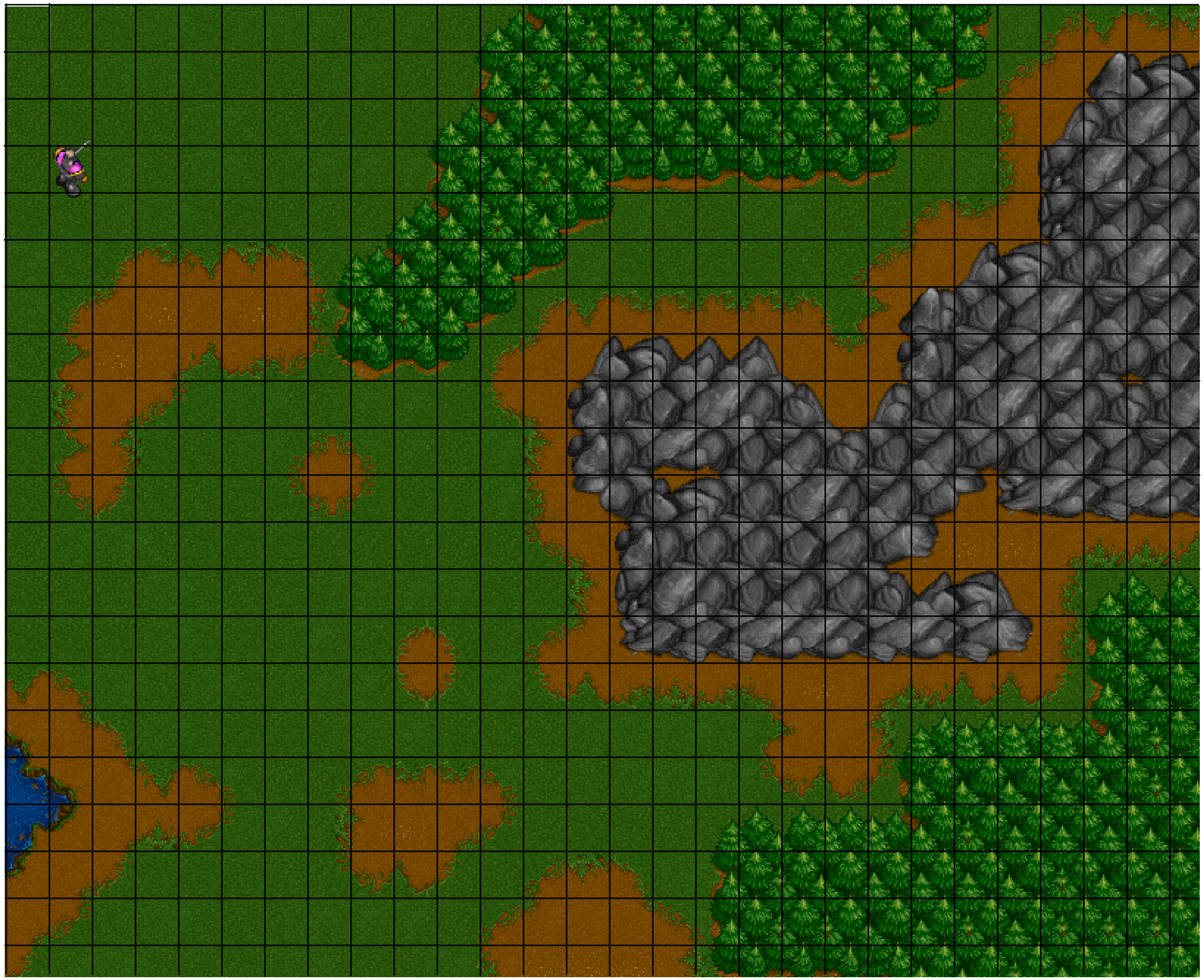
Introduction



Warcraft II

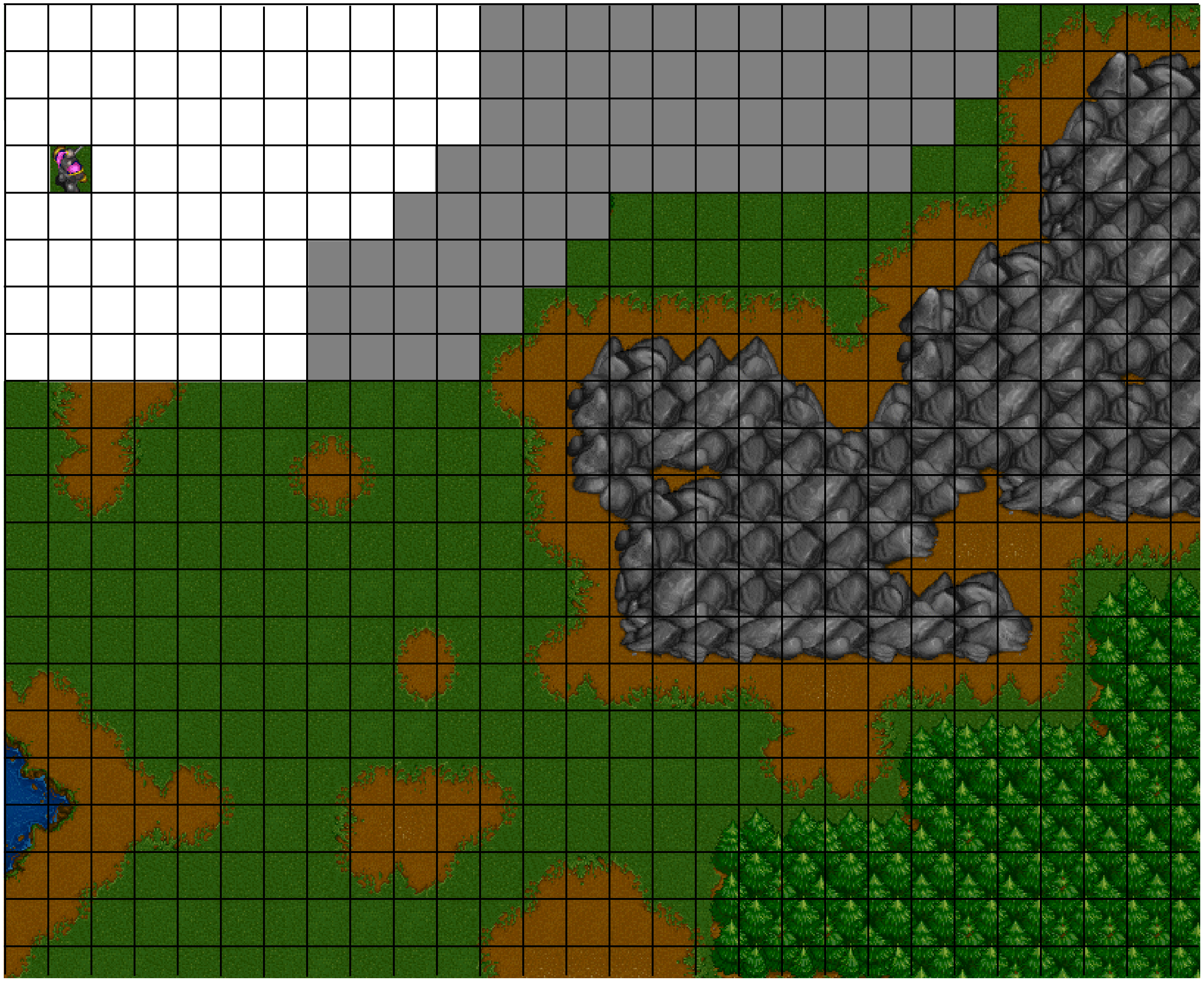
[from JPL]

Introduction



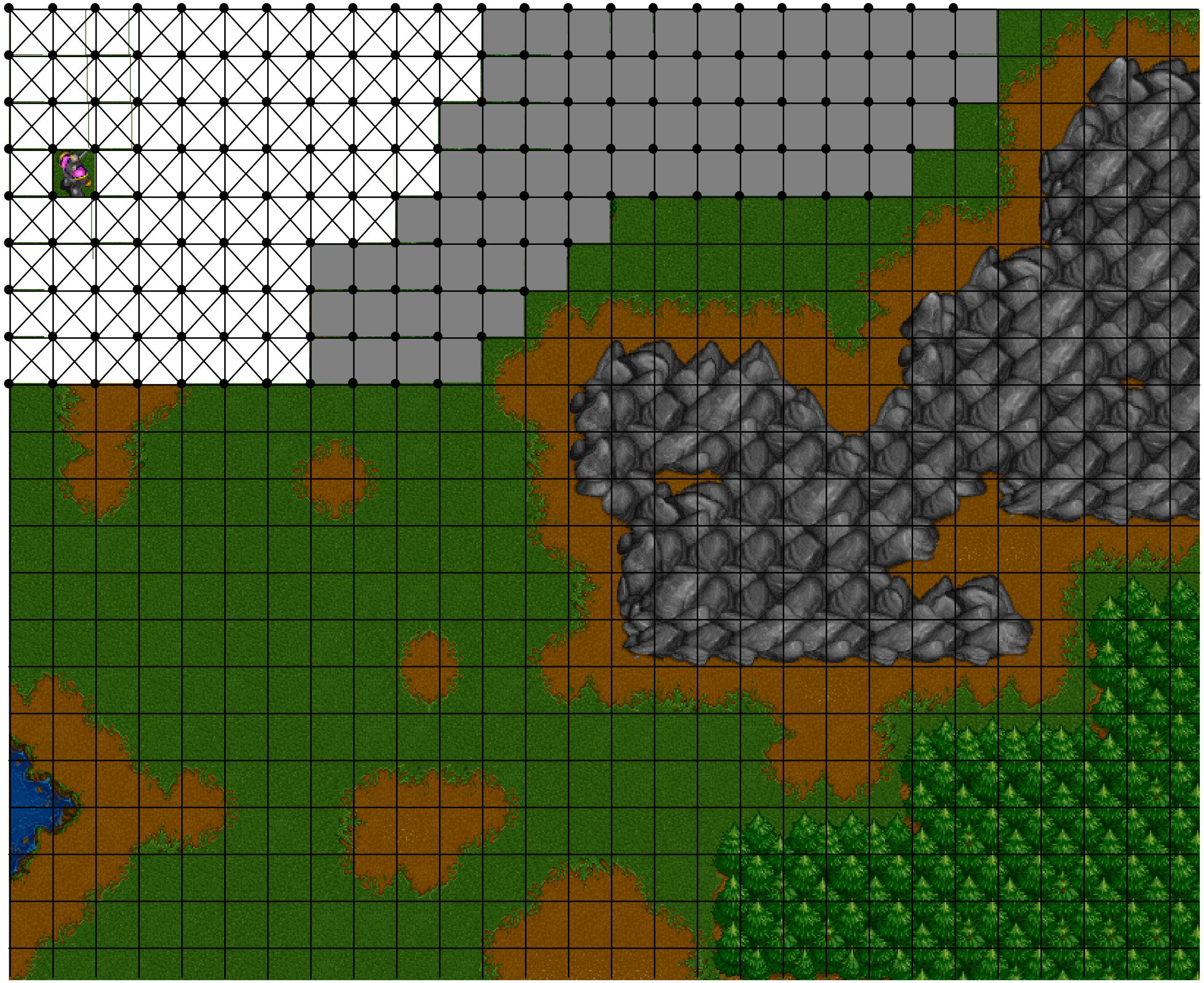
Warcraft II

Introduction

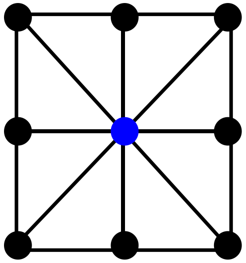


Warcraft II

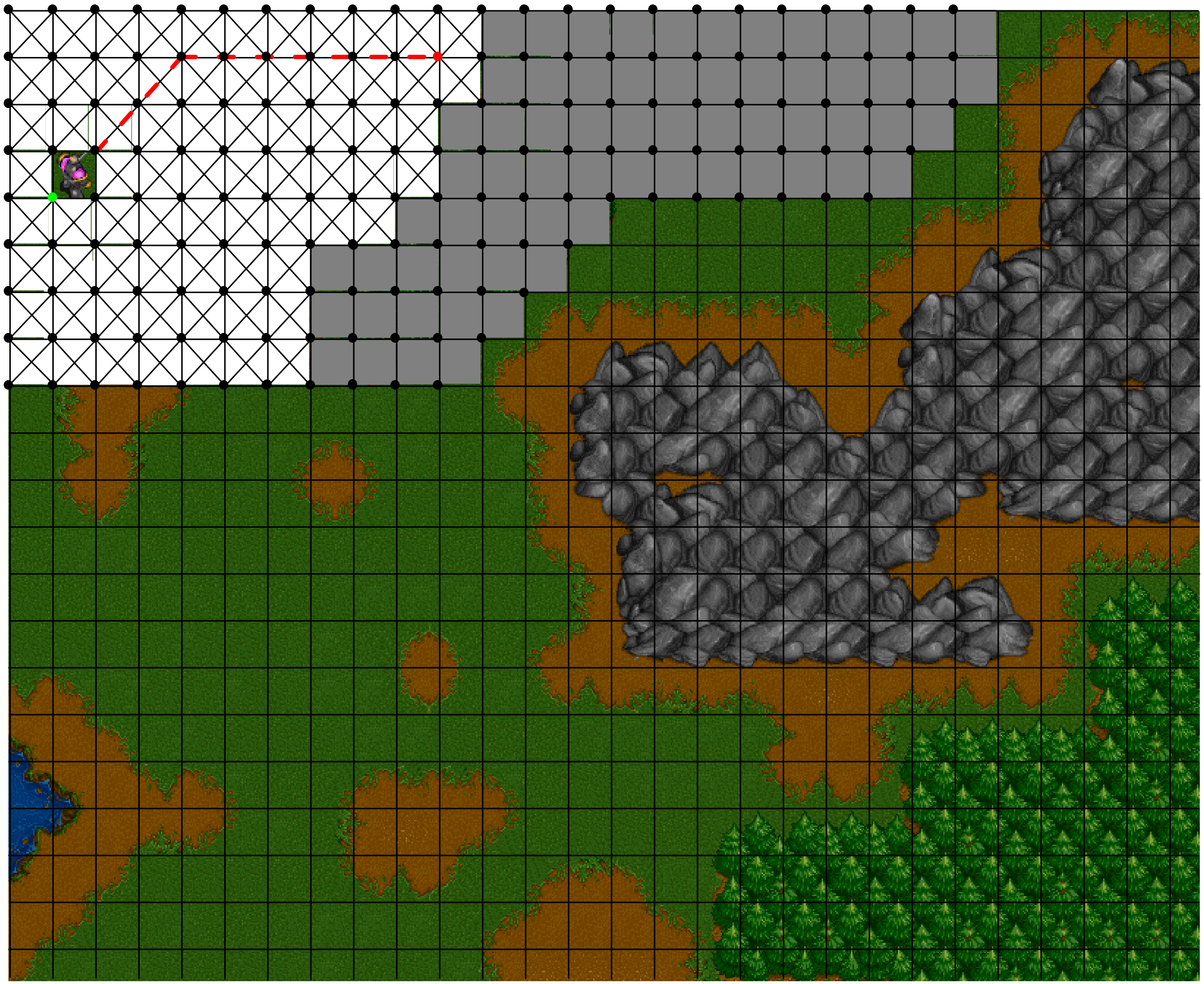
Introduction



Warcraft II

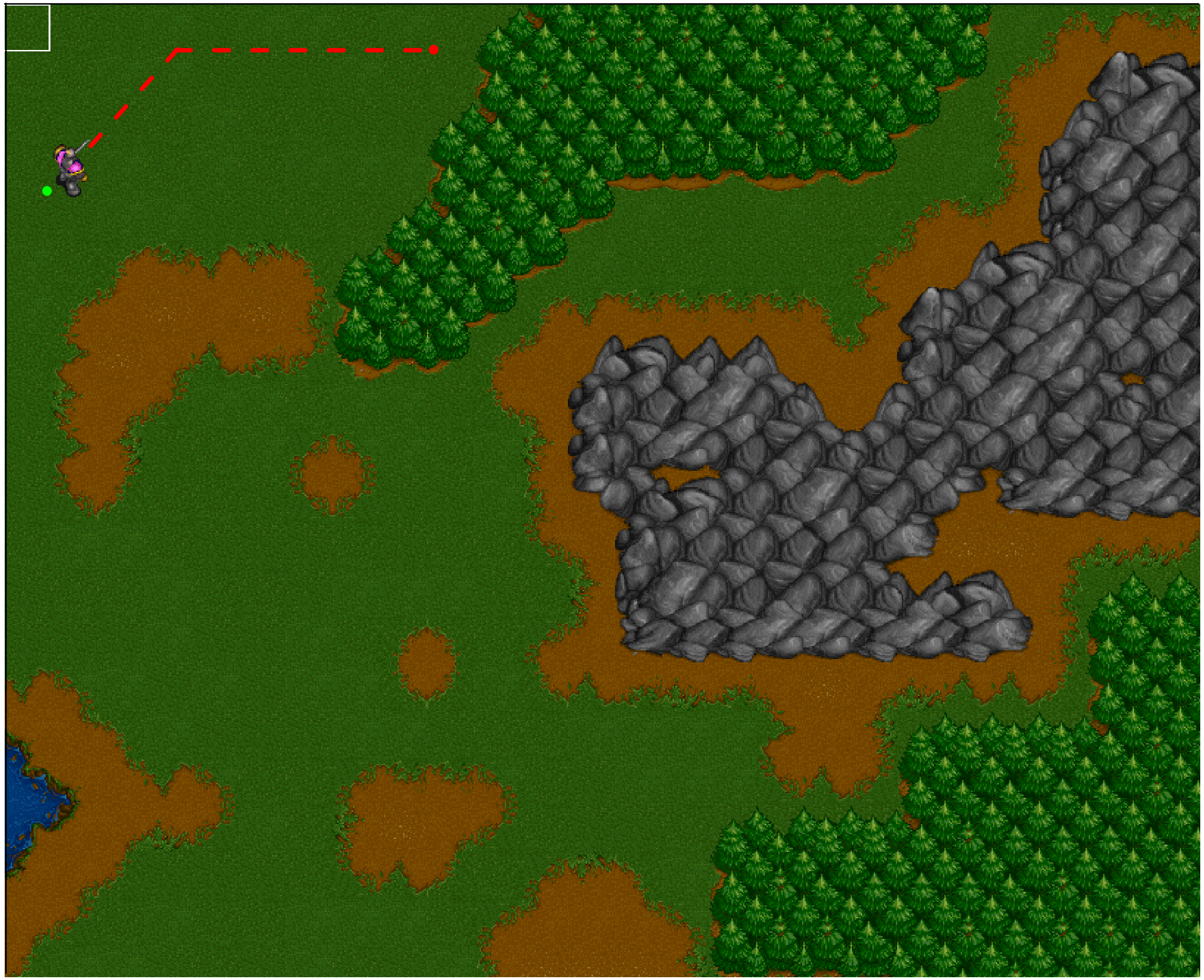


Introduction



Warcraft II

Introduction



Warcraft II

Introduction

Introduction

- Edge-constrained search methods (such as A*)
  - Efficient
  - Simple
  - Generic

  - Long and unrealistic looking paths

Table of Contents

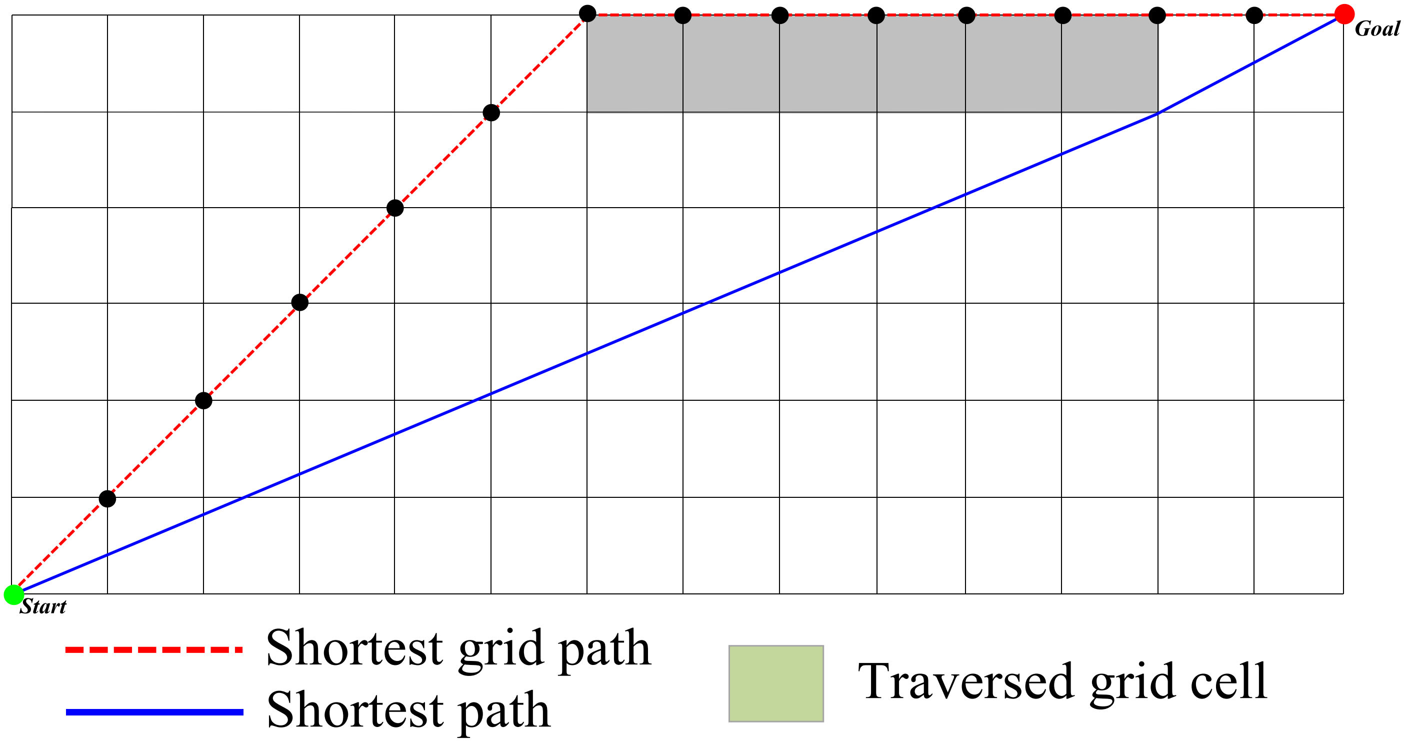
- Introduction
- Analysis of Path Lengths
- Any-Angle Search Methods
    - Known 2D Environments
    - Known 3D Environments
    - Unknown 2D Environments (1 slide only)
- Conclusion

Analysis of Path Lengths

- How much longer can the paths found by traditional edge-constrained search methods (= the shortest paths formed by grid edges) be than the shortest paths in the environment (= the shortest “any-angle” paths = the shortest paths)?

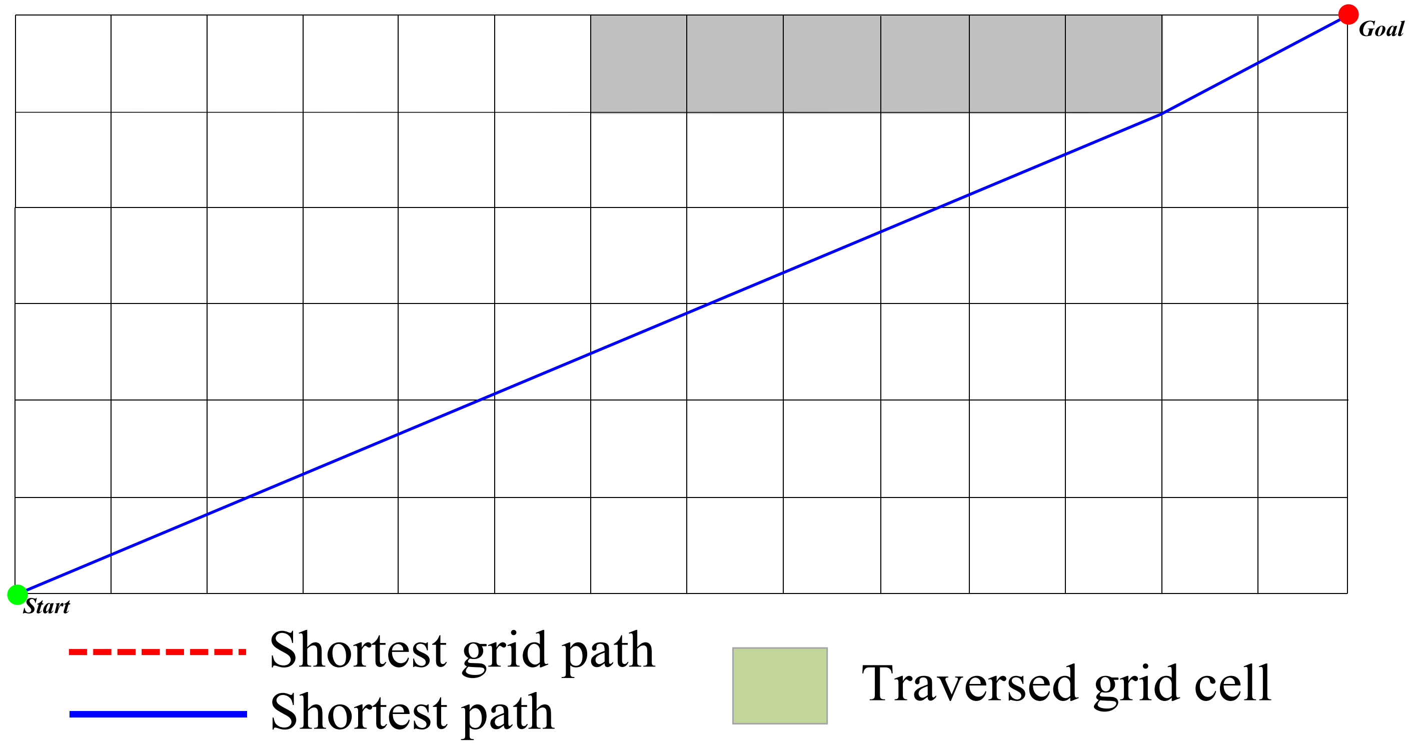
Analysis of Path Lengths

- Blocked cells do not matter



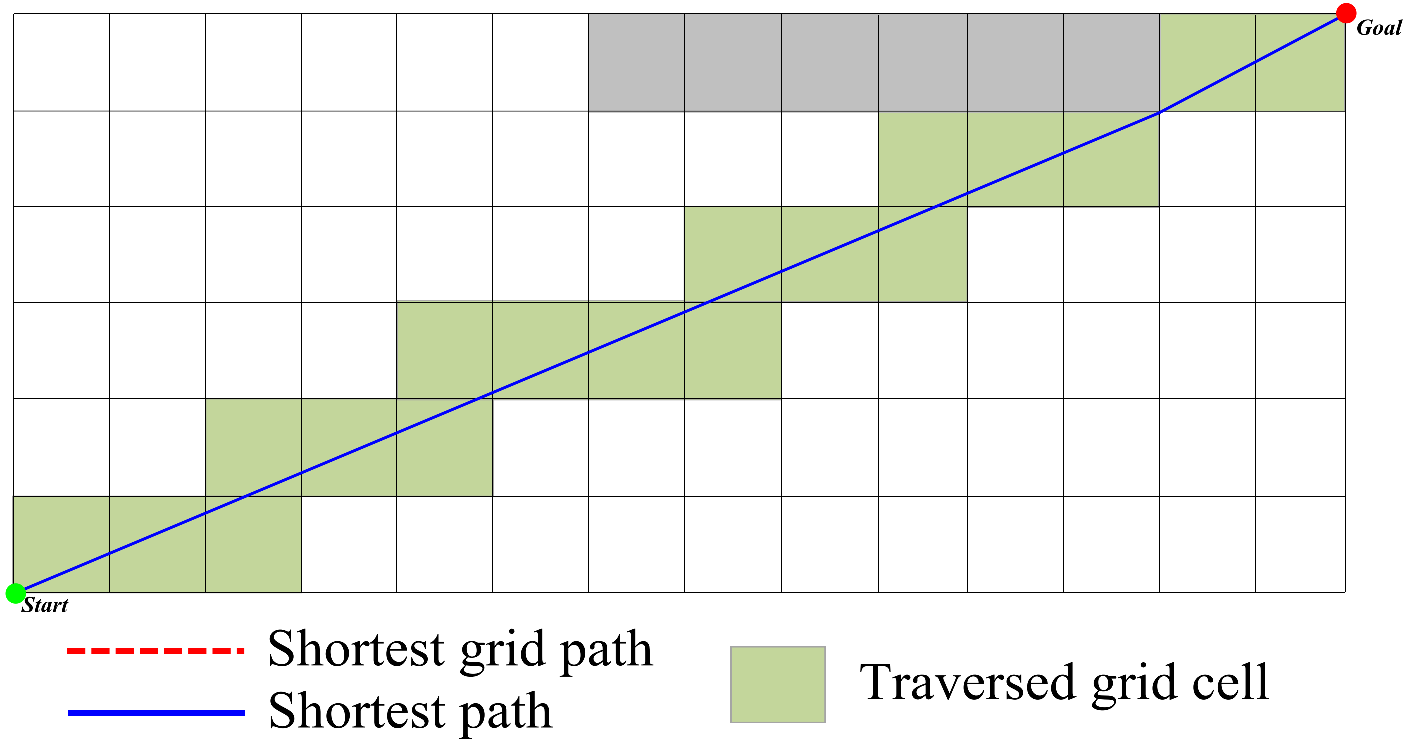
Analysis of Path Lengths

- Blocked cells do not matter



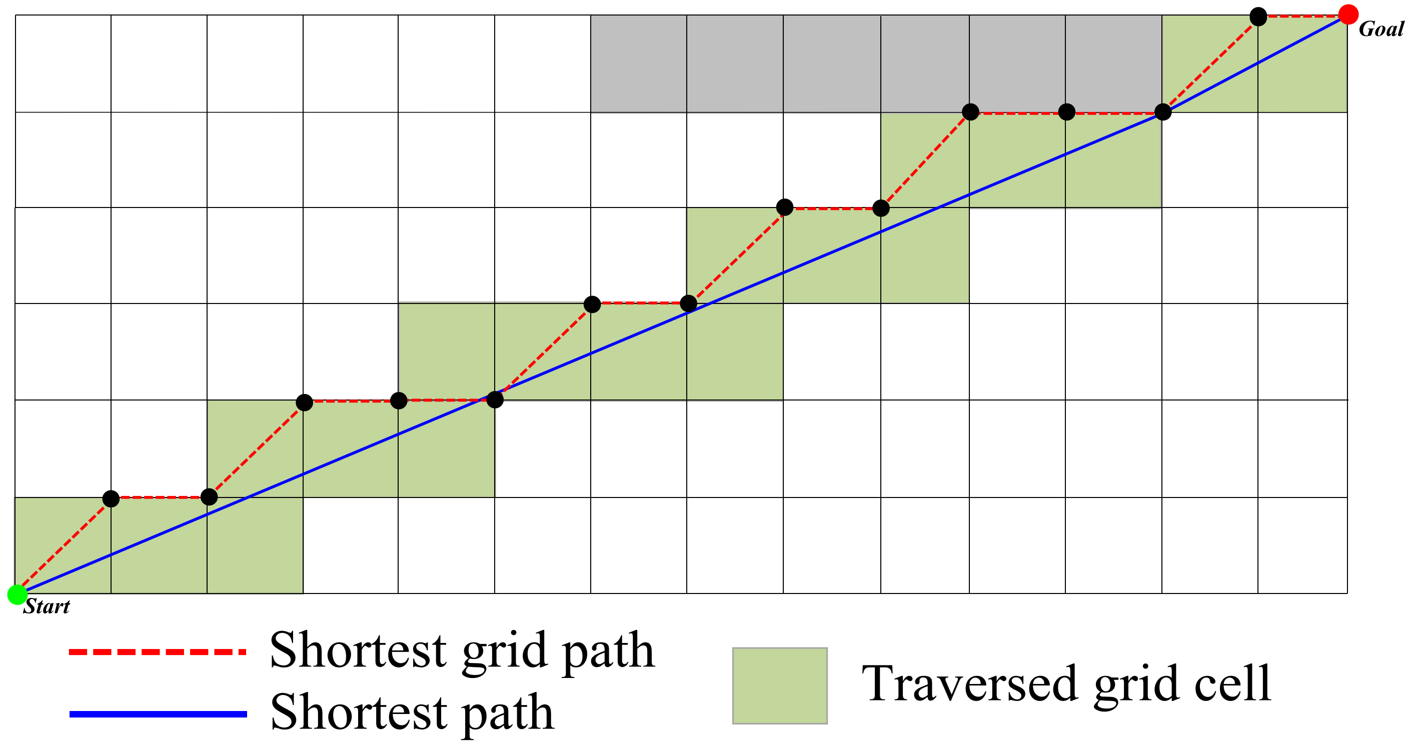
Analysis of Path Lengths

- Blocked cells do not matter



Analysis of Path Lengths

- Blocked cells do not matter



Analysis of Path Lengths

- Part 1: Show that the shortest path formed by grid edges can traverse only grid cells traversed by the shortest path (= blocked cells do not matter)
- Part 2: Determine the worst-case ratio of the length of the shortest path formed by grid edges and the shortest path
  - Optimization problem with Lagrange multipliers
  - $L(x_1,\ldots,x_n,\lambda) = f(x_1,\ldots,x_n) + \lambda(g(x_1,\ldots,x_n)-c)$
  - Minimize $f(x_1,\ldots,x_n)$ subject to $g(x_1,\ldots,x_n) = c$
  - $f(x_1,\ldots,x_n)$ = length of the shortest path
  - $g(x_1,\ldots,x_n)$ = length of the shortest grid path

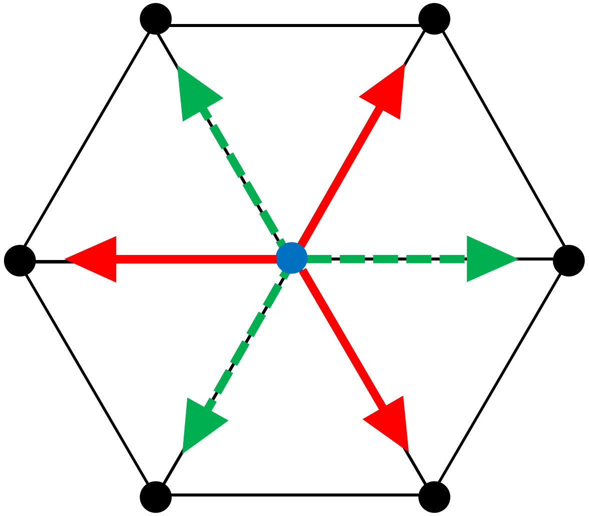
Analysis of Path Lengths

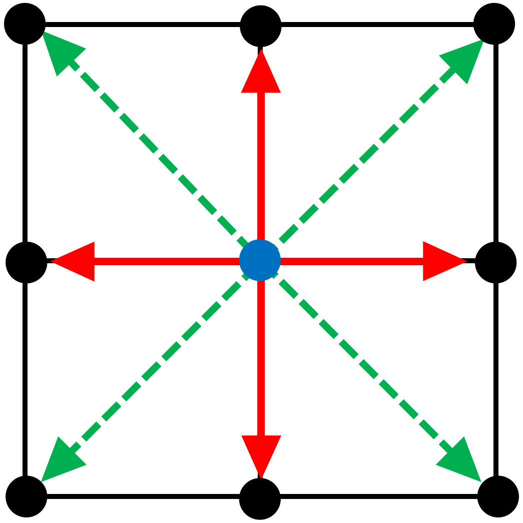
Dimension	Regular Grid	Neighbors	% Longer Than Shortest Path
2D	triangular grid	3-neighbor	
		6-neighbor	
	square grid	4-neighbor	
		8-neighbor	≈ 8
	hexagonal grid	6-neighbor	
		12-neighbor	
3D	cubic grid	6-neighbor	
		26-neighbor	

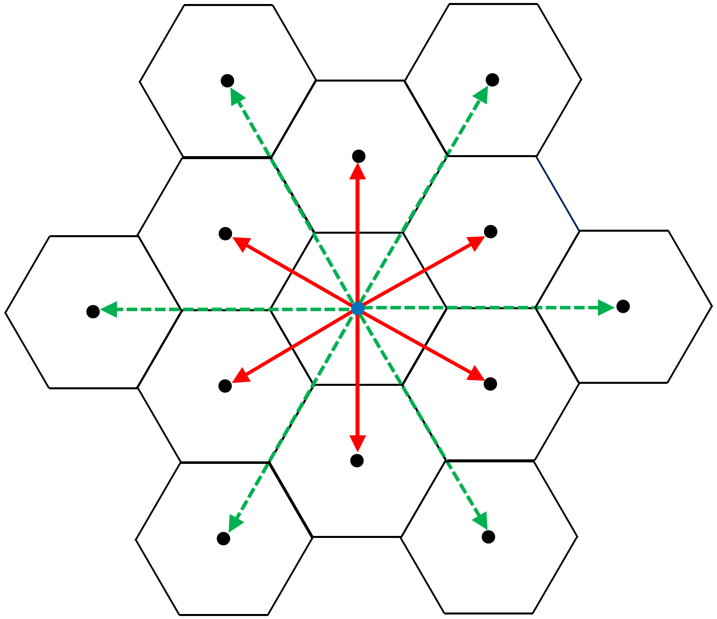
Analysis of Path Lengths

- Only three types of regular (equilateral and equiangular) polygons can be used to tessellate 2D environments
  - Triangles
  - Squares
  - Hexagons

Analysis of Path Lengths







Civilization V

Red Arrows

3-Neighbor Triangular Grid 4-Neighbor Square Grid 6-Neighbor Hexagonal Grid

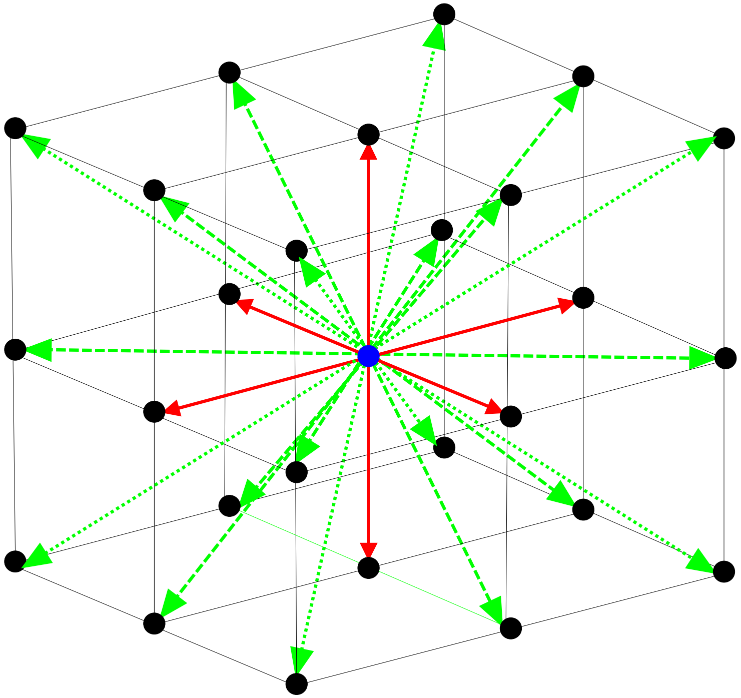
Red and Green Arrows

6-Neighbor Triangular Grid 8-Neighbor Square Grid 12-Neighbor Hexagonal Grid

Analysis of Path Lengths

- Only one type of regular polyhedron can be used to tessellate 3D environments
  - Cubes

Analysis of Path Lengths



Red Arrows

6-Neighbor Cubic Grid

Red and Green Arrows

26-Neighbor Cubic Grid

Analysis of Path Lengths

Dimension	Regular Grid	Neighbors	% Longer Than Shortest Path
2D	triangular grid	3-neighbor	≈ 100
		6-neighbor	≈ 15
	square grid	4-neighbor	≈ 41
		8-neighbor	≈ 8
	hexagonal grid	6-neighbor	≈ 15
		12-neighbor	≈ 4
3D	cubic grid	6-neighbor	≈ 73
		26-neighbor	≈ 13

Table of Contents

- Introduction
- Analysis of Path Lengths
- Any-Angle Search Methods
  - Known 2D Environments
  - Known 3D Environments
  - Unknown 2D Environments (1 slide only)
- Conclusion

Any-Angle Search

- Two conflicting criteria

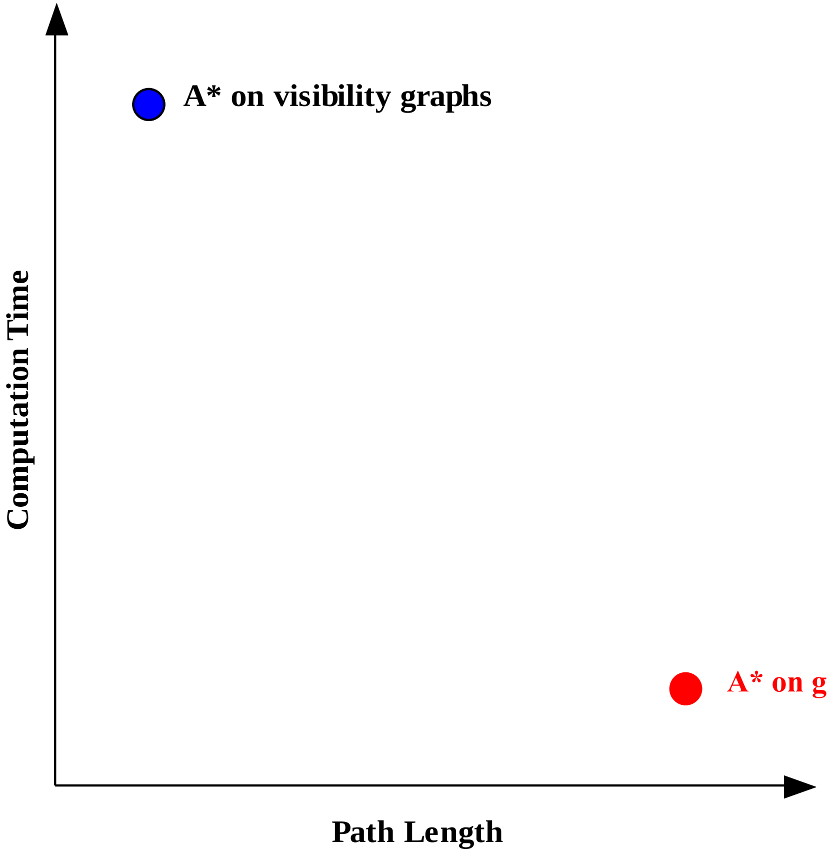
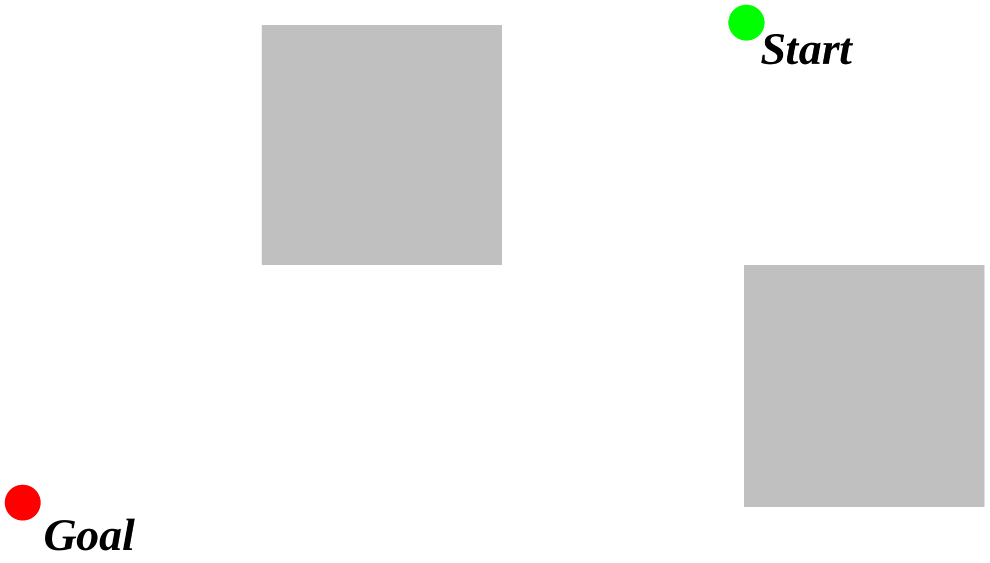


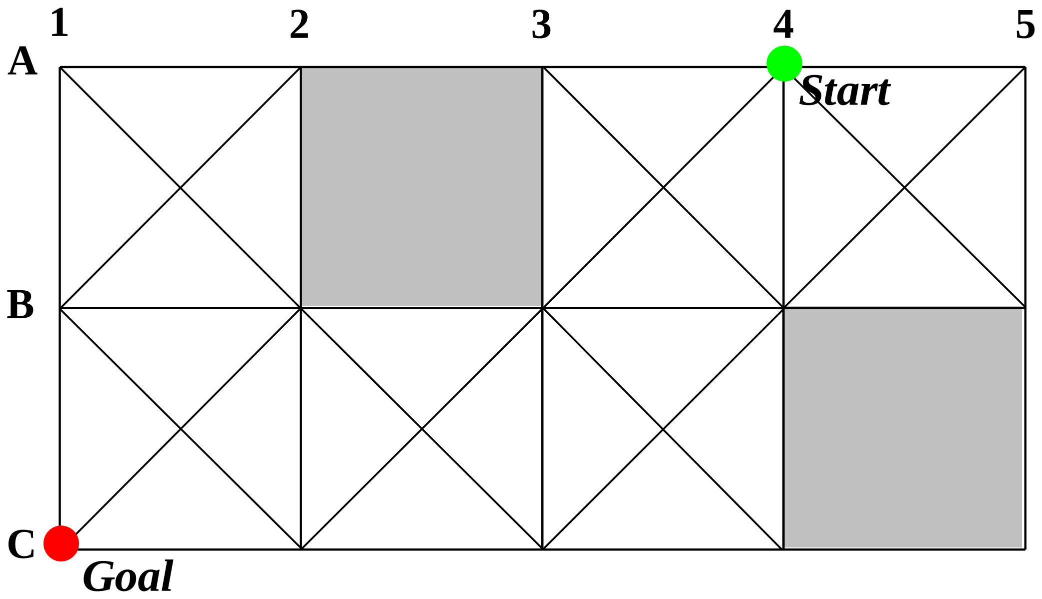
figure is notional

A* on Grid Graphs



A* on Grid Graphs

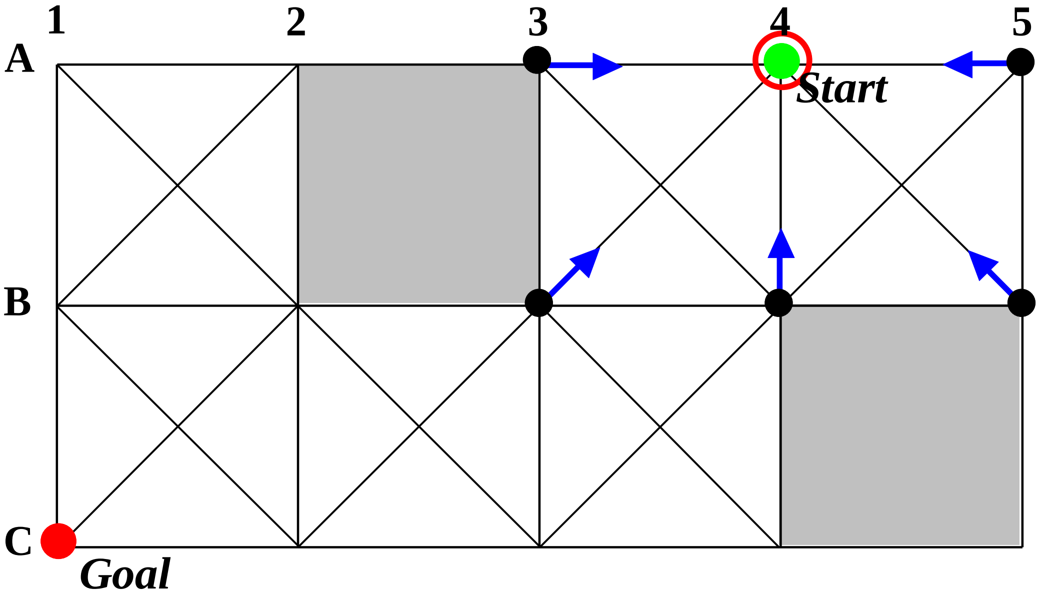
- A* [Hart et al. (1968)] on grid graphs



- A* assigns two values to every vertex s
  - g(s): the length of the shortest path from the start vertex to s found so far
  - parent(s): the parent pointer used to extract the path after termination
  - Following the parents from s to the start vertex results in a path of length g(s)

A* on Grid Graphs

- A* on grid graphs

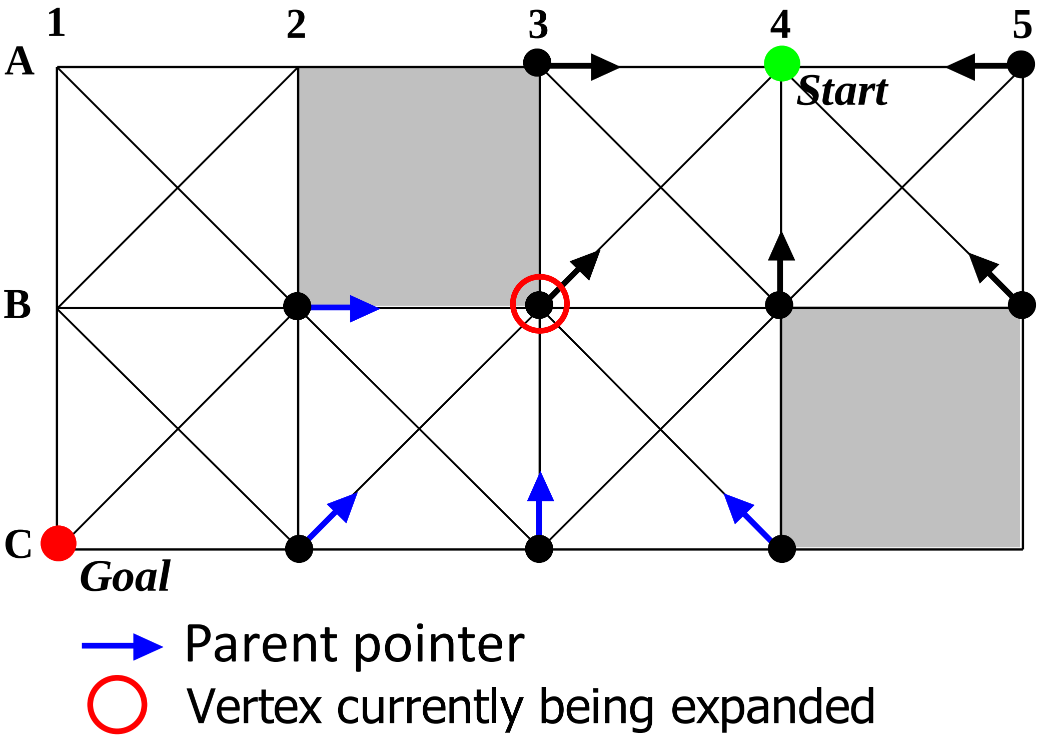


→ Parent pointer

○ Vertex currently being expanded

A* on Grid Graphs

- A* on grid graphs

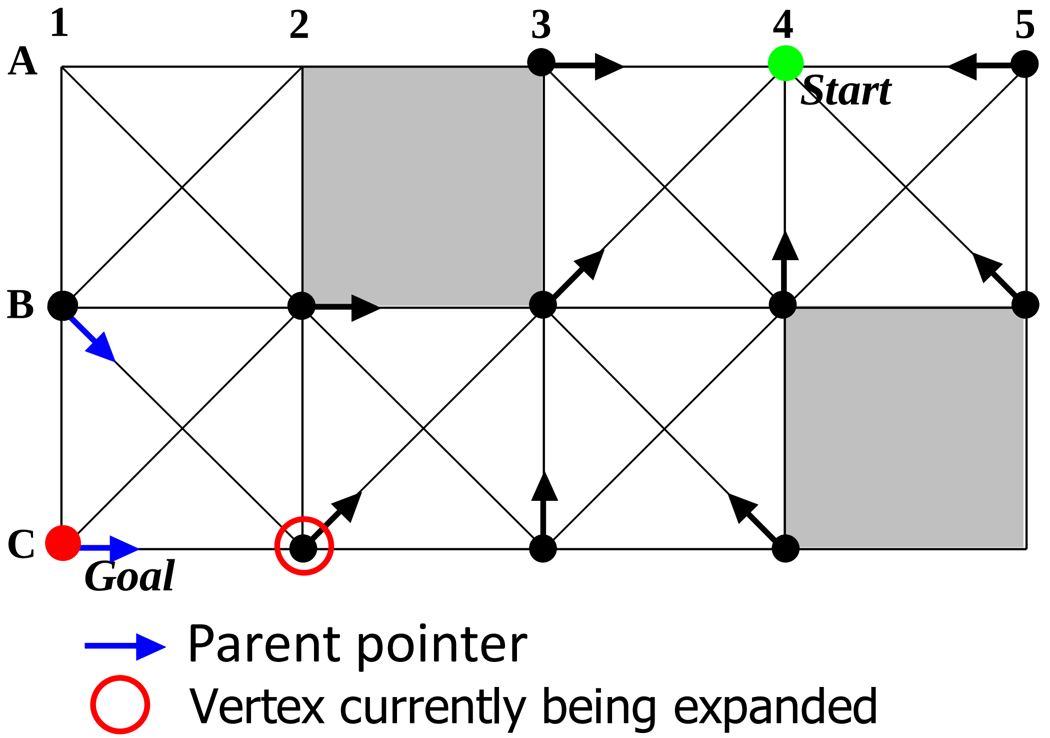


→ Parent pointer

○ Vertex currently being expanded

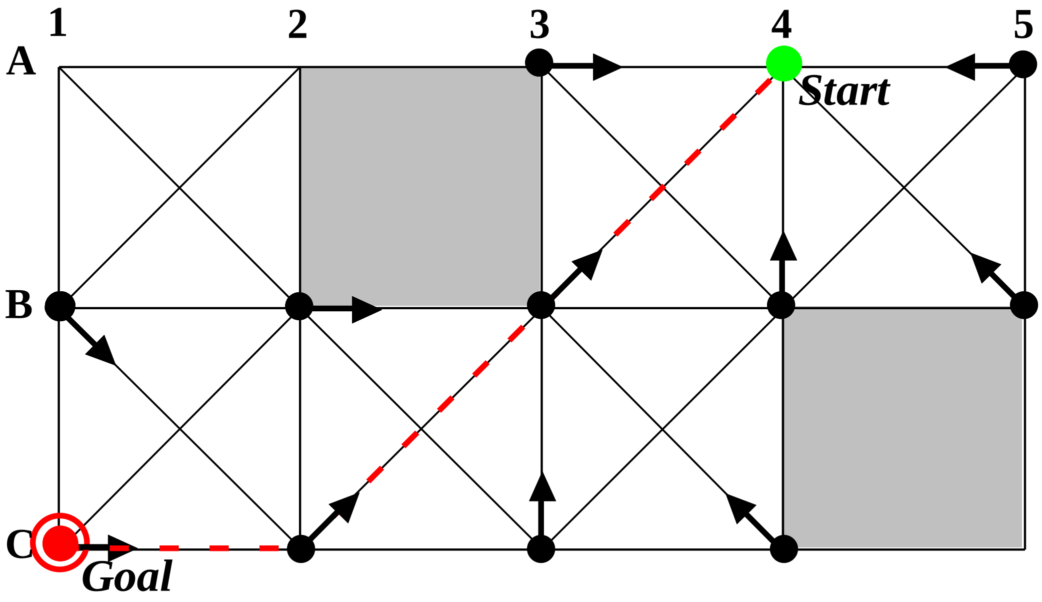
A* on Grid Graphs

- A* on grid graphs



A* on Grid Graphs

- A* on grid graphs



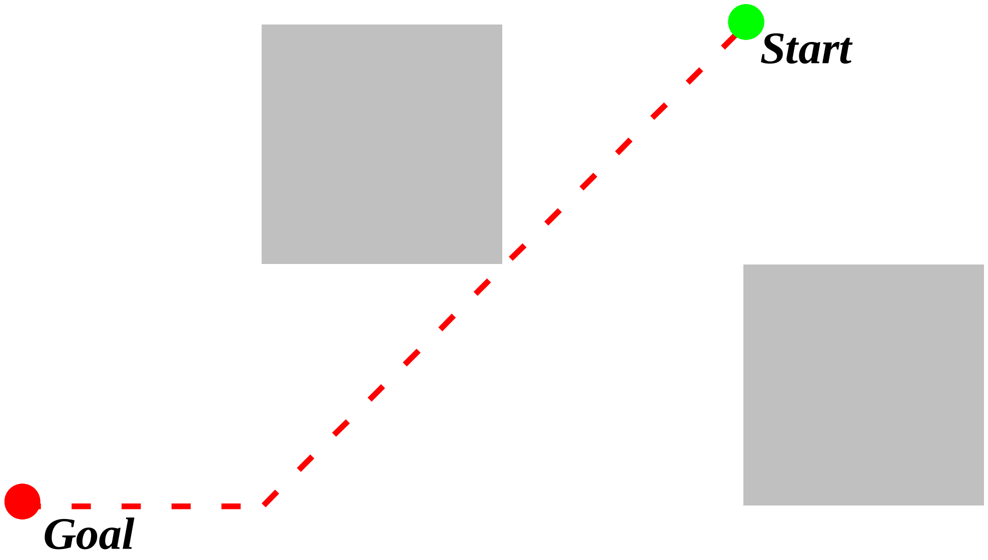
→ Parent pointer

○ Vertex currently being expanded

from now on, we will show only the cells

A* on Grid Graphs

- A* on grid graphs



- Long and unrealistic looking path
- Fast

Known 2D Environments

A* on Visibility Graphs

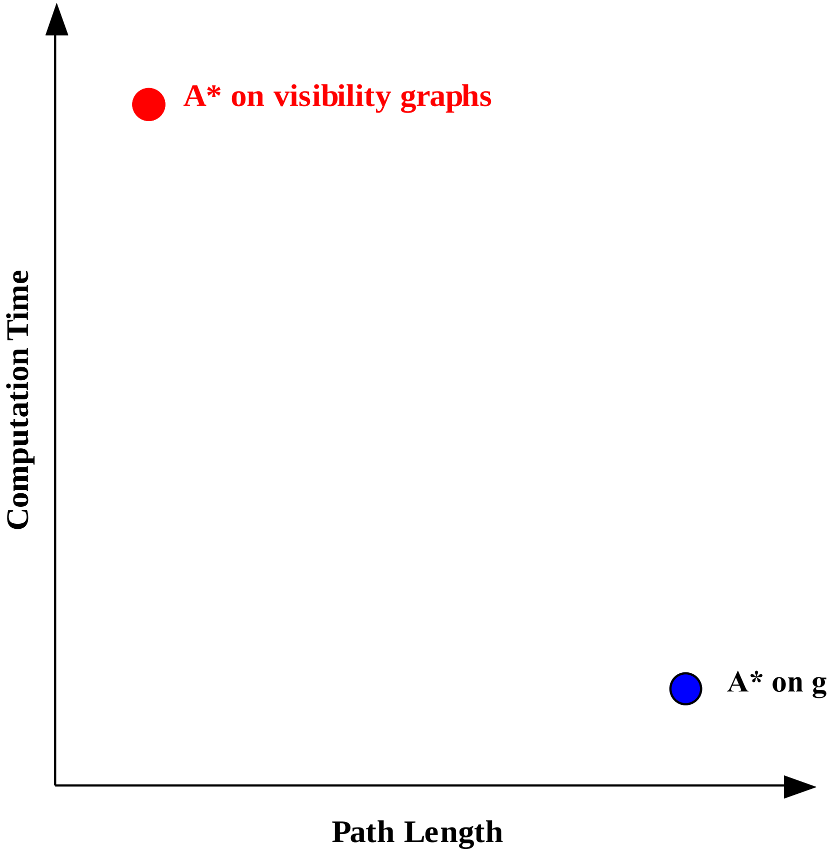
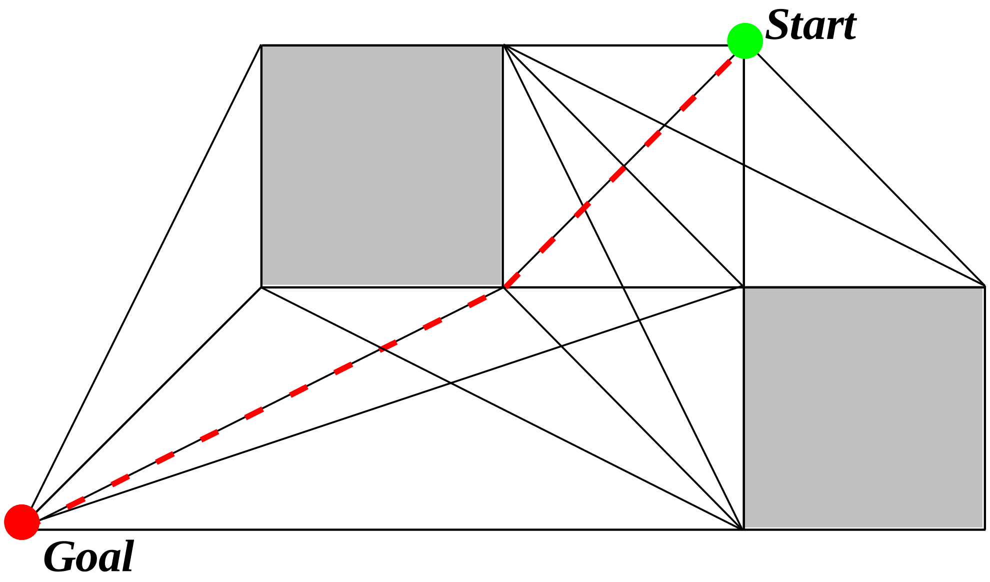


figure is notional

A* on Visibility Graphs

- A* on Visibility Graphs [Lozano-Perez et al. (1979)]



- Shortest path
- Slow due to many edges and line-of-sight checks

Known 2D Environments

Any-Angle Search

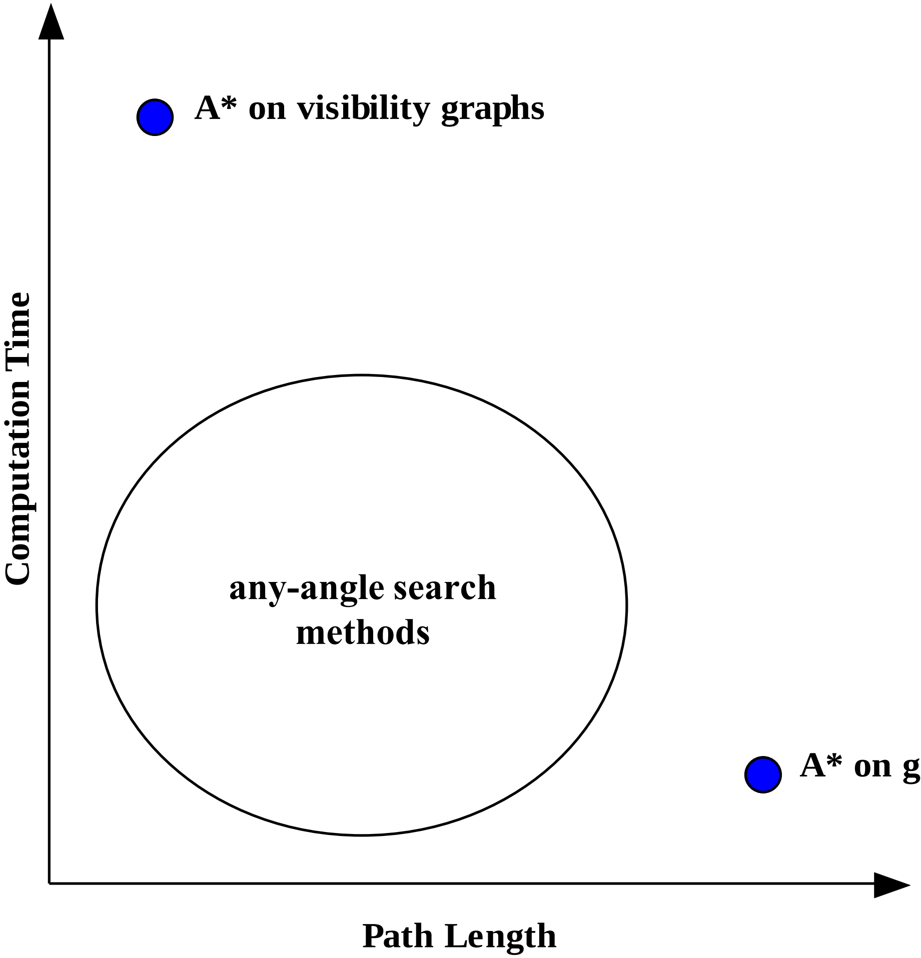


figure is notional

Any-Angle Search

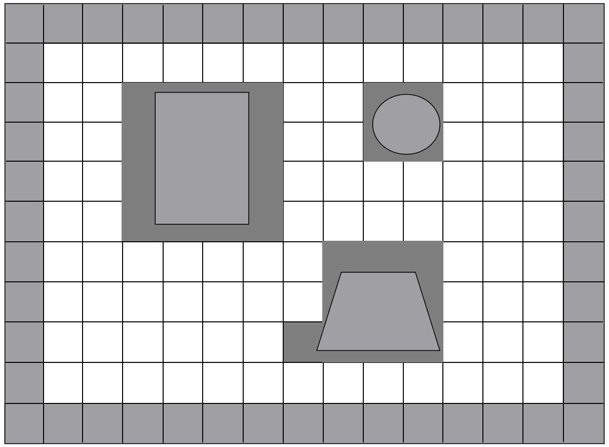
- Any-angle search methods
  - Perform an A* search
  - Propagate information along grid edges
(= small computation time)
  - Do not constrain the paths to be formed by grid edges
(= short paths)

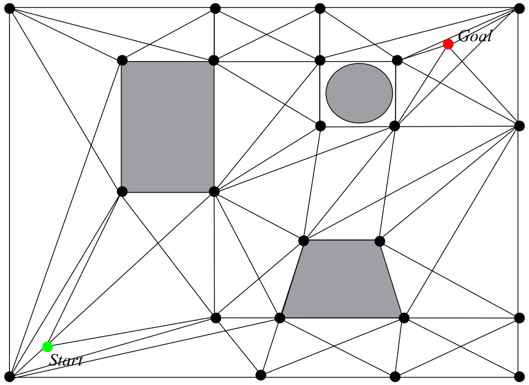
Any-Angle Search

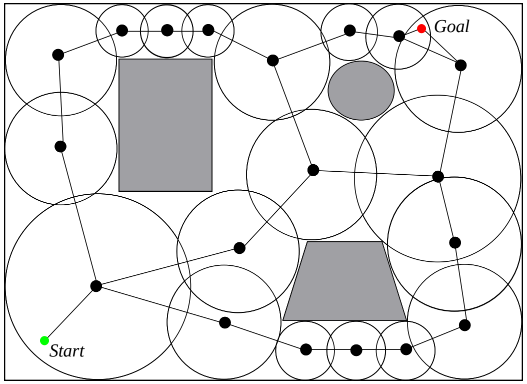
- Evaluation in
  known 2D environments,
  known 3D environments, and
  unknown 2D environments
  - Property 1 (Efficiency)
    Good tradeoff between computation times and path lengths
  - Property 2 (Simplicity)
    Simple to understand and implement
  - Property 3 (Generality)
    Works on every graph embedded in 2D or 3D Euclidean space (= Euclidean graph), that is, all discretization techniques

Any-Angle Search

- Different discretization techniques







Regular Grids

Dawn of War 1 and 2
Civilization V
Company of Heroes
[Champandard (2010)]

Navigation Meshes

Halo 2
Counter-Strike:
Source and Metroid Prime
[Tozour (2008)]

Circle Based Waypoint Graphs

MechWarrior 4: Vengeance

A* with Post Smoothing

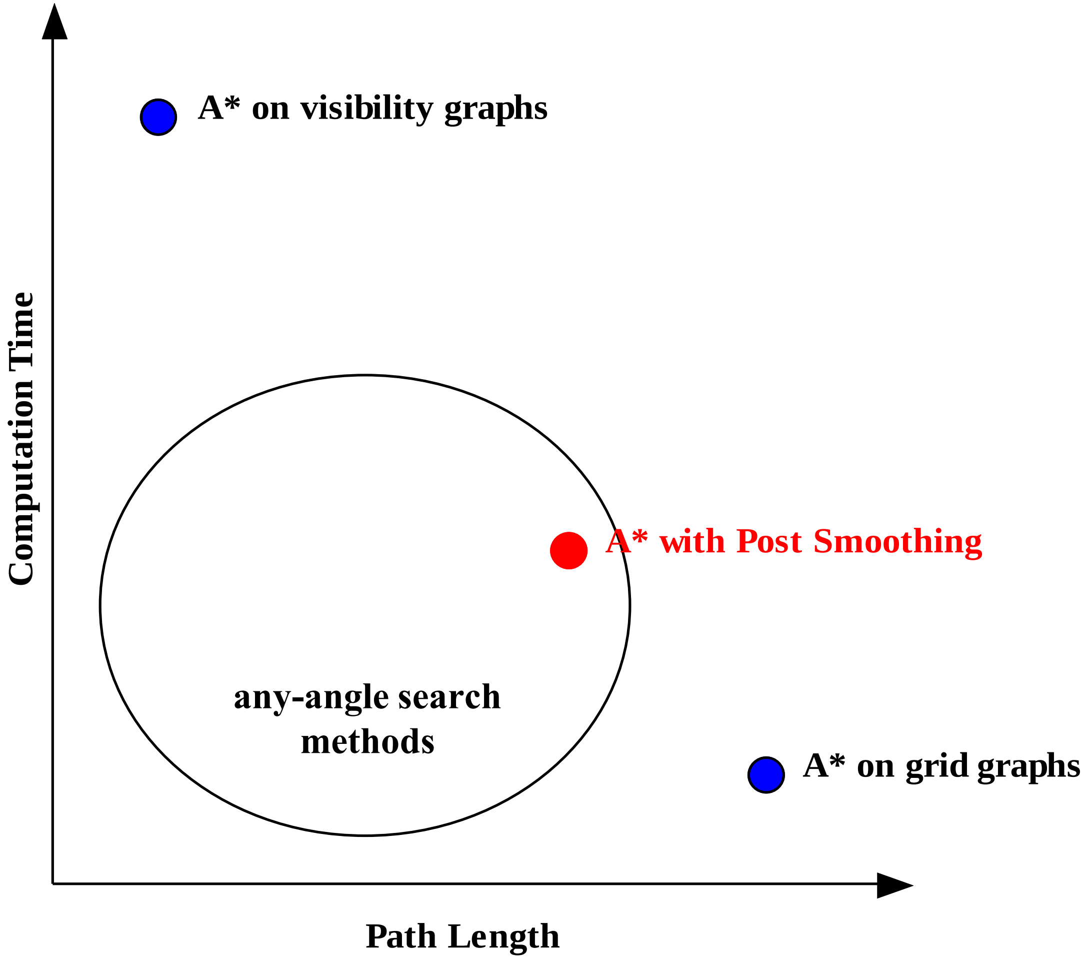
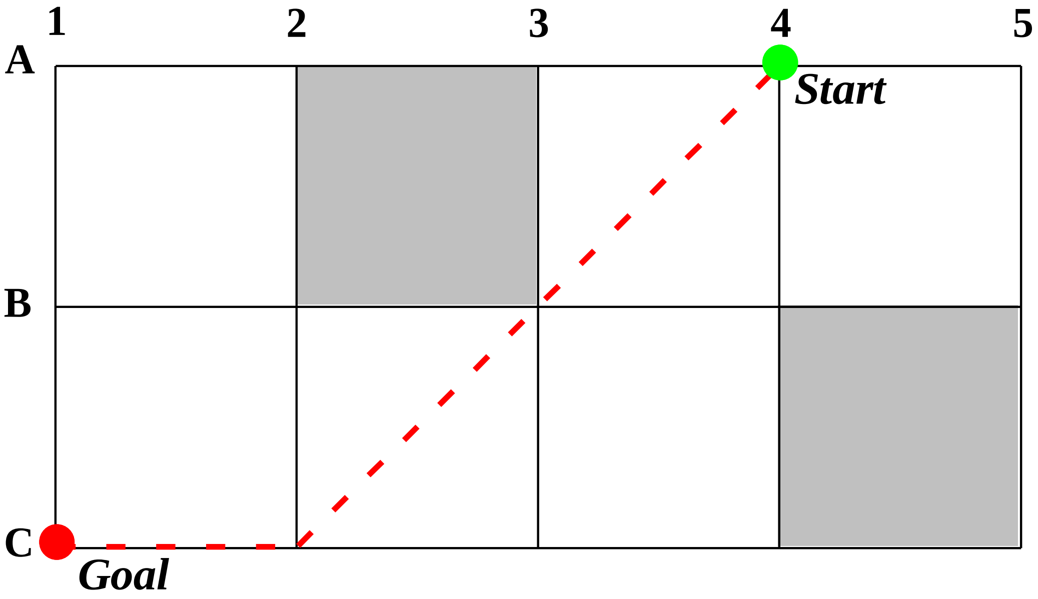


figure is notional

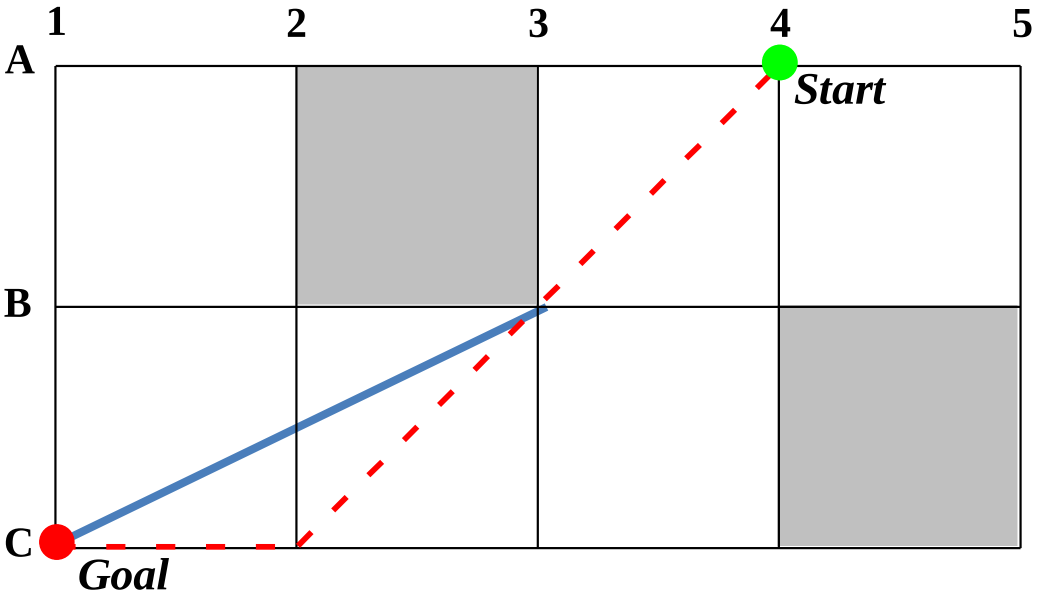
A* with Post Smoothing

- A* with Post Smoothing [Thorpe (1984), Botea et al. (2004), Millington (2009)]



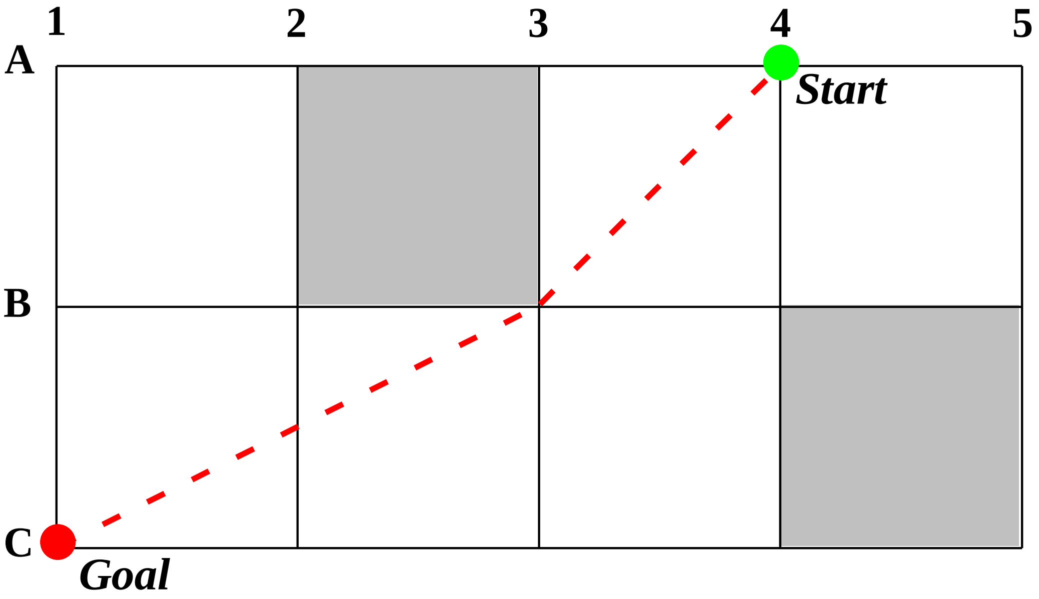
A* with Post Smoothing

- A* with Post Smoothing [Thorpe (1984), Botea et al. (2004), Millington (2009)]



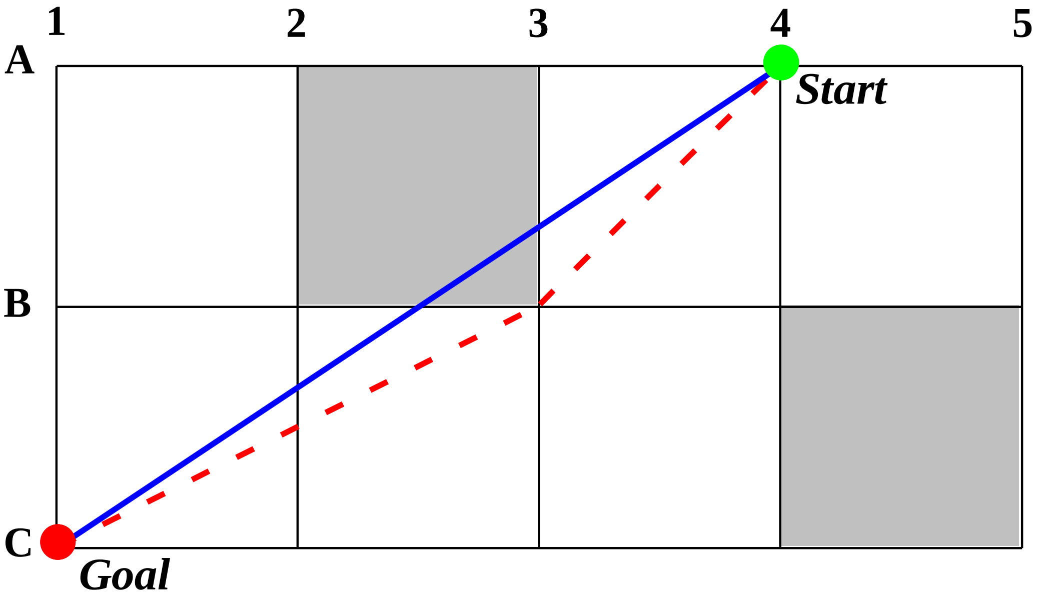
A* with Post Smoothing

- A* with Post Smoothing



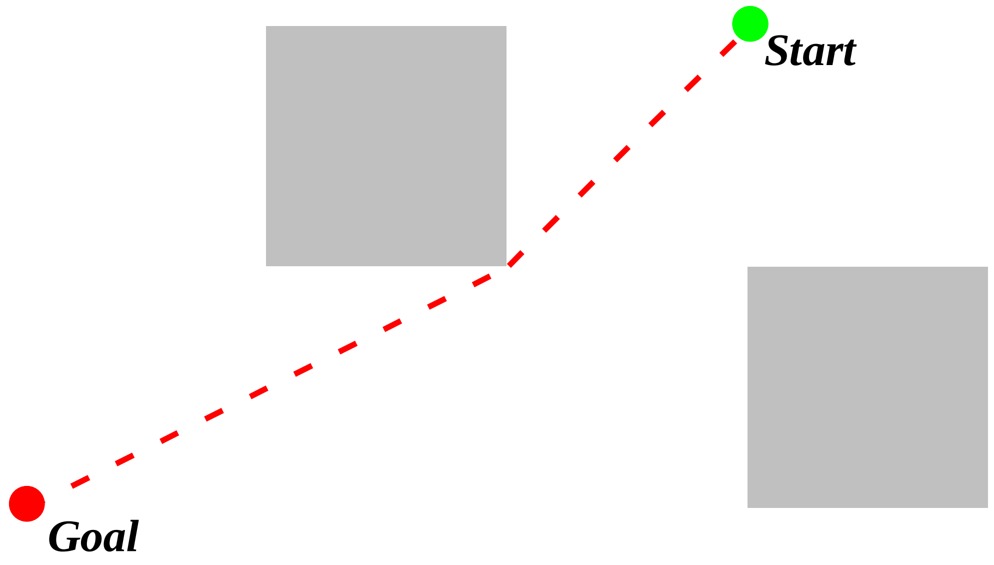
A* with Post Smoothing

- A* with Post Smoothing



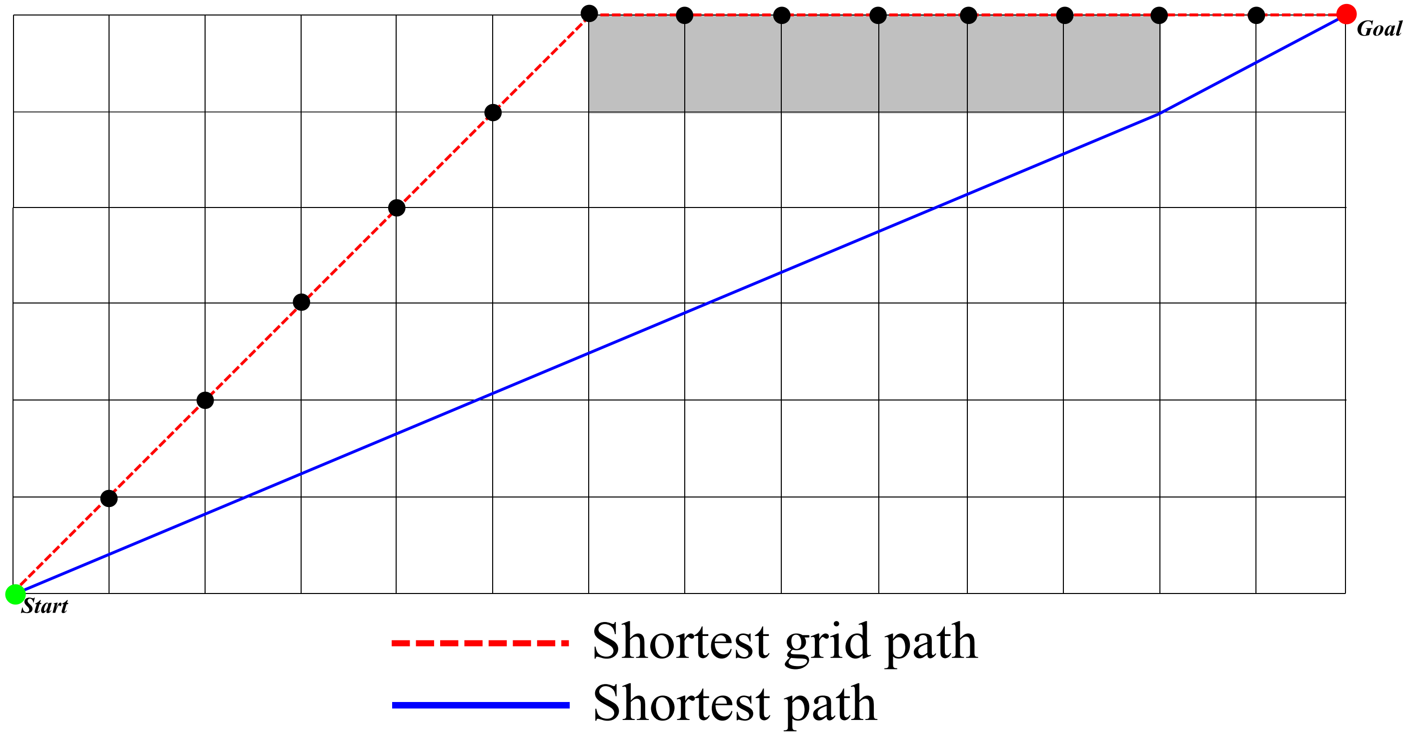
A* with Post Smoothing

- A* with Post Smoothing



A* with Post Smoothing

- A* with Post Smoothing



- Postprocessing often leaves path topology unchanged
- Better to interleave the search and the optimization

Known 2D Environments

Field D*

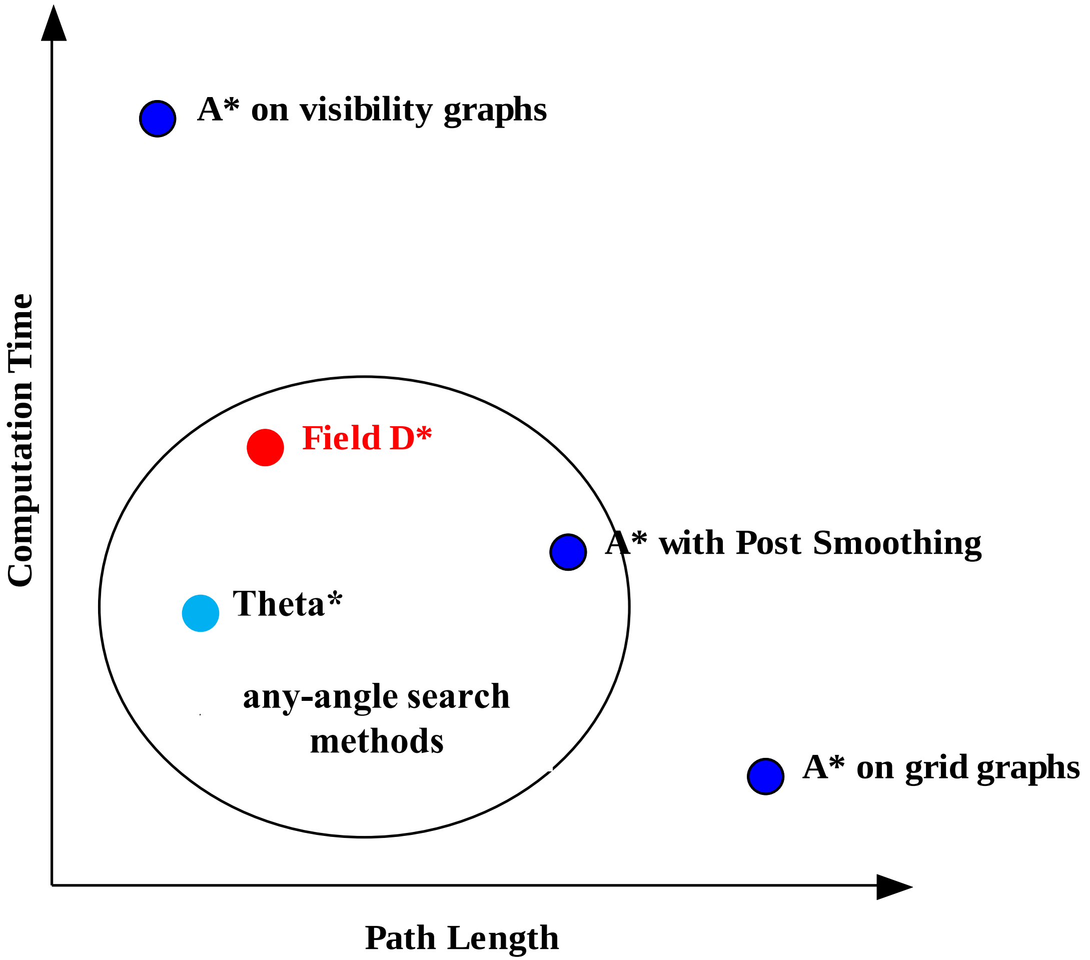
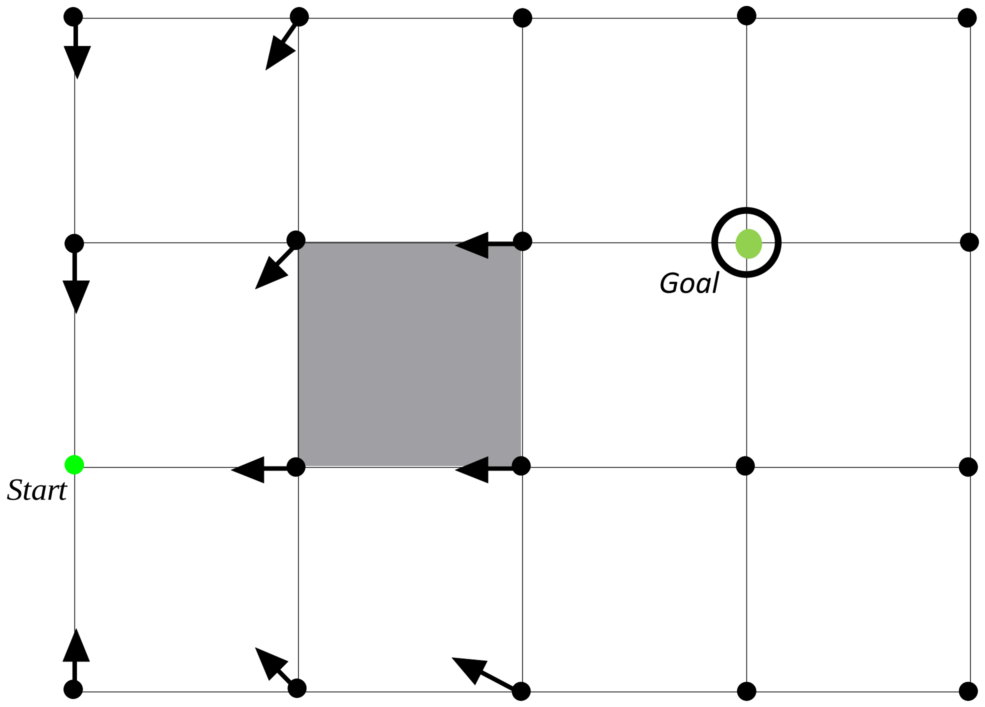


figure is notional

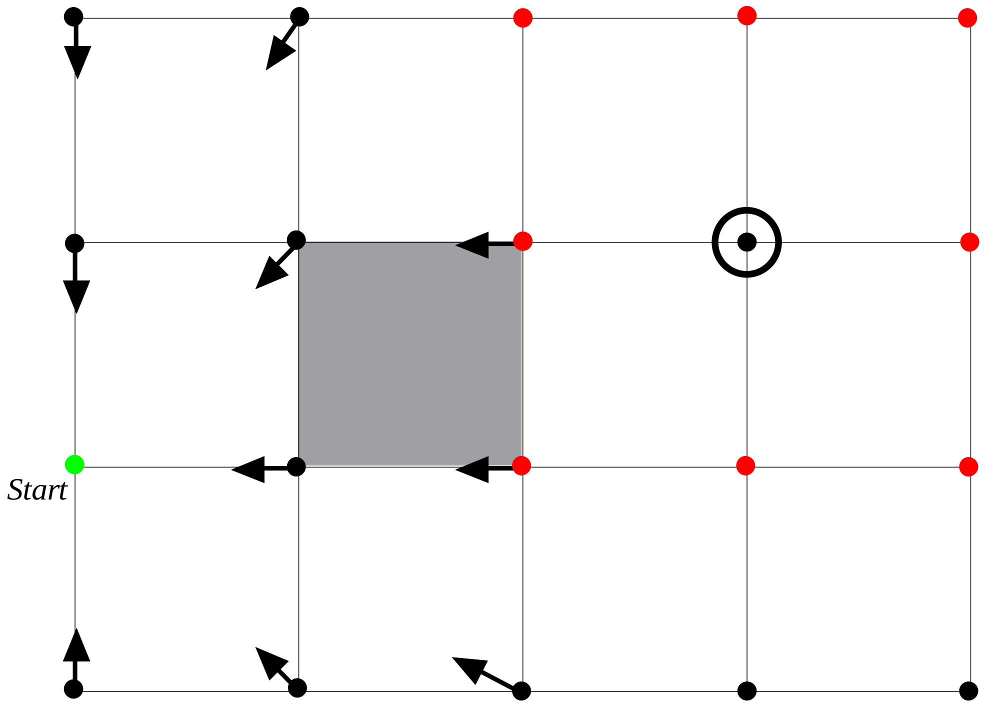
Field D*

- Field D* (a version of D* Lite) [Ferguson and Stentz, 2005]



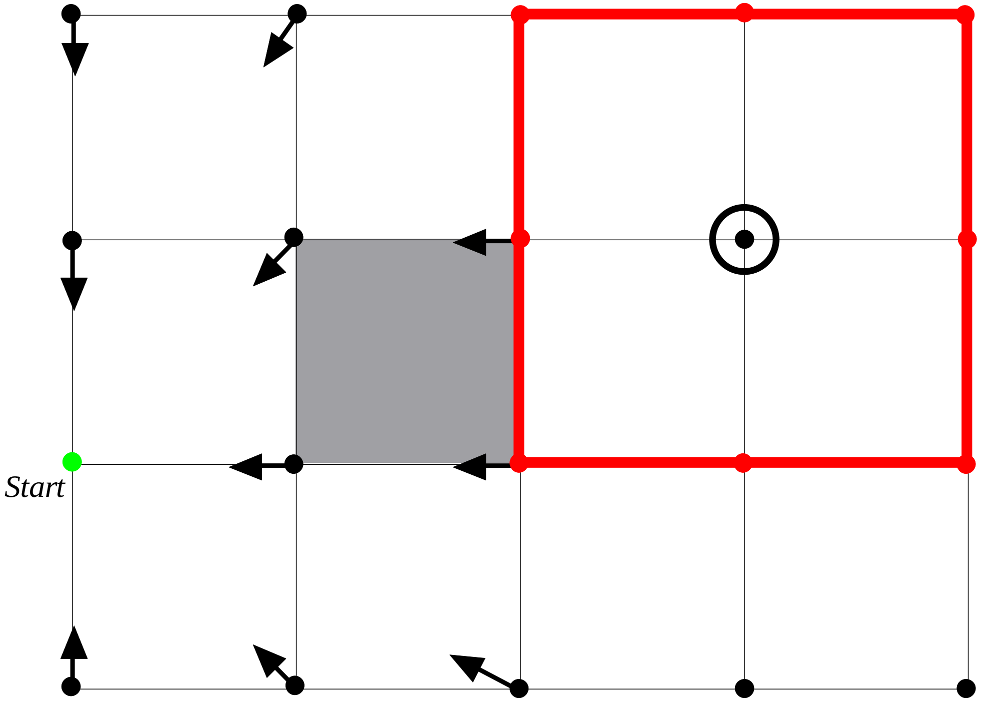
Field D*

- Field D* (a version of D* Lite)



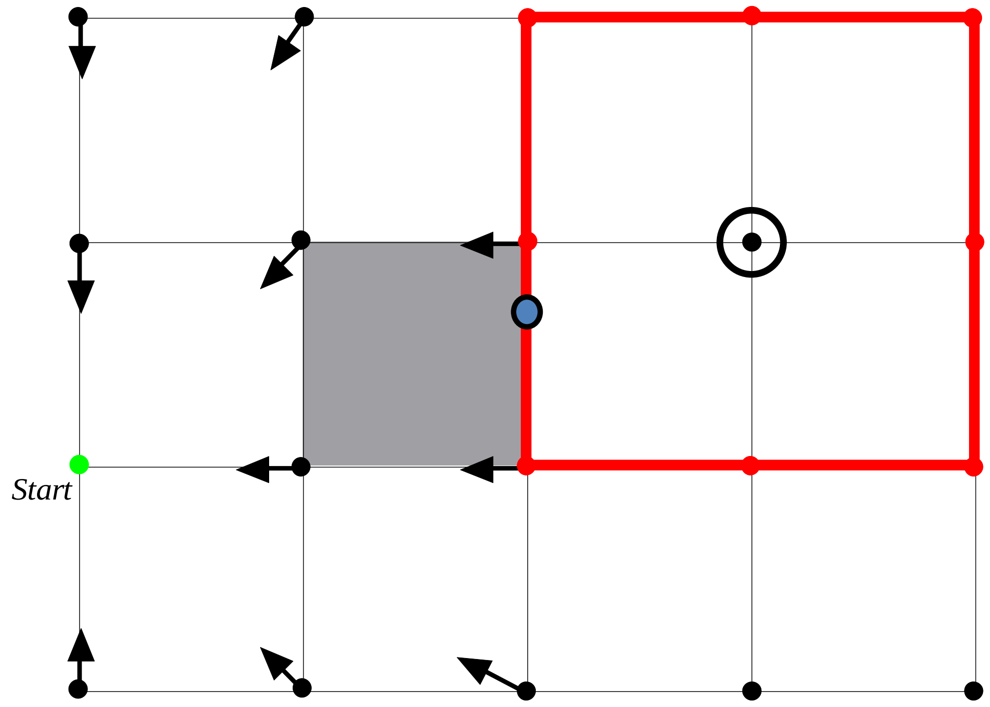
Field D*

- Field D* (a version of D* Lite)



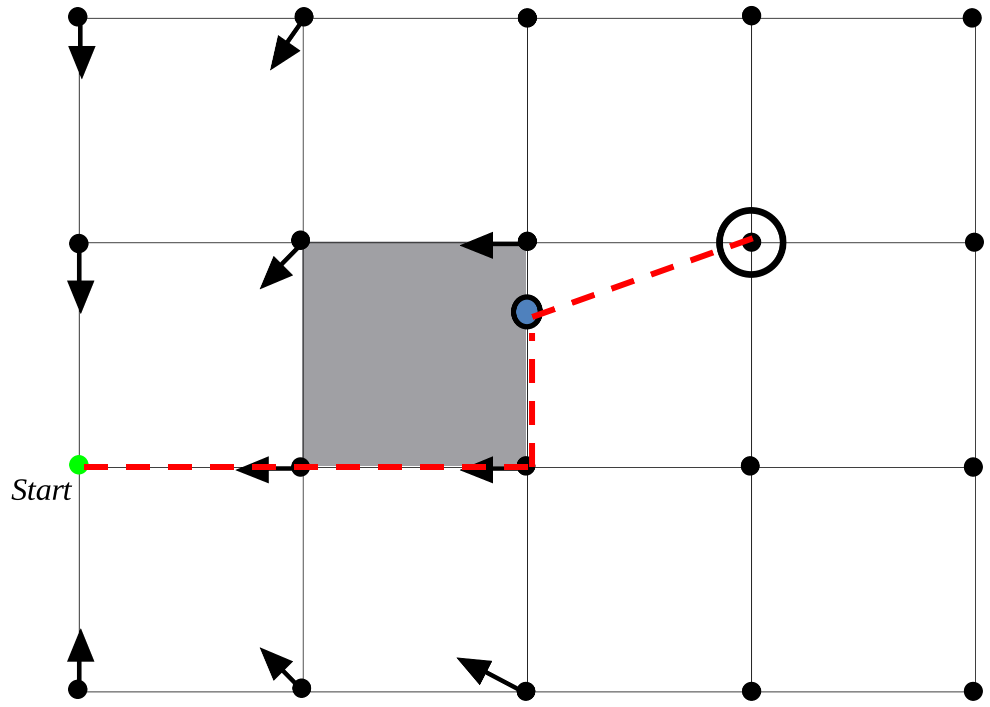
Field D*

- Field D* (a version of D* Lite)

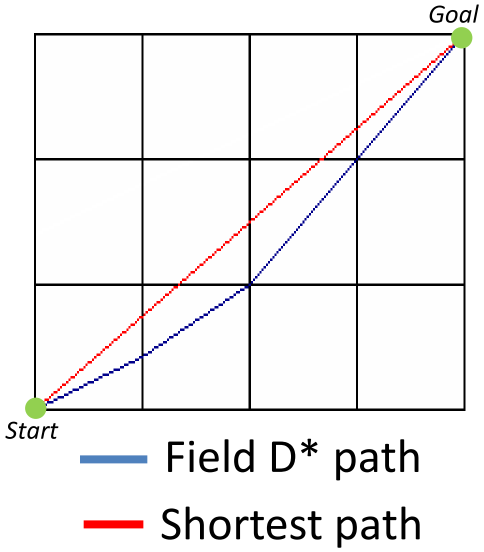


Field D*

- Field D* (a version of D* Lite)



Field D*



- Field D* (a version of D* Lite)
- Field D* can easily take traversal costs into account
- Field D* is restricted to square grids
- Sophisticated path extraction is necessary
- Paths have lots of small heading changes in open space (but could be optimized further)

Known 2D Environments

Theta*

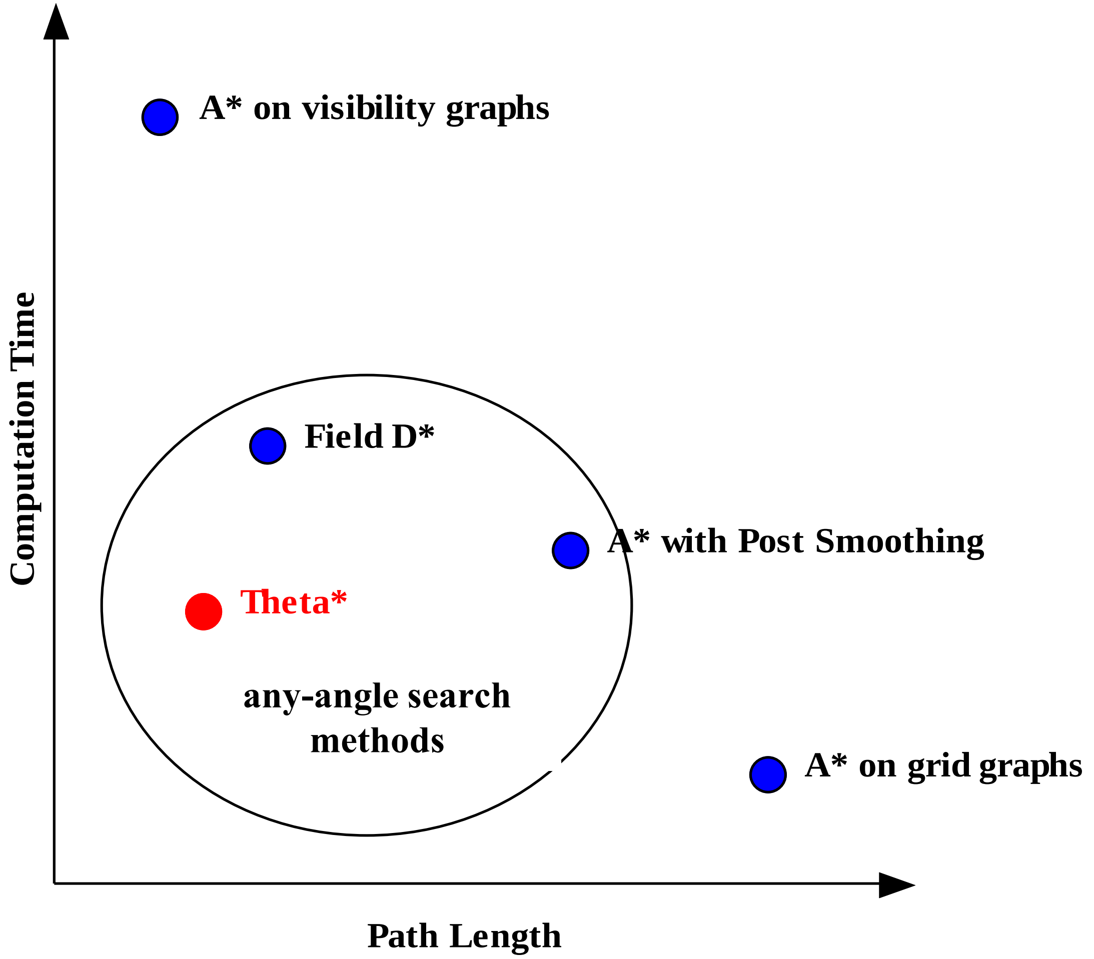
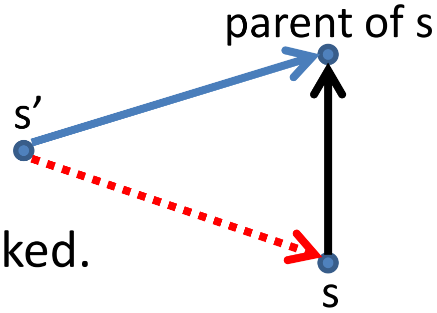


figure is notional

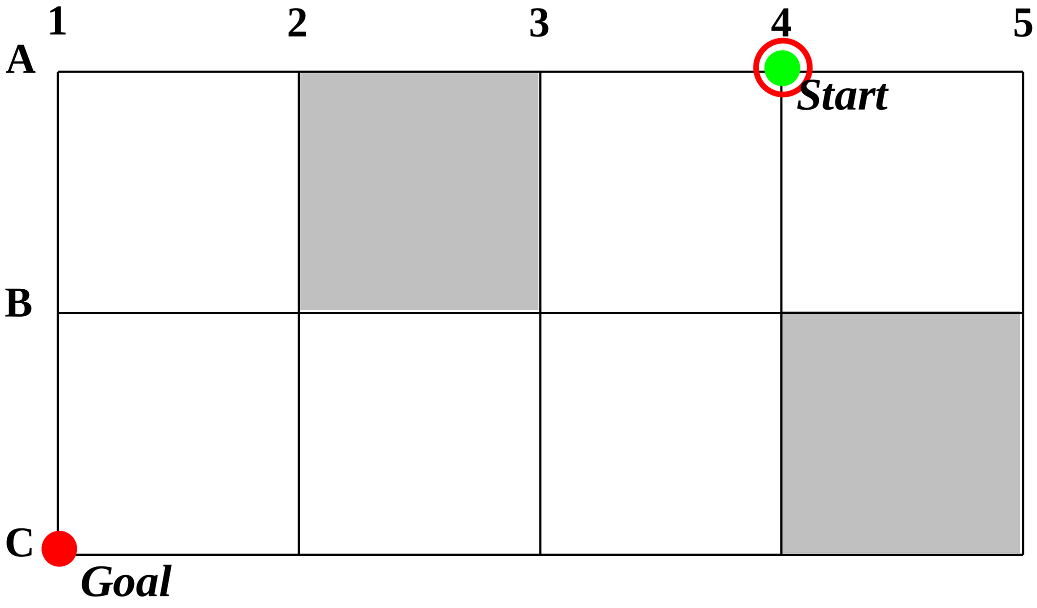
Theta*

- A*
  - The parent of a vertex has to be its neighbor in the graph.
  - When expanding vertex s and generating its neighbor s', A* considers
    - Making s the parent of s' (Path 1)
- Theta*
  - The parent of a vertex does not need to be its neighbor
  - When expanding vertex s and generating its neighbor s', Theta* considers
    - Making s the parent of s' (Path 1)
    - Making the parent of s the parent of s' (Path 2)
    - Note: Path 2 is no longer than Path 1 iff it is unblocked. The line-of-sight check can be performed with fast line-drawing algorithms from computer graphics.



Theta*

- Theta* [Nash, Daniel, Koenig and Felner, 2007]

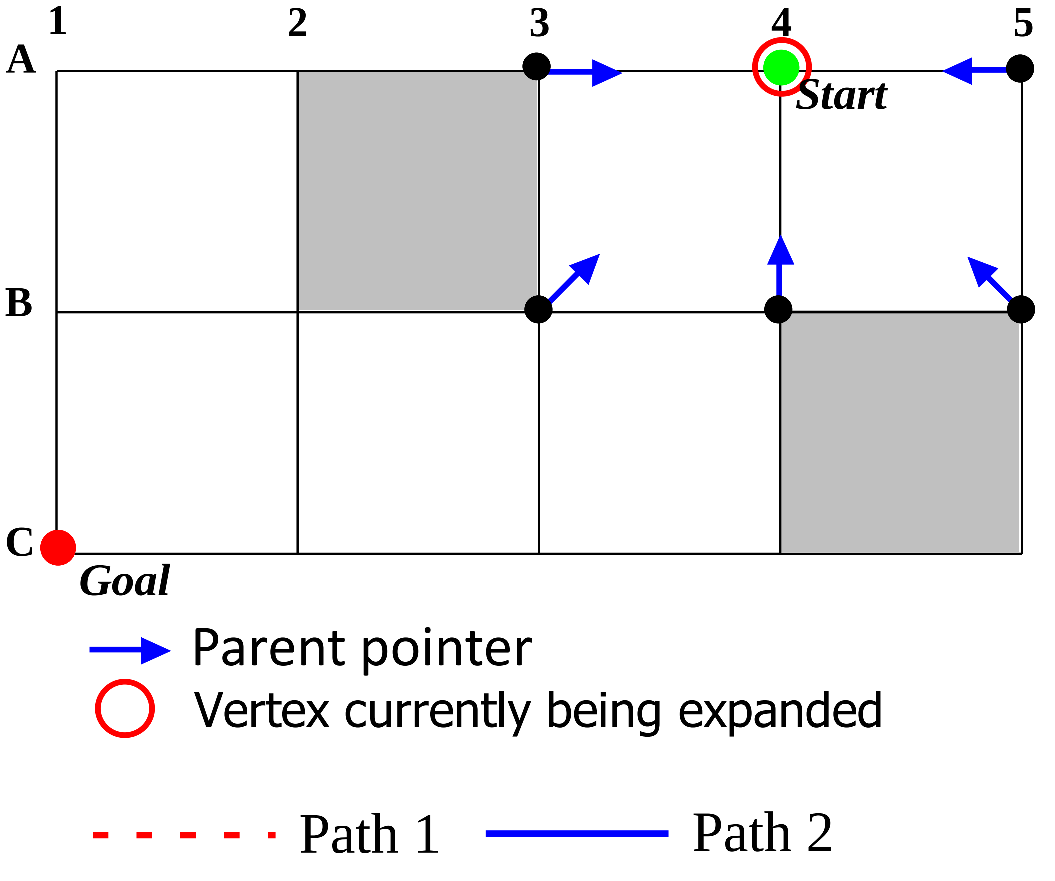


→ Parent pointer

○ Vertex currently being expanded

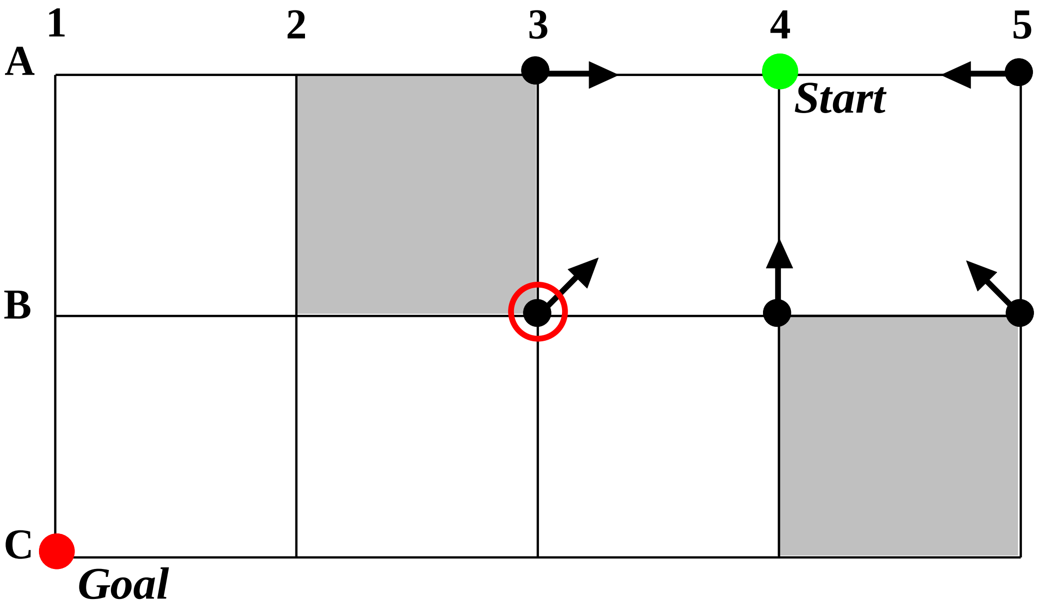
Theta*

- Theta*


1
2
3
4
5
A
B
C
Start
Goal
Parent pointer
Vertex currently being expanded
Path 1
Path 2

Theta*

- Theta*



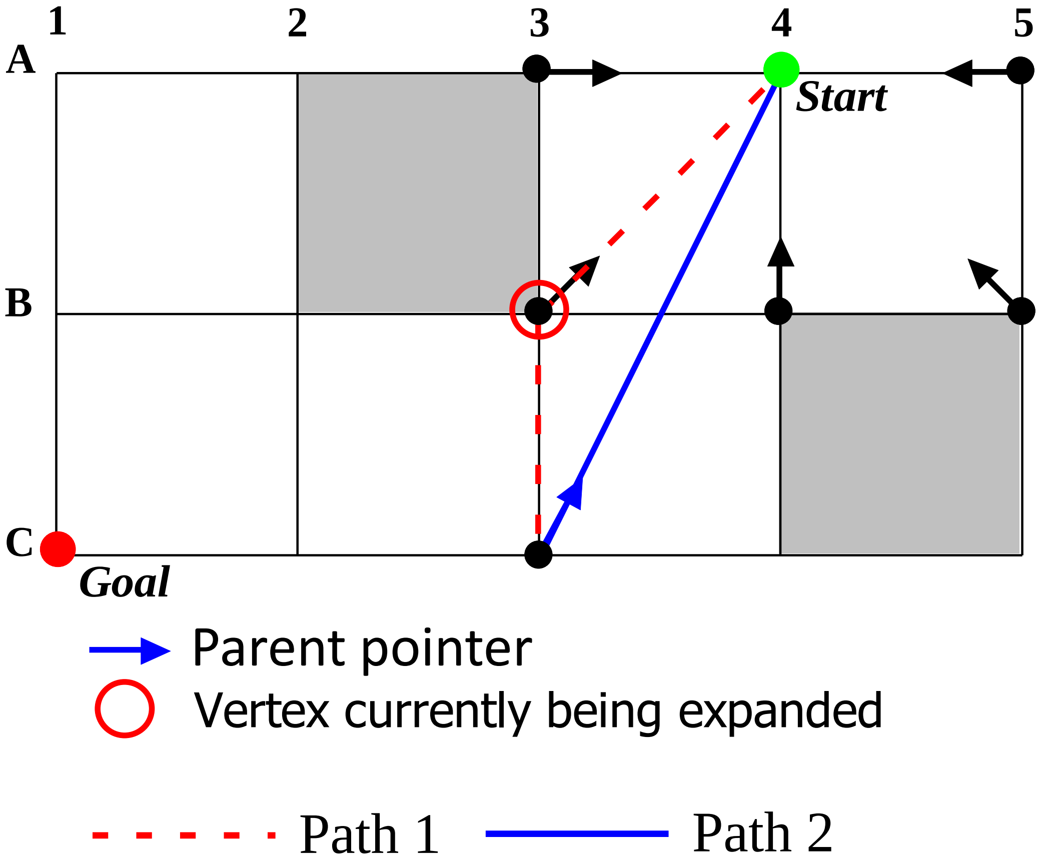
→ Parent pointer

○ Vertex currently being expanded

- - - - - Path 1 ——— Path 2

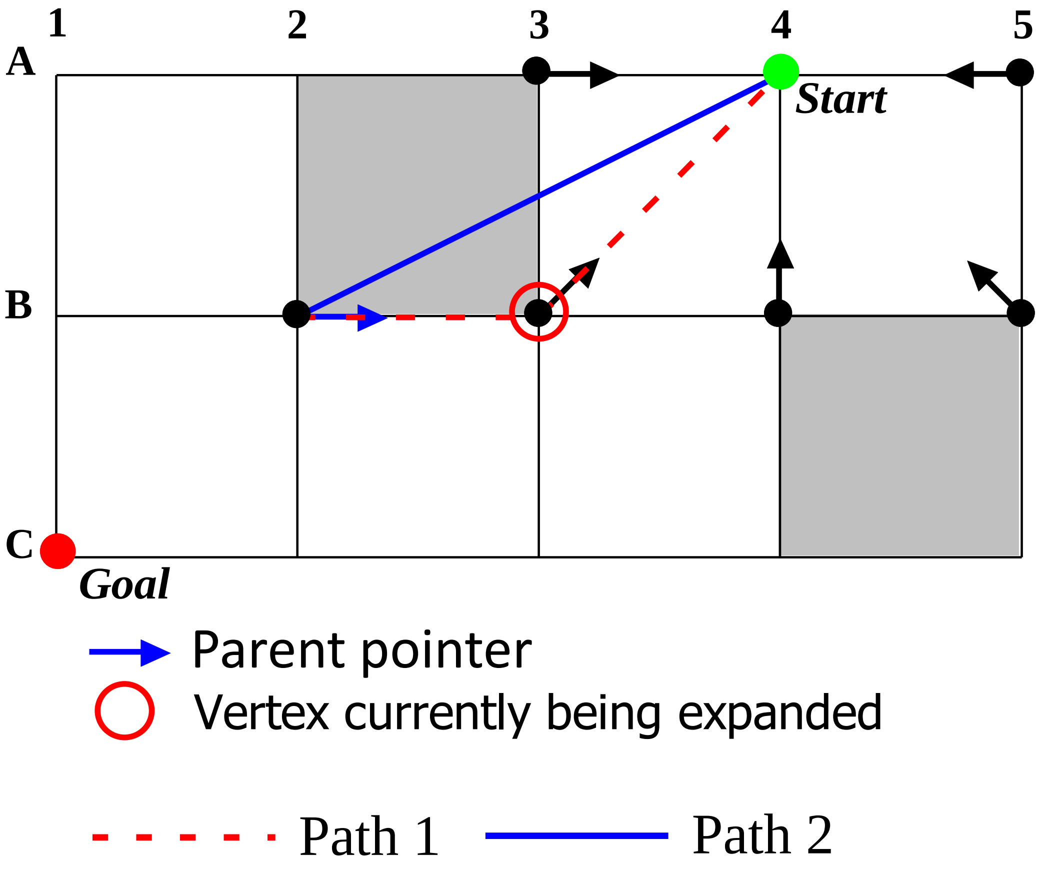
Theta*

- Theta*


1
2
3
4
5
A
B
C
Start
Goal
Parent pointer
Vertex currently being expanded
Path 1
Path 2

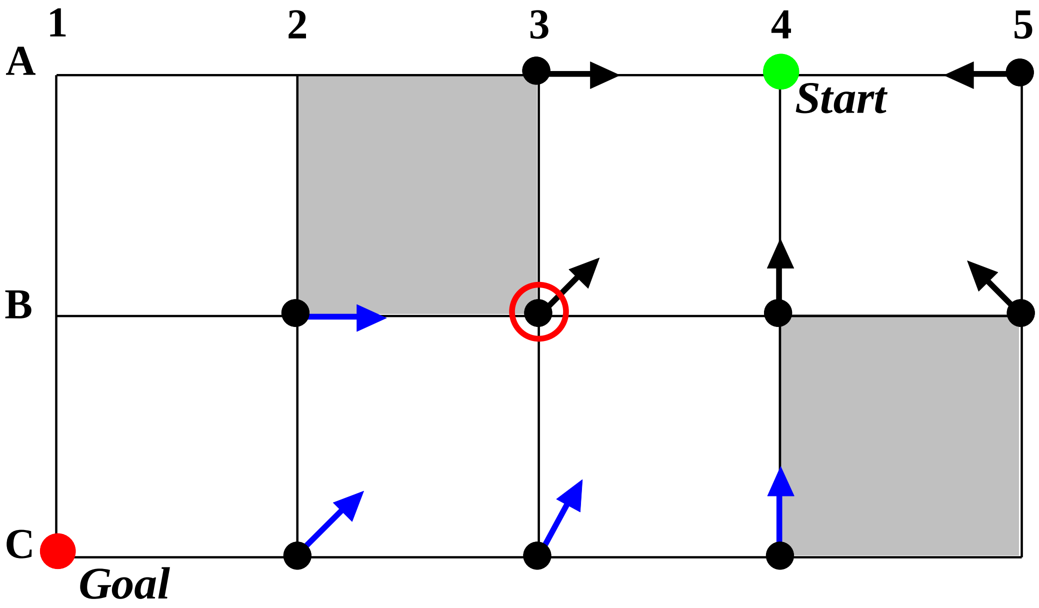
Theta*

- Theta*


1
2
3
4
5
A
B
C
Start
Goal
Parent pointer
Vertex currently being expanded
Path 1
Path 2

Theta*

- Theta*



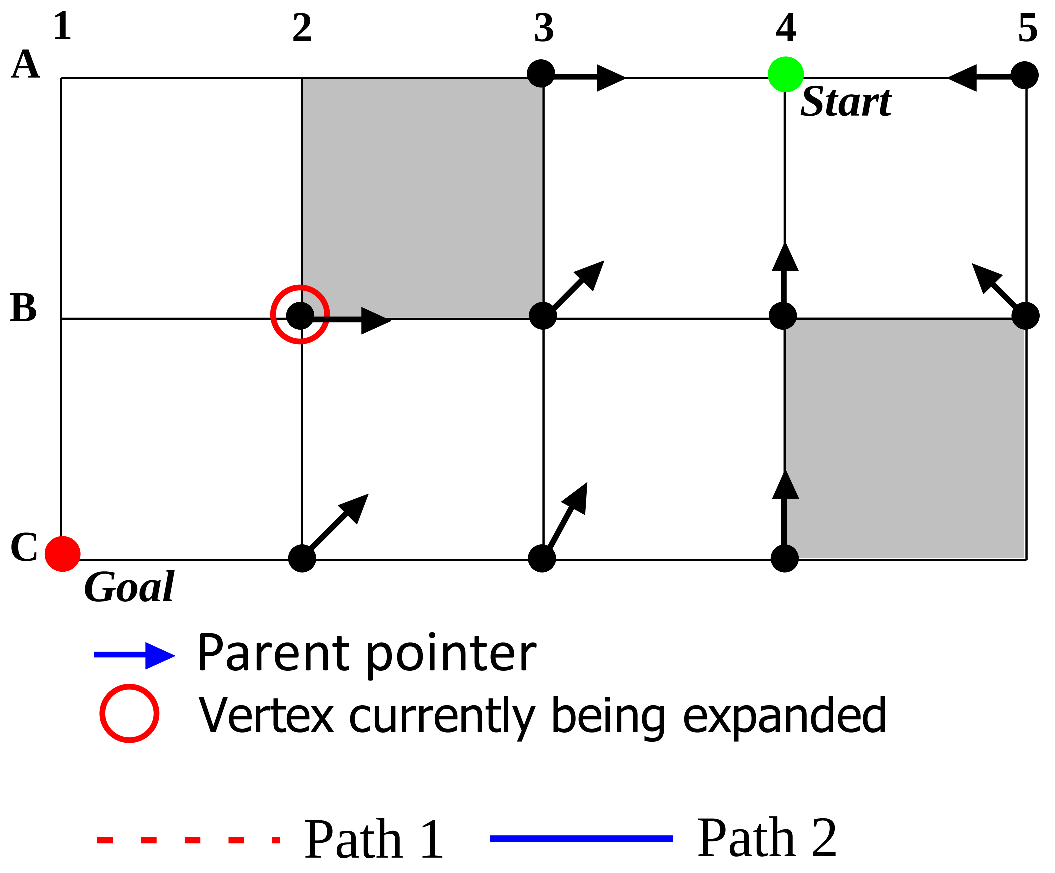
→ Parent pointer

○ Vertex currently being expanded

- - - - - Path 1 ——— Path 2

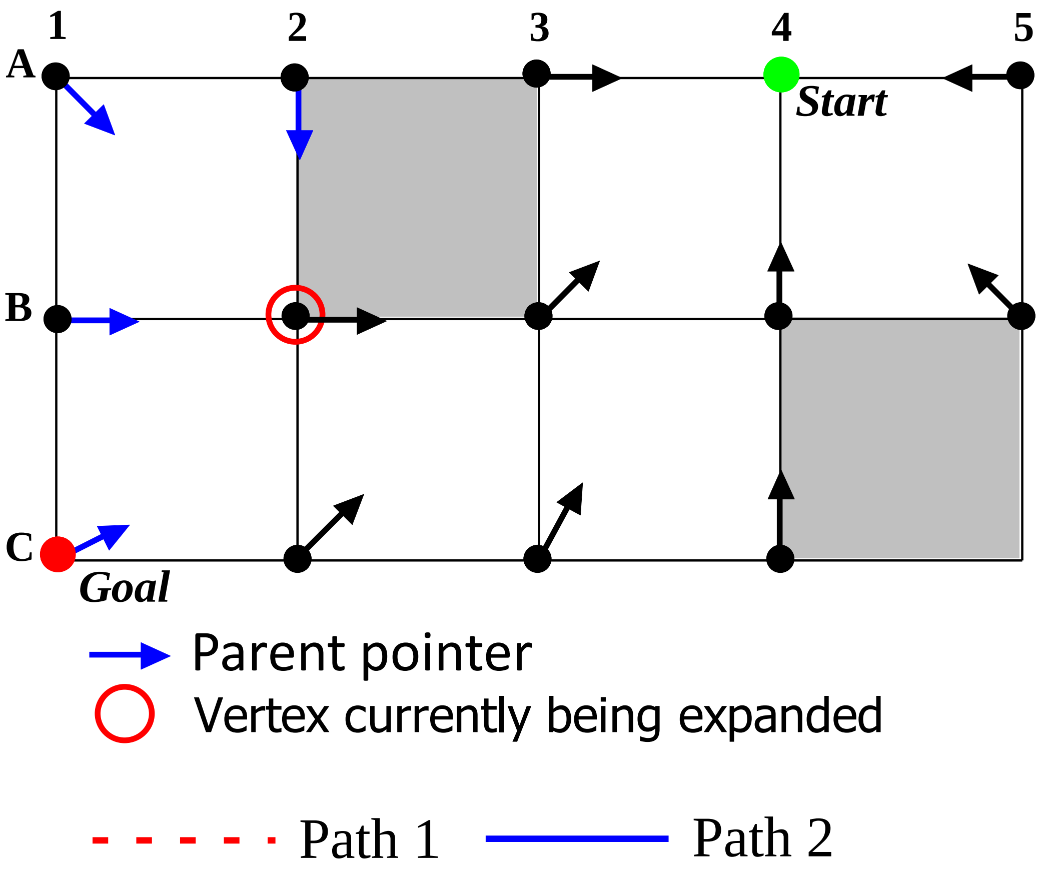
Theta*

- Theta*


1
2
3
4
5
A
B
C
Start
Goal
Parent pointer
Vertex currently being expanded
Path 1
Path 2

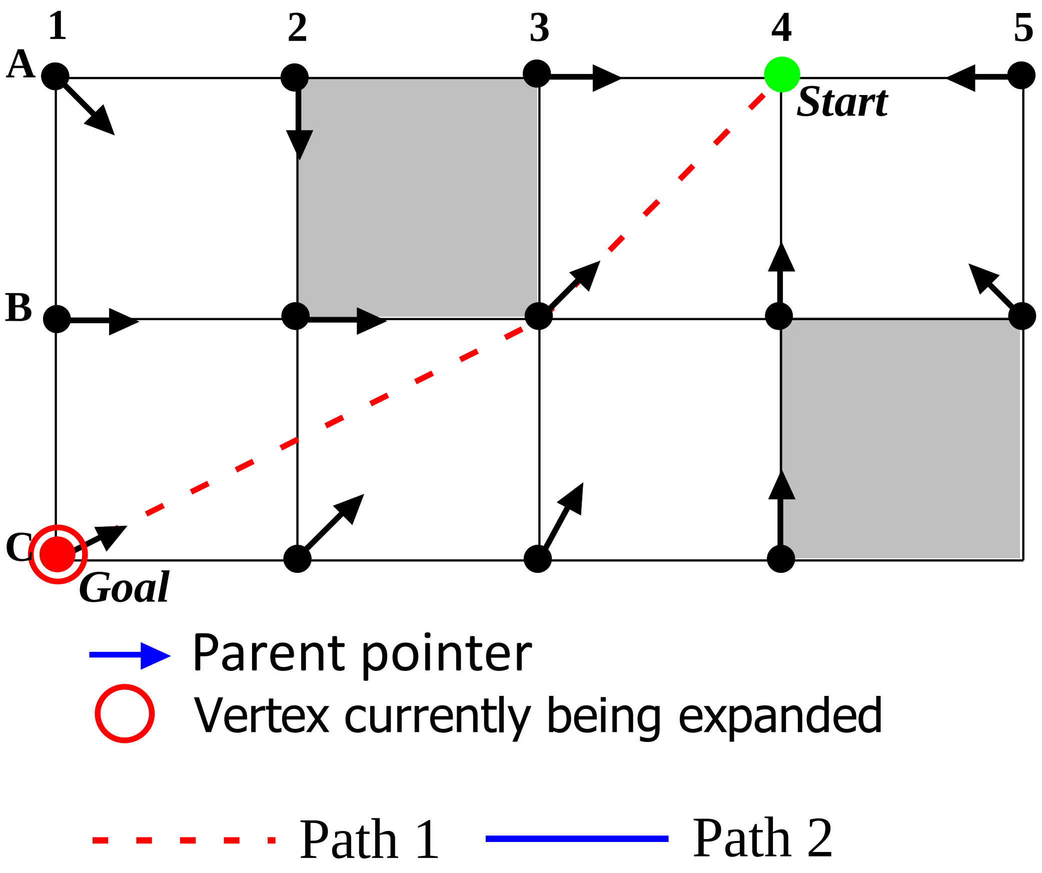
Theta*

- Theta*


1
2
3
4
5
A
B
C
Start
Goal
Parent pointer
Vertex currently being expanded
Path 1
Path 2

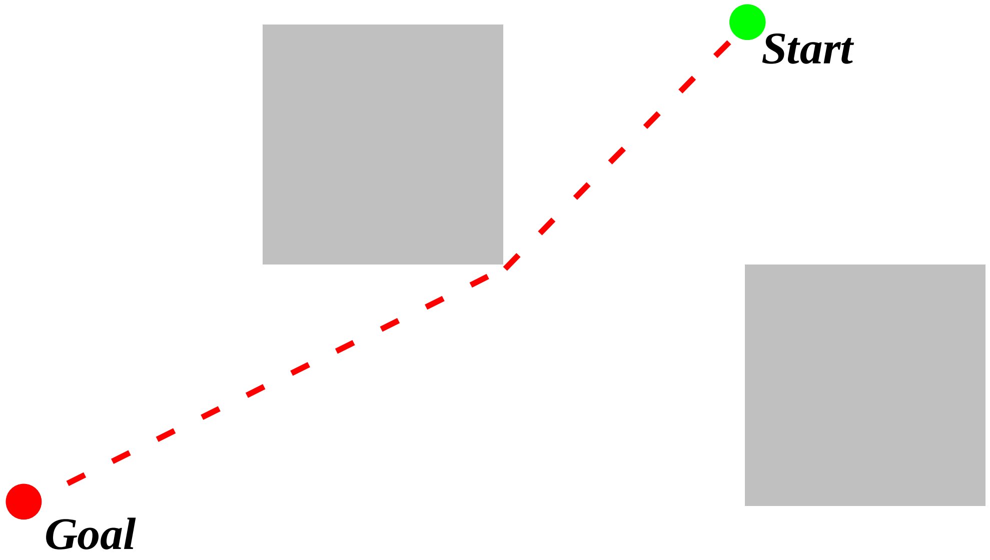
Theta*

- Theta*


1
2
3
4
5
A
B
C
Start
Goal
Parent pointer
Vertex currently being expanded
Path 1
Path 2

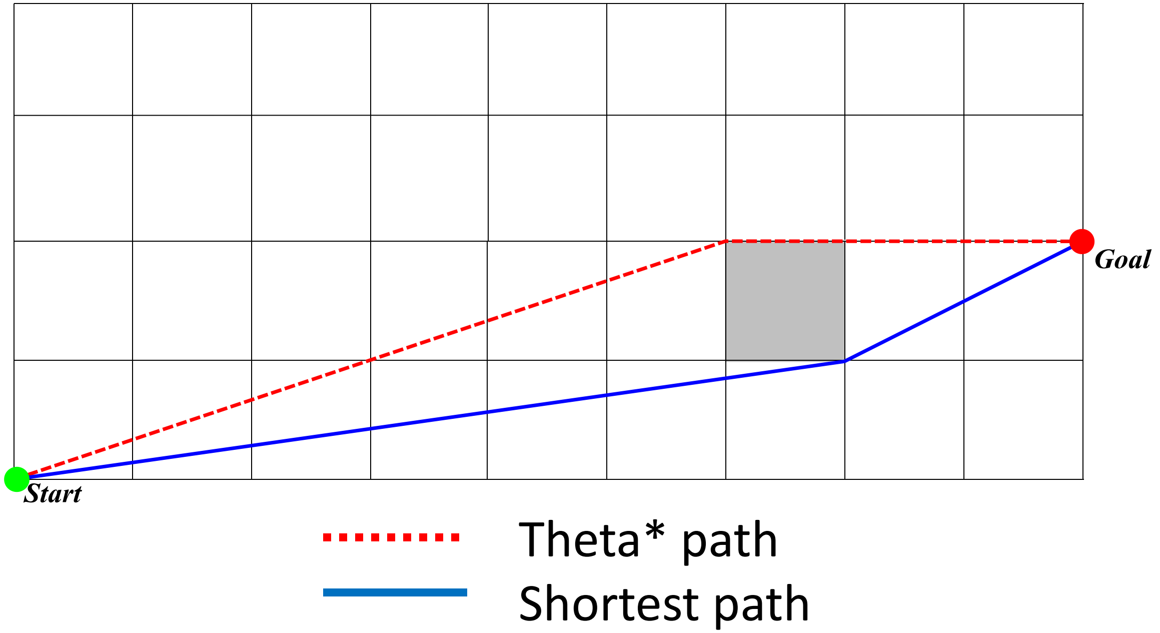
Theta*

- Theta*



Path Length

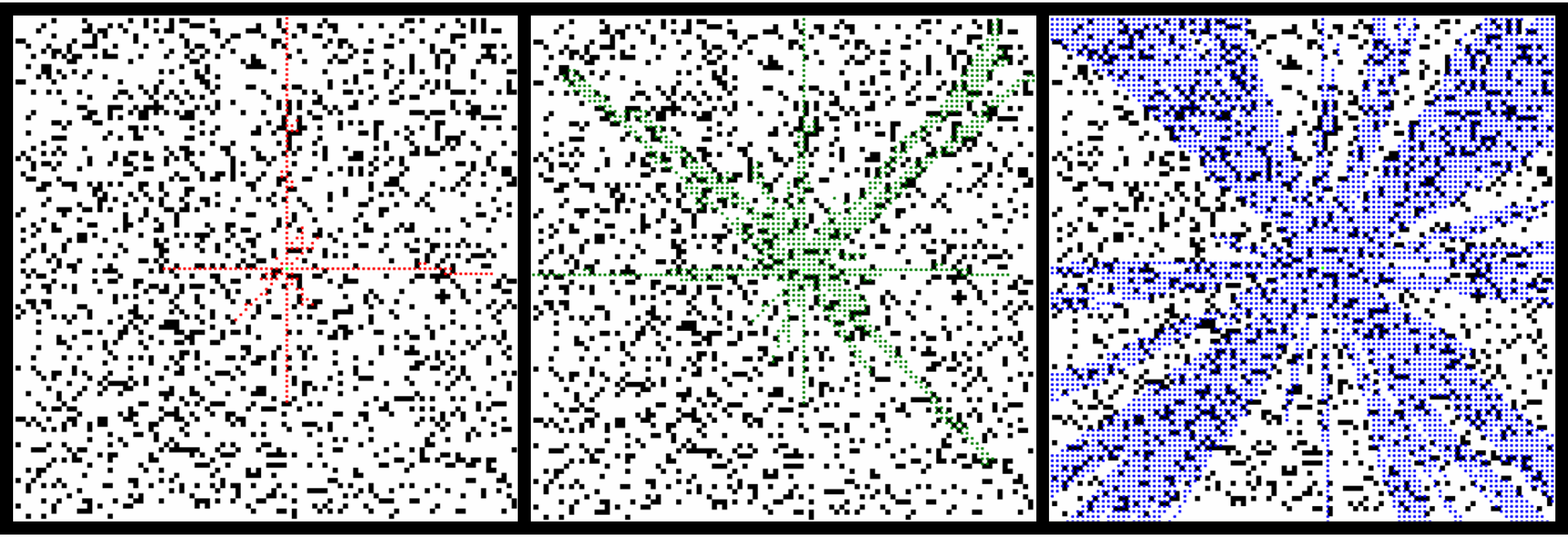
- Theta* is not guaranteed to find shortest paths since the parent of a vertex can only be a neighbor of the vertex or the parent of a neighbor



- The length of the path is still within 0.2% of optimal

Path Lengths

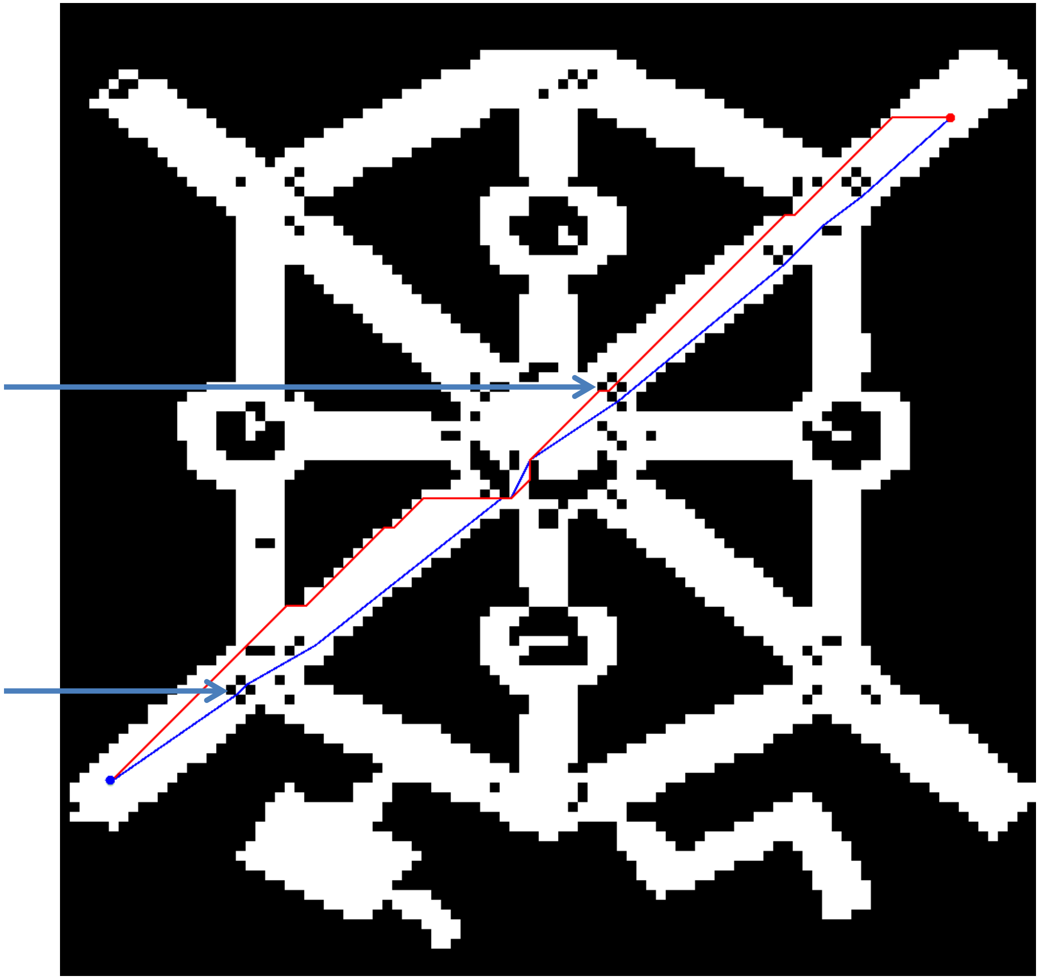
- Goal vertices to which a shortest path was found from the center vertex



🟥 Field D* 🟩 A* with Post Smoothing 🟦 Theta*

Path Length

We allow paths to pass through diagonally touching blocked cells here but this an be changed easily in the code



A* on grid graphs

Theta*

Experimental Setups

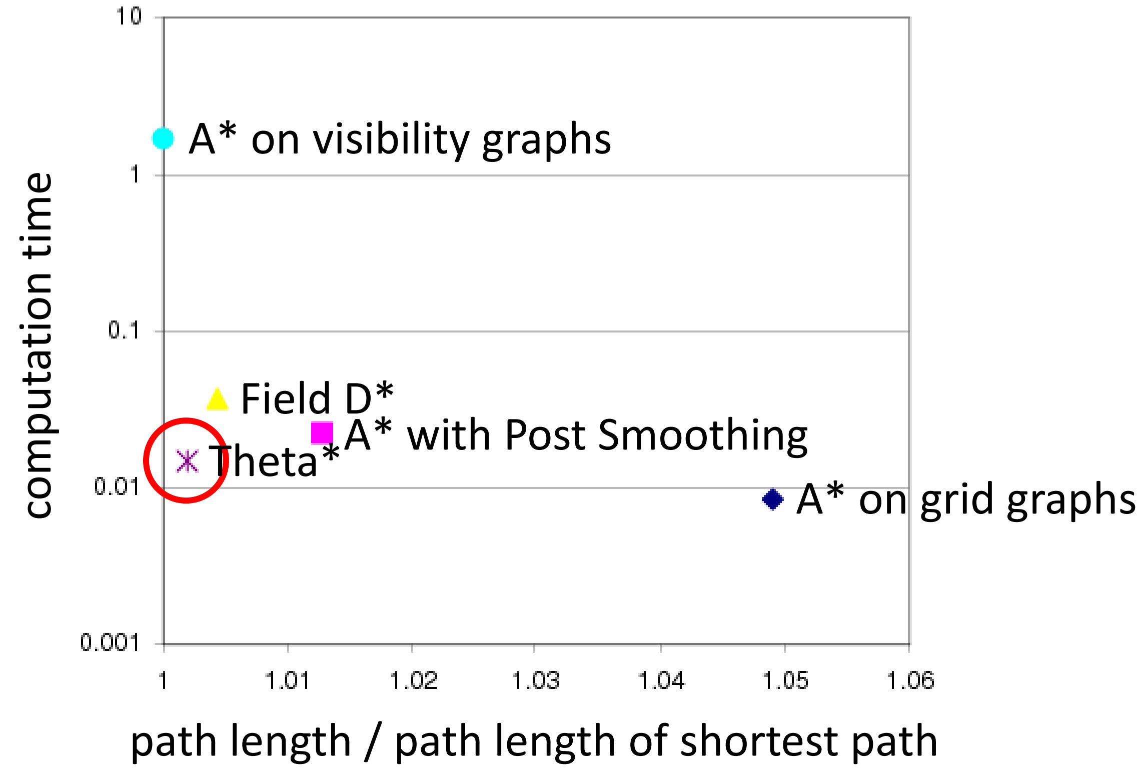
- For A* on grid graphs, we use the octile heuristics as h-values. For A* on visibility graphs, A* with Post Smoothing (= helps smoothing), Field D*, Theta* and Lazy Theta*, we use the straight-line distances as h-values.
- For A* on grid graphs, A* on visibility graphs, A* with Post Smoothing and Field D*, we break ties among vertices with the same f-values in favor of vertices with smaller g-values. For Theta* and Lazy Theta*, we break ties among vertices with the same f-values in favor of vertices with smaller g-values.

Experimental Setups

- We place the start vertex in the bottom left corner, the goal vertex randomly into the rightmost column of cells, and a one unit border of unblocked grid cells around the grid to ensure a path exists.
- Other experimental setups have been used as well, which is important since the setup used here constrains the search space to the left and bottom, [Yap (2011), personal communication]. One could, for example, use randomly selected start and goal vertices on game maps with some randomly blocked cells added that simulate structures built by players [Yap et al. (2011)].

Efficiency

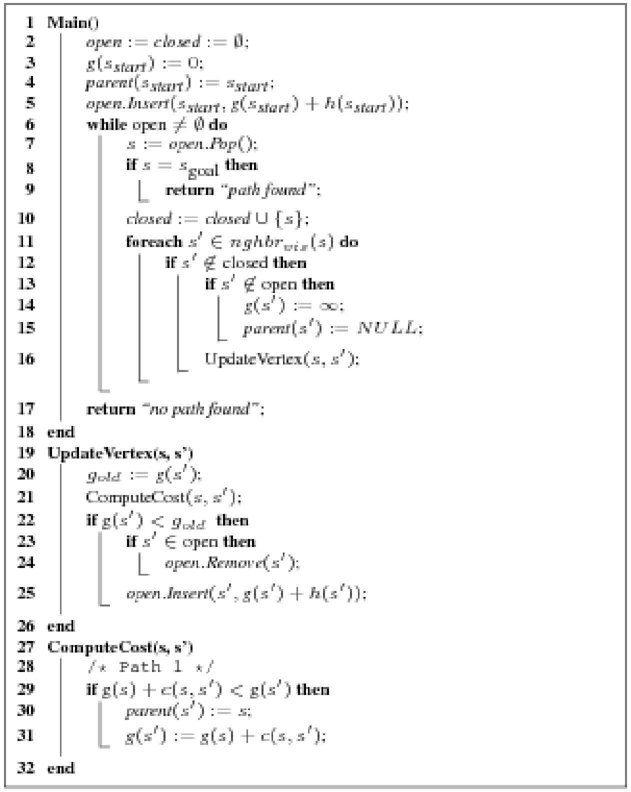
- 100 x 100 grids with 20% randomly blocked cells



the computation times and their relationship depend on implementation details (such as how the priority queue and line-of-sight checks are implemented)

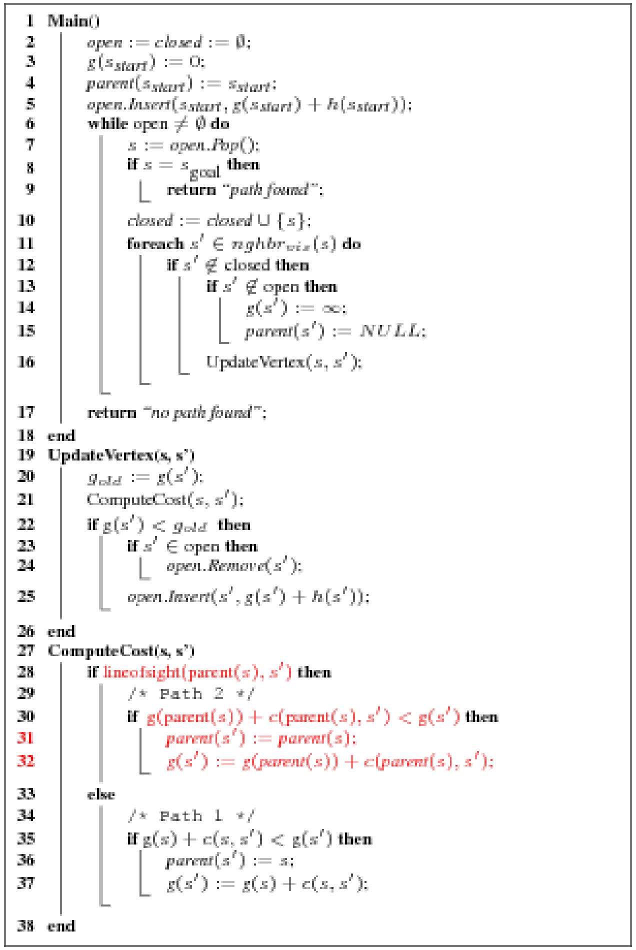
Known 2D Environments

Simplicity



```
Main()
    $open := closed := \emptyset$;
    $g(s_{start}) := 0$;
    $parent(s_{start}) := s_{start}$;
    $open.Insert(s_{start}, g(s_{start}) + h(s_{start}))$;
    while $open \neq \emptyset$ do
        $s := open.Pop()$;
        if $s = s_{goal}$ then
            return "path found";
        $closed := closed \cup \{s\}$;
        foreach $s' \in nghbr_{vis}(s)$ do
            if $s' \notin closed$ then
                if $s' \notin open$ then
                    $g(s') := \infty$;
                    $parent(s') := NULL$;
                UpdateVertex($s, s'$);
    return "no path found";
end
UpdateVertex(s, s')
    $g_{old} := g(s')$;
    ComputeCost($s, s'$);
    if $g(s') < g_{old}$ then
        if $s' \in open$ then
            $open.Remove(s')$;
        $open.Insert(s', g(s') + h(s'))$;
end
ComputeCost(s, s')
    /* Path 1 */
    if $g(s) + c(s, s') < g(s')$ then
        $parent(s') := s$;
        $g(s') := g(s) + c(s, s')$;
end
```

A*



```
Main()
    $open := closed := \emptyset$;
    $g(s_{start}) := 0$;
    $parent(s_{start}) := s_{start}$;
    $open.Insert(s_{start}, g(s_{start}) + h(s_{start}))$;
    while $open \neq \emptyset$ do
        $s := open.Pop()$;
        if $s = s_{goal}$ then
            return "path found";
        $closed := closed \cup \{s\}$;
        foreach $s' \in nghbr_{vis}(s)$ do
            if $s' \notin closed$ then
                if $s' \notin open$ then
                    $g(s') := \infty$;
                    $parent(s') := NULL$;
                UpdateVertex($s, s'$);
    return "no path found";
end
UpdateVertex(s, s')
    $g_{old} := g(s')$;
    ComputeCost($s, s'$);
    if $g(s') < g_{old}$ then
        if $s' \in open$ then
            $open.Remove(s')$;
        $open.Insert(s', g(s') + h(s'))$;
end
ComputeCost(s, s')
    if lineofsight($parent(s), s'$) then
        /* Path 2 */
        if $g(parent(s)) + c(parent(s), s') < g(s')$ then
            $parent(s') := parent(s)$;
            $g(s') := g(parent(s)) + c(parent(s), s')$;
    else
        /* Path 1 */
        if $g(s) + c(s, s') < g(s')$ then
            $parent(s') := s$;
            $g(s') := g(s) + c(s, s')$;
end
```

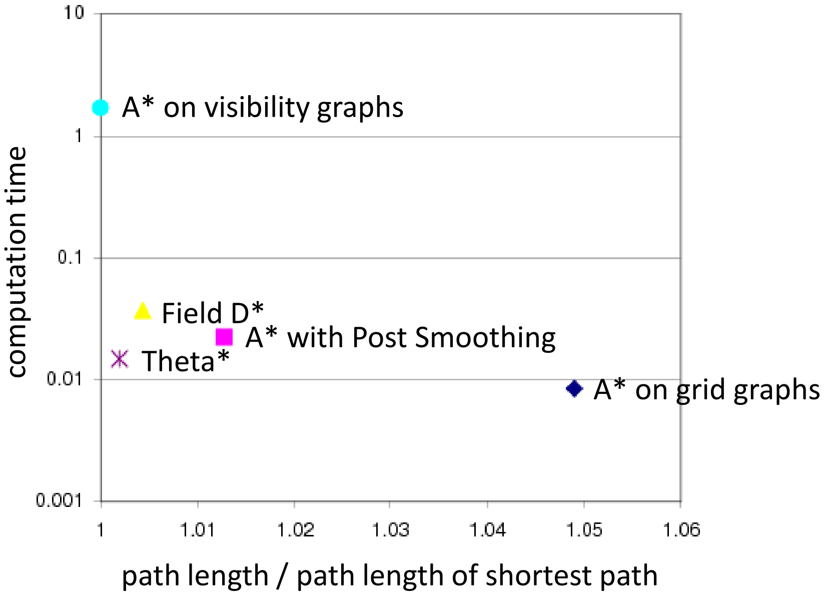
Theta*

Simplicity

- A class project on any-angle search with Theta* was developed as part of the “Computer Games in the Classroom” initiative, see idm-lab.org/gameai
- Used at
  - University of Nevada, Reno
  - University of Southern California
- Online tutorial
  - AiGameDev.com

Any-Angle Search

Property 1: Efficiency



Any-Angle Search Method	Property 2: Simplicity	Property 3: Generality (Any Euclidean Graph)
A* with Post Smoothing	Yes	Yes
Field D*	No	No
Theta*	Yes	Yes

Table of Contents

- Introduction
- Analysis of Path Lengths
- Any-Angle Search Methods
  - Known 2D Environments
  - Known 3D Environments
  - Unknown 2D Environments (1 slide only)
- Conclusion

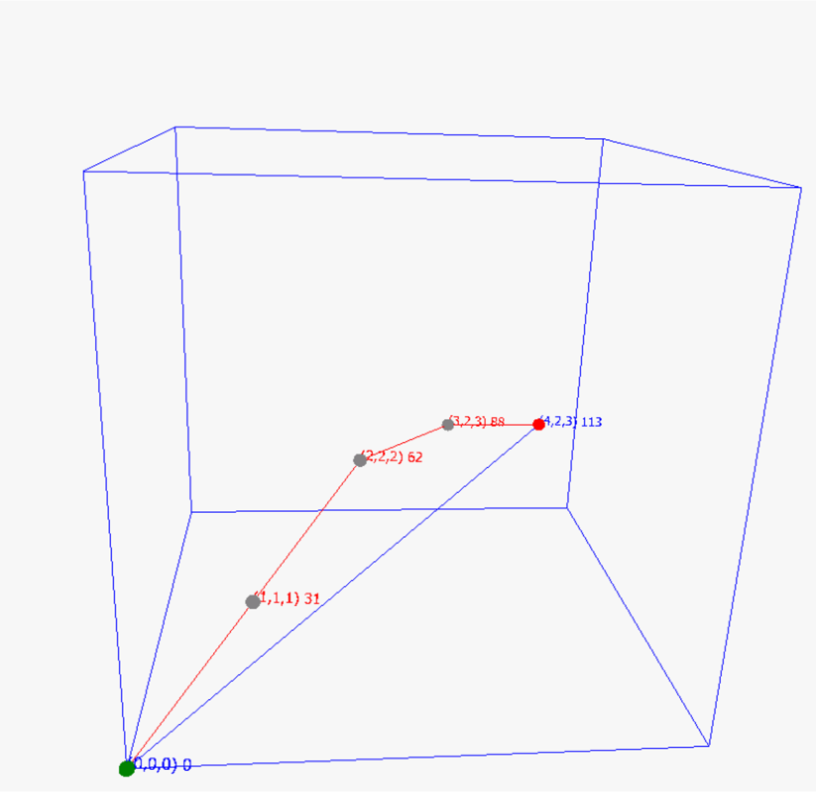
A* on Grid Graphs

- Paths get longer and more unrealistic looking

Dimension	Regular Grid	Neighbors	% Longer Than Shortest Path
2D	triangular grid	3-neighbor	≈ 100
		6-neighbor	≈ 15
	square grid	4-neighbor	≈ 41
		8-neighbor	≈ 8
	hexagonal grid	6-neighbor	≈ 15
		12-neighbor	≈ 4
3D	cubic grid	6-neighbor	≈ 73
		26-neighbor	≈ 13

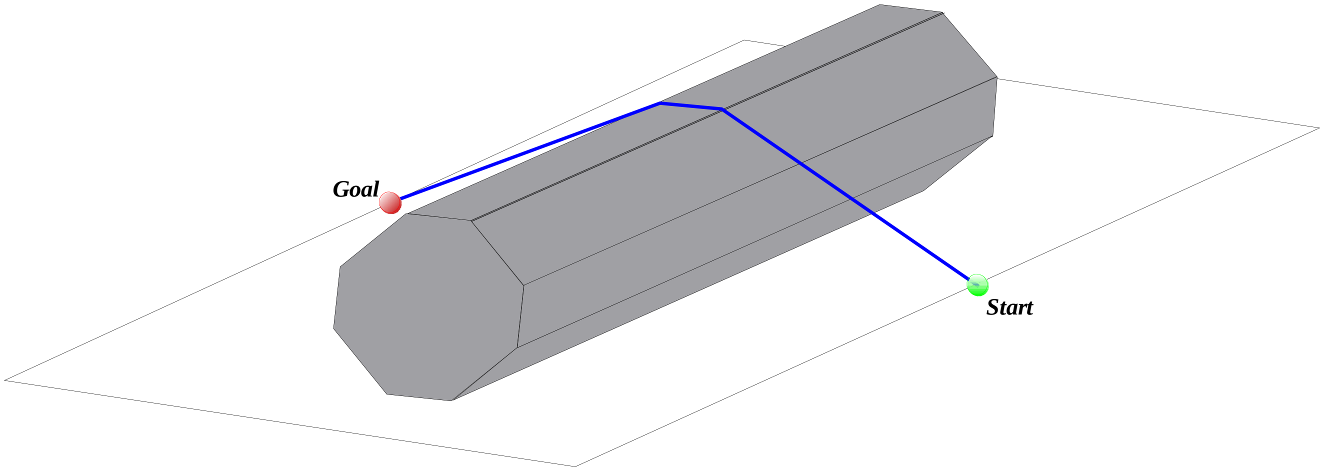
A* on Grid Graphs

- Paths get longer and more unrealistic looking



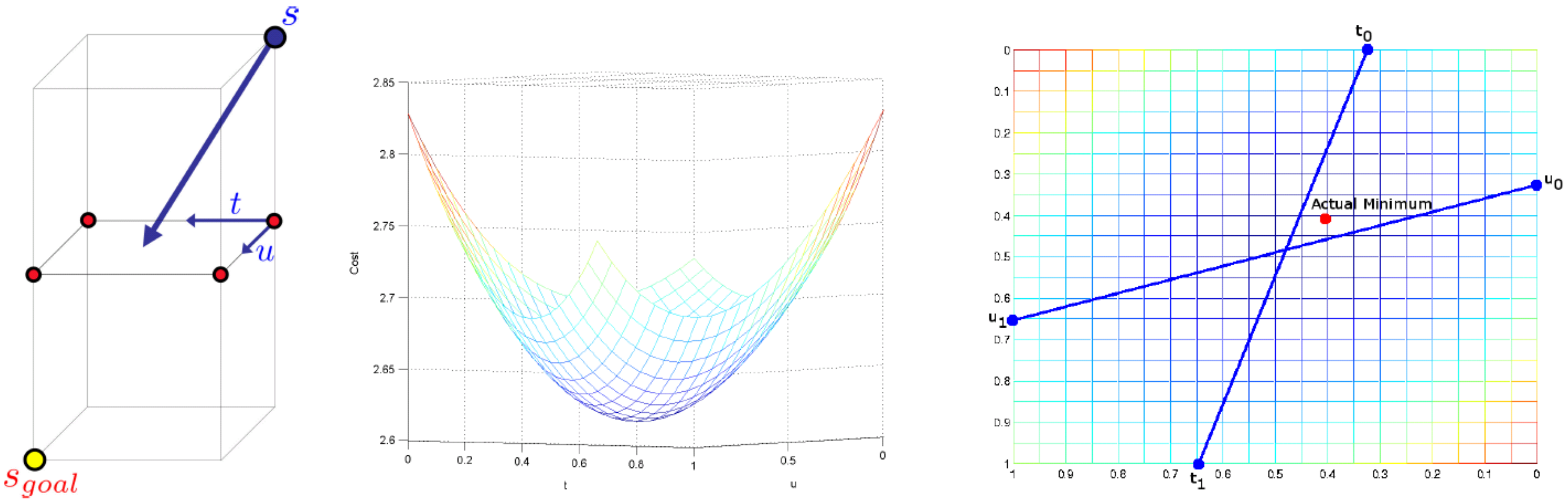
A* on Visibility Graphs

- Paths are no longer optimal



Field D*

- There is a 3D version of Field D* [Carsten, Ferguson and Stentz (2006)], which is more complex than the 2D version, specific to cubic grids, and cannot solve the optimization in closed form

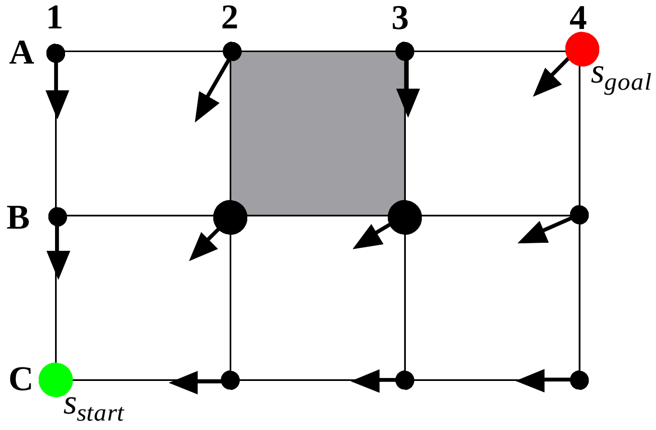


(a) Planning Problem (b) Cost Surface (c) Approximate Solution

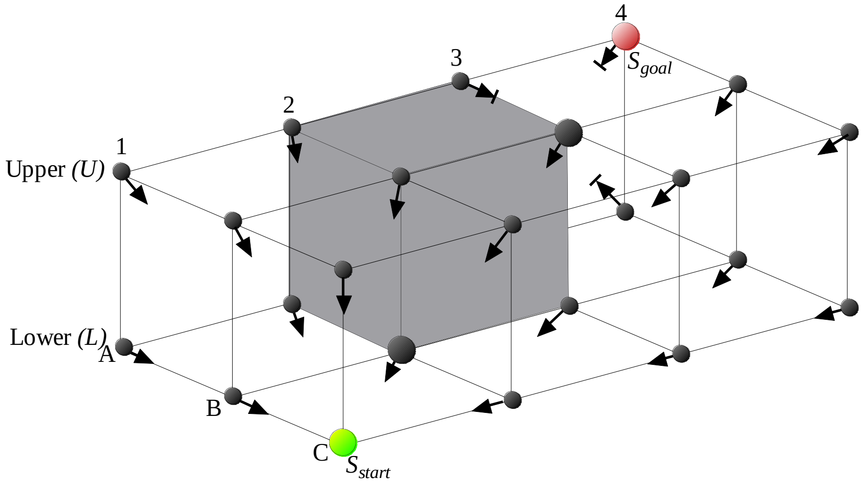
figures from the cited paper

Theta*

- Theta* applies to 3D environments without any modifications but the number of line-of-sight checks increases since there is one line-of-sight check for each unexpanded visible neighbor (that is, potentially 26 instead of 8 neighbors)



3+7+7=17 line-of-sight checks



7+15+15=37 line-of-sight checks

Lazy Theta*

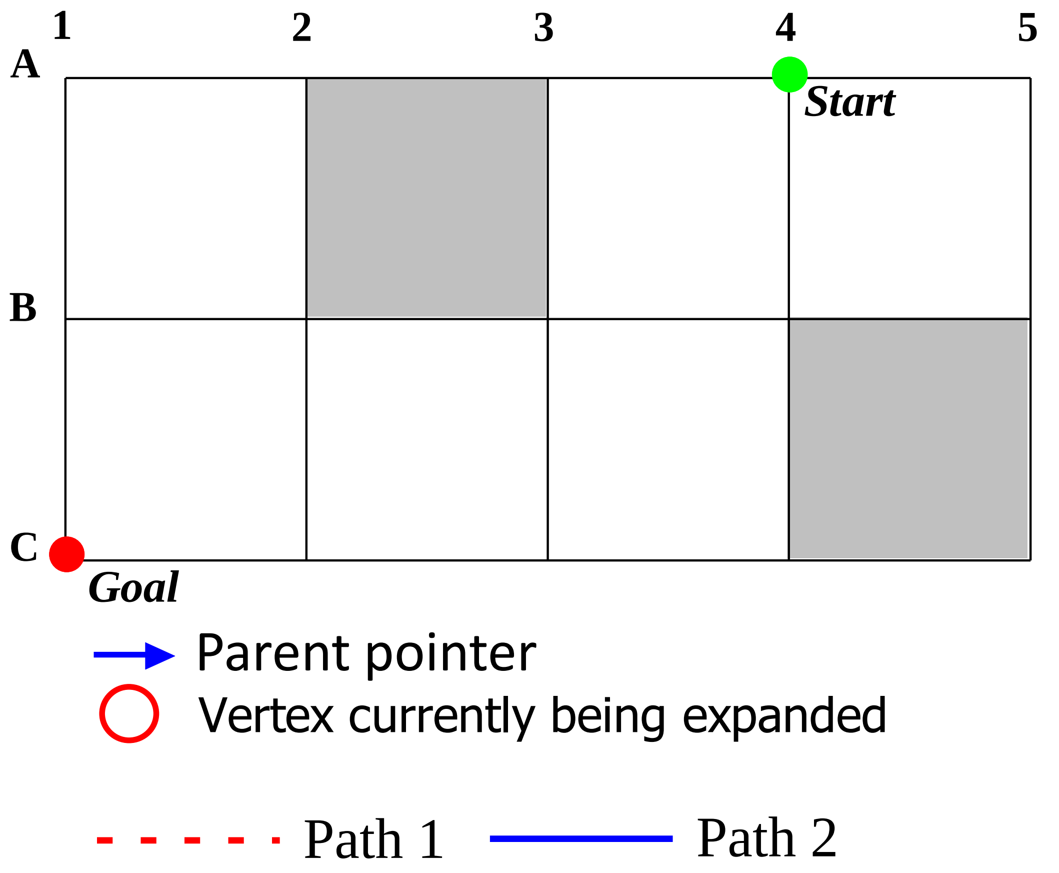
- Lazy Theta* [Nash, Koenig and Tovey (2010)] performs one line-of-sight check only for each expanded vertex rather than each generated vertex
- Lazy Theta* works in 2D and 3D environments

Lazy Theta*

- Theta*
  - When expanding vertex s and generating its neighbor s’, Theta* considers
    - Making s the parent of s’ (Path 1)
    - Making the parent of s the parent of s’ (Path 2)
- Lazy Theta*
  - When expanding vertex s and generating its neighbor s’, Lazy Theta* makes the parent of s the parent of s’ (Path 2) without a line-of-sight check
  - When expanding vertex s’ and s’ does not have line-of-sight to its parent, then Lazy Theta* makes the best neighbor of s’ (= the one that minimizes the g-value of s’) the parent of s’ (Path 1).
  - [Such a neighbor exists since s is one of them.]

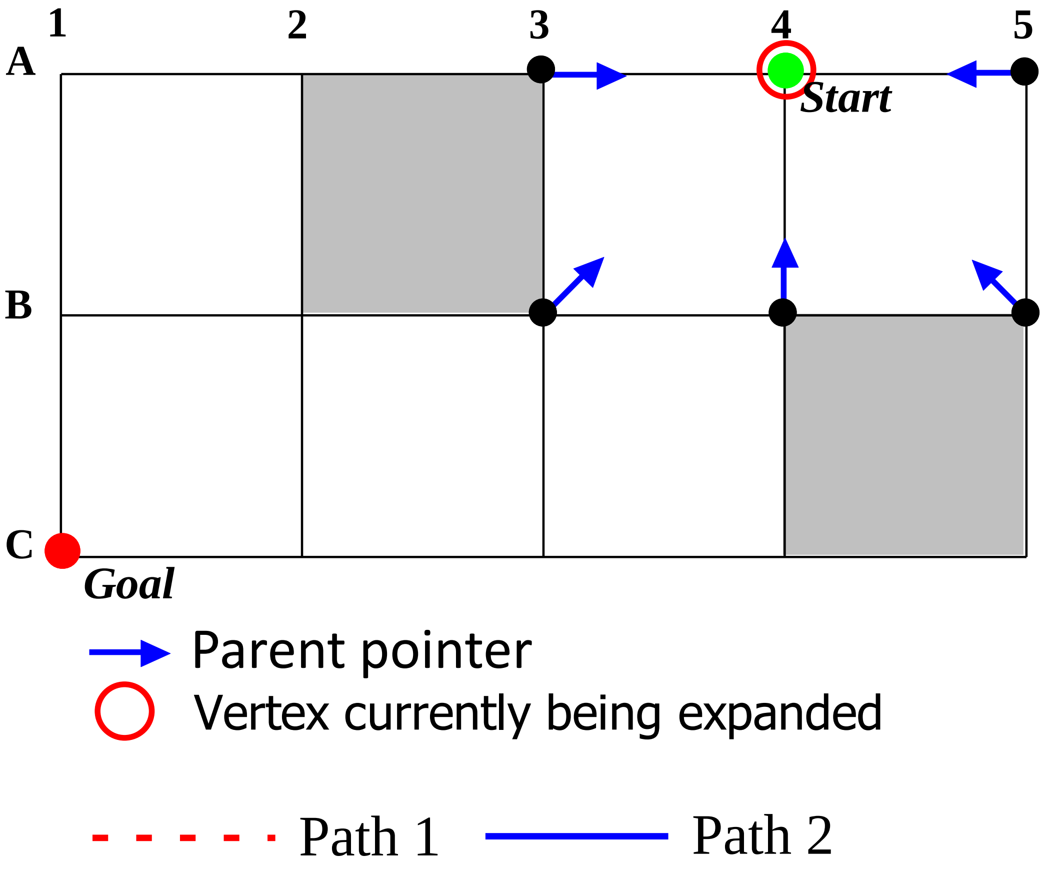
Lazy Theta*

- Lazy Theta* [Nash, Koenig and Tovey (2010)]



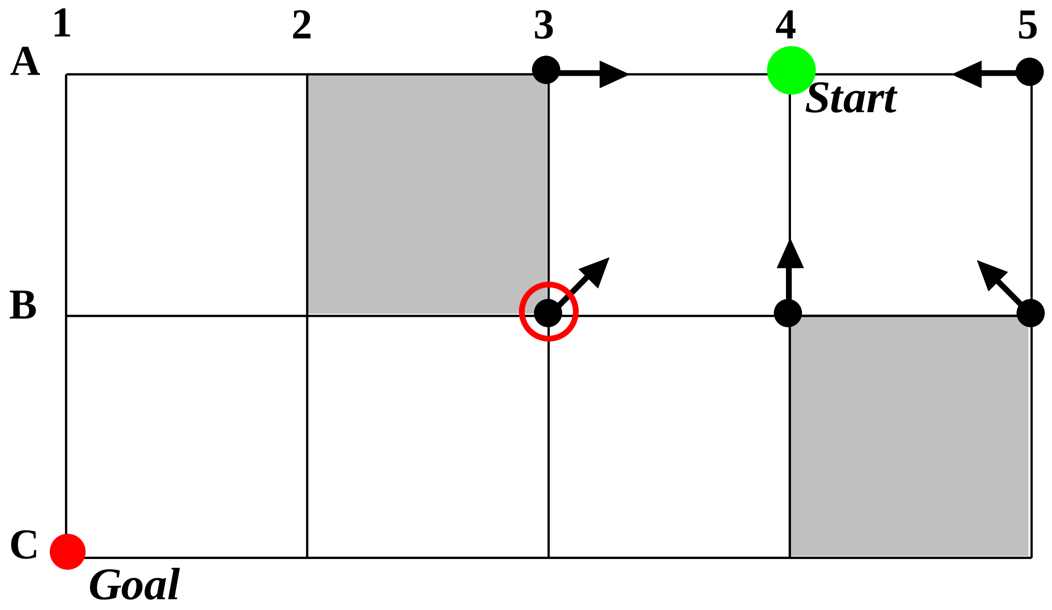
Lazy Theta*

- Lazy Theta* [Nash, Koenig and Tovey (2010)]



Lazy Theta*

- Lazy Theta* [Nash, Koenig and Tovey (2010)]



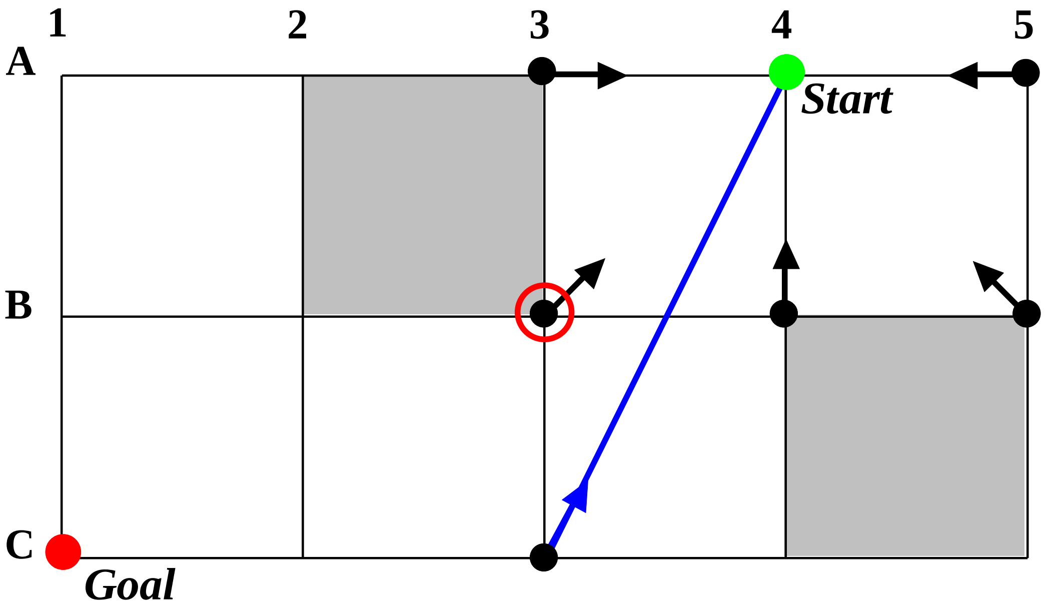
→ Parent pointer

○ Vertex currently being expanded

- - - - - Path 1 ——— Path 2

Lazy Theta*

- Lazy Theta* [Nash, Koenig and Tovey (2010)]



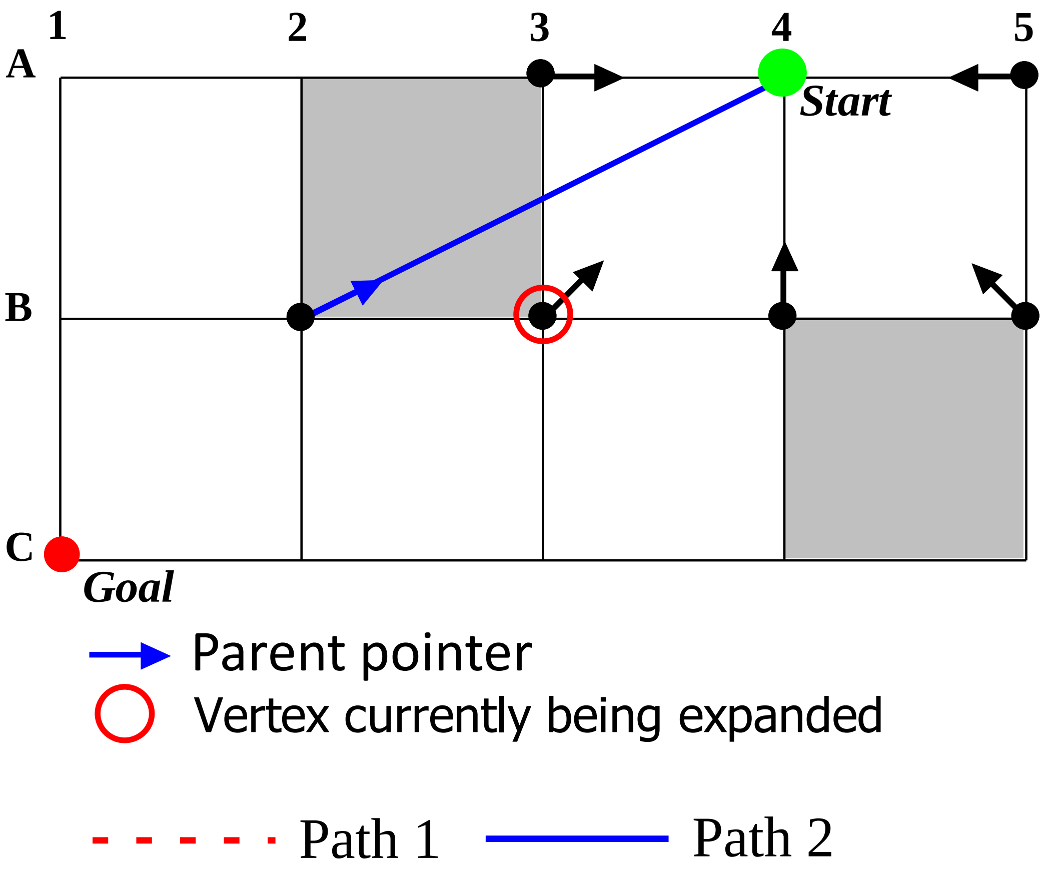
→ Parent pointer

○ Vertex currently being expanded

- - - - - Path 1 ——— Path 2

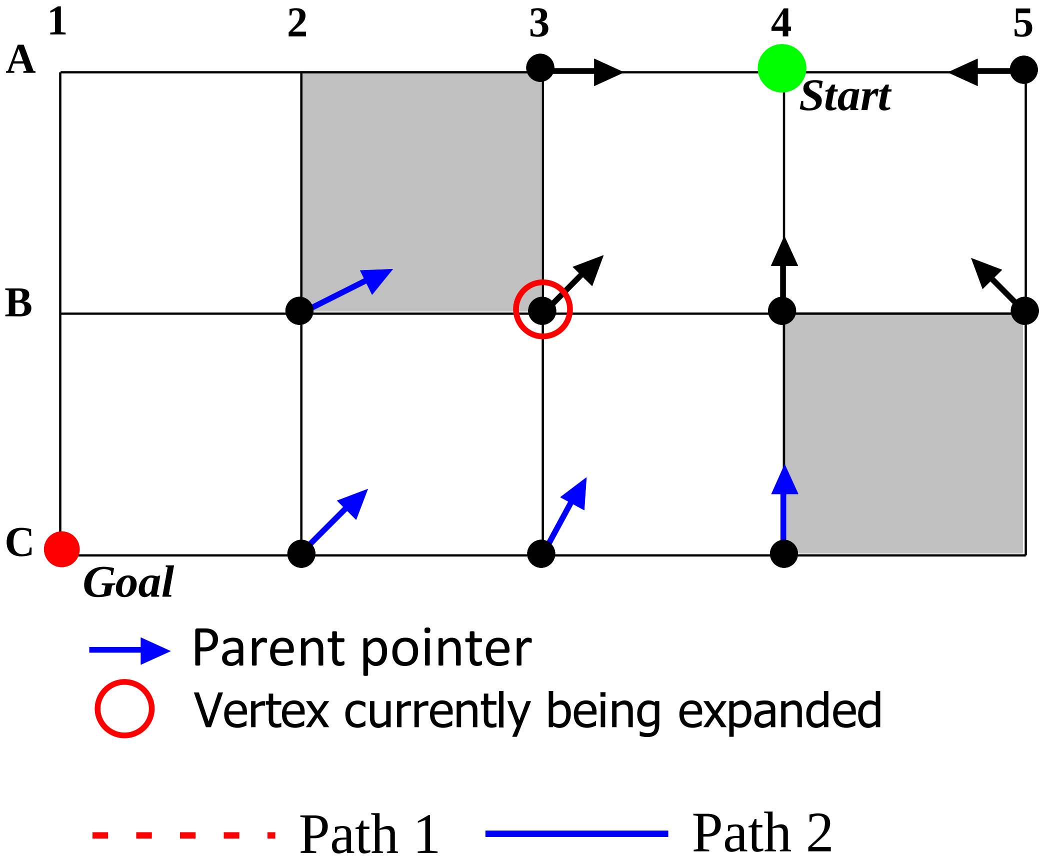
Lazy Theta*

- Lazy Theta* [Nash, Koenig and Tovey (2010)]



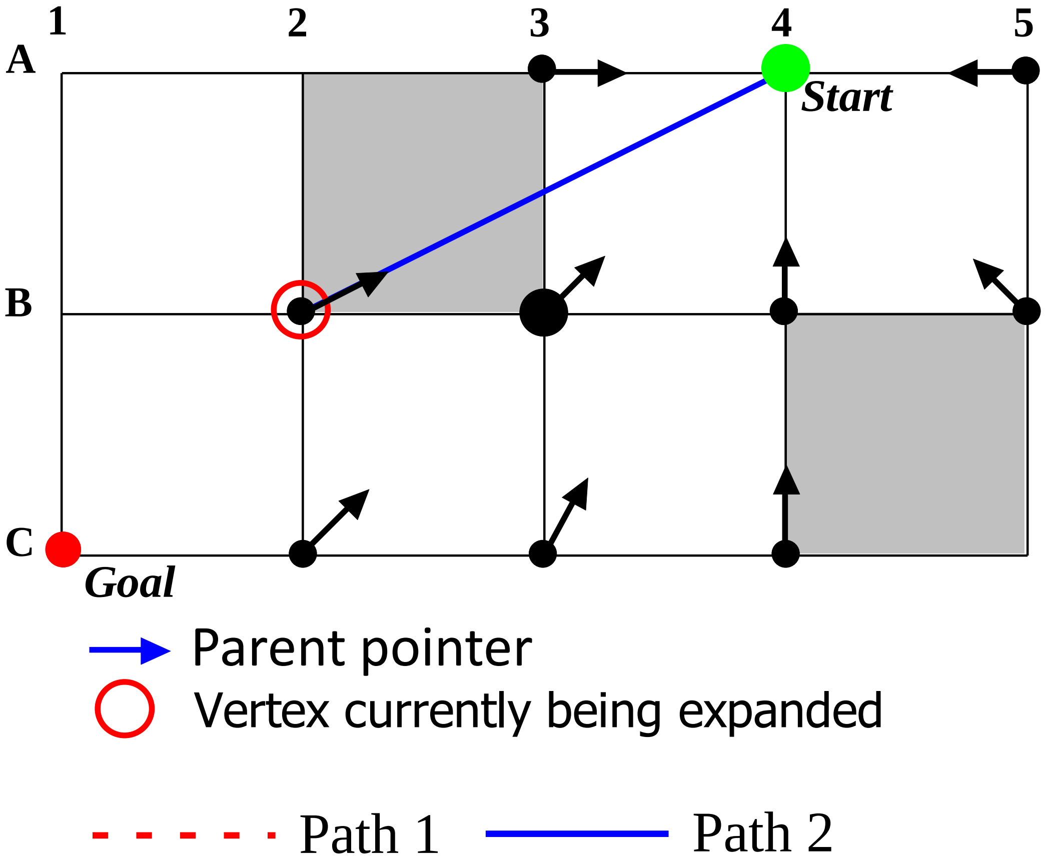
Lazy Theta*

- Lazy Theta* [Nash, Koenig and Tovey (2010)]



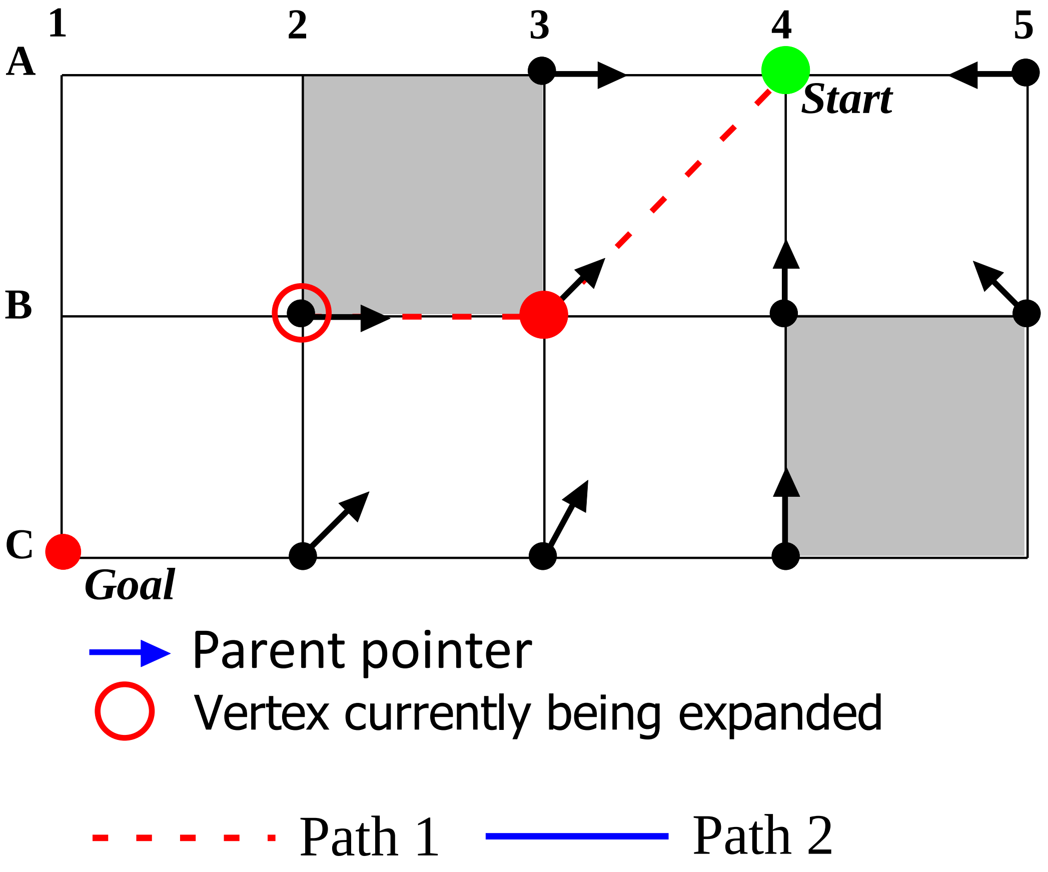
Lazy Theta*

- Lazy Theta* [Nash, Koenig and Tovey (2010)]



Lazy Theta*

- Lazy Theta* [Nash, Koenig and Tovey (2010)]



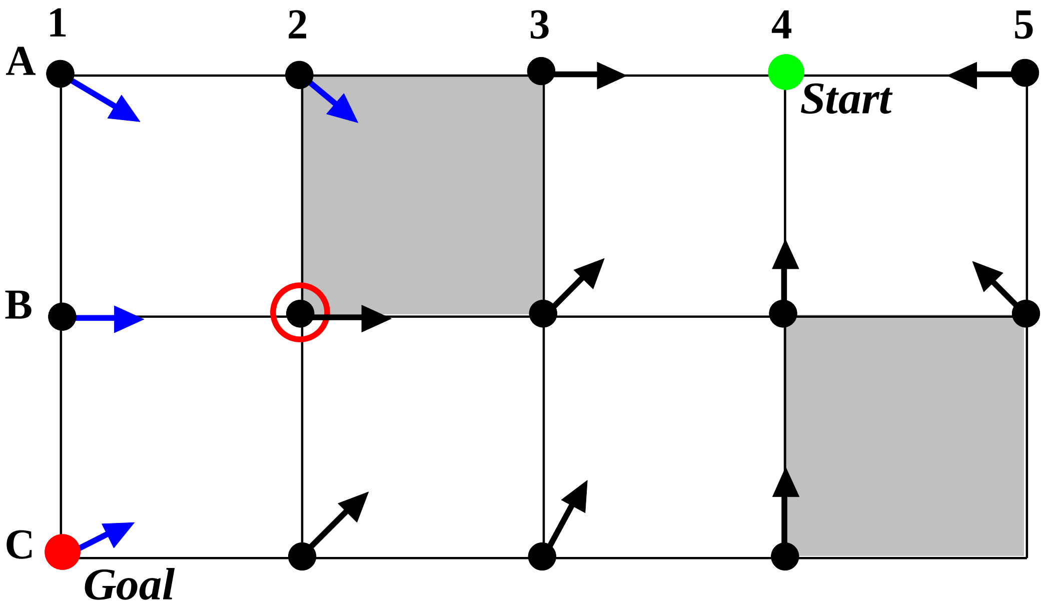
→ Parent pointer

○ Vertex currently being expanded

- - - - - Path 1 ——— Path 2

Lazy Theta*

- Lazy Theta* [Nash, Koenig and Tovey (2010)]



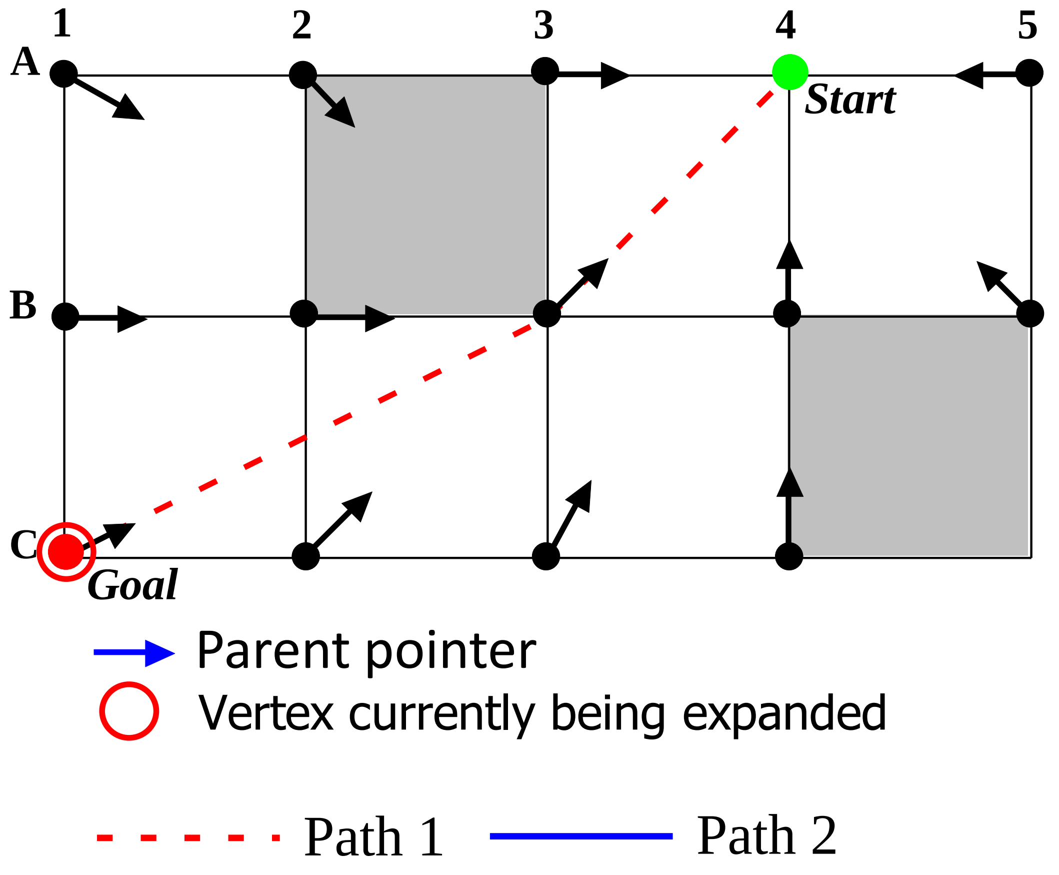
→ Parent pointer

○ Vertex currently being expanded

- - - - - Path 1 ——— Path 2

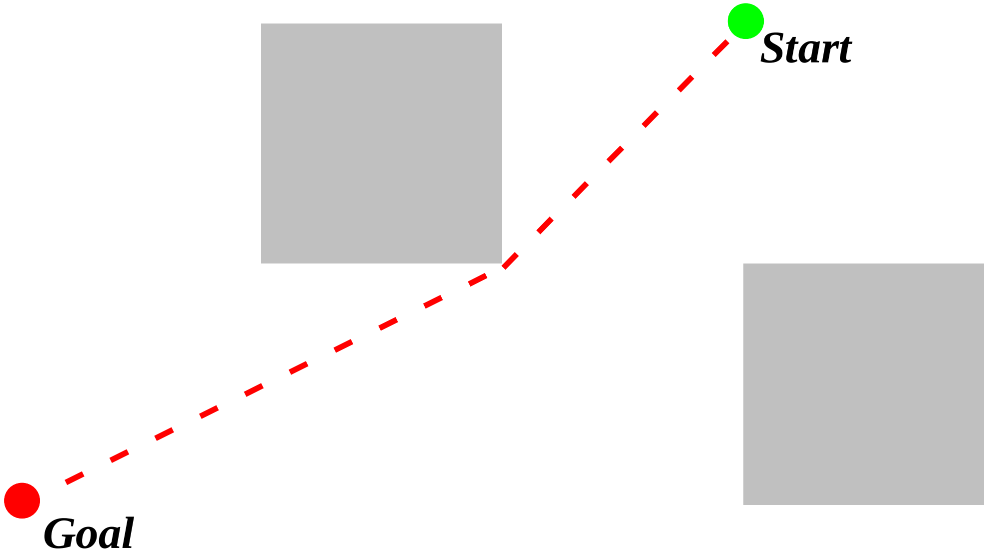
Lazy Theta*

- Lazy Theta* [Nash, Koenig and Tovey (2010)]



Lazy Theta*

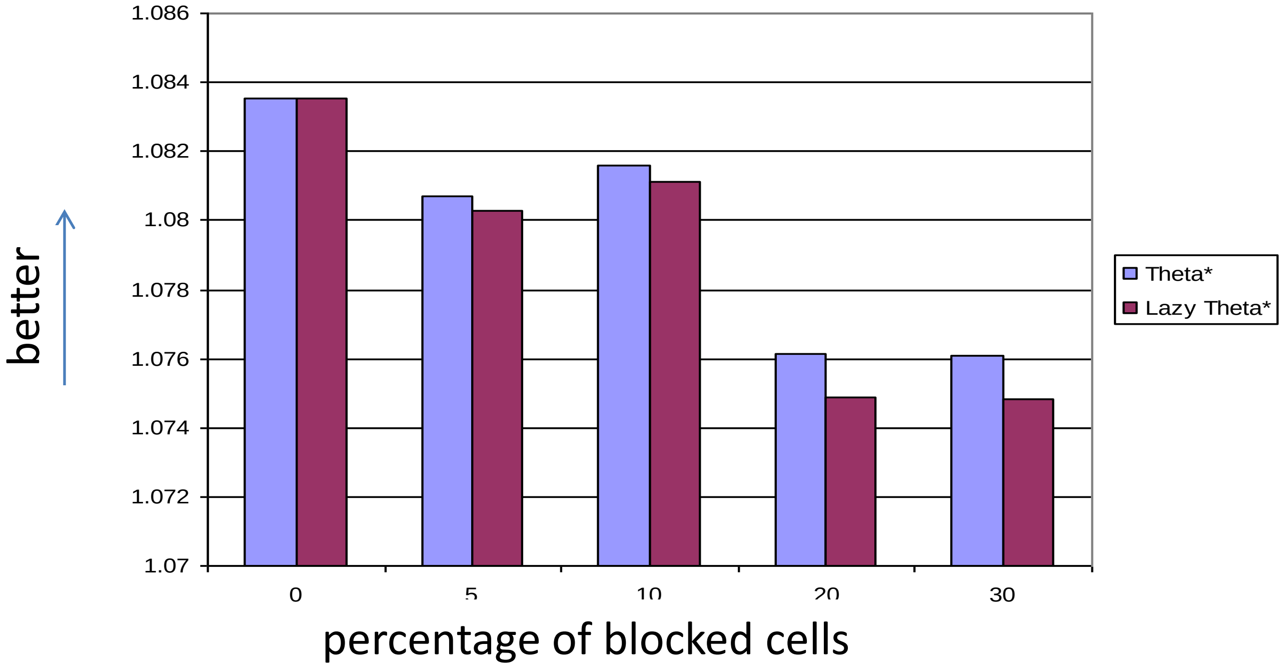
- Lazy Theta* [Nash, Koenig and Tovey (2010)]



- Theta* performed 19 line-of-sight checks
- Lazy Theta* performs 4 line-of-sight checks

Path Length

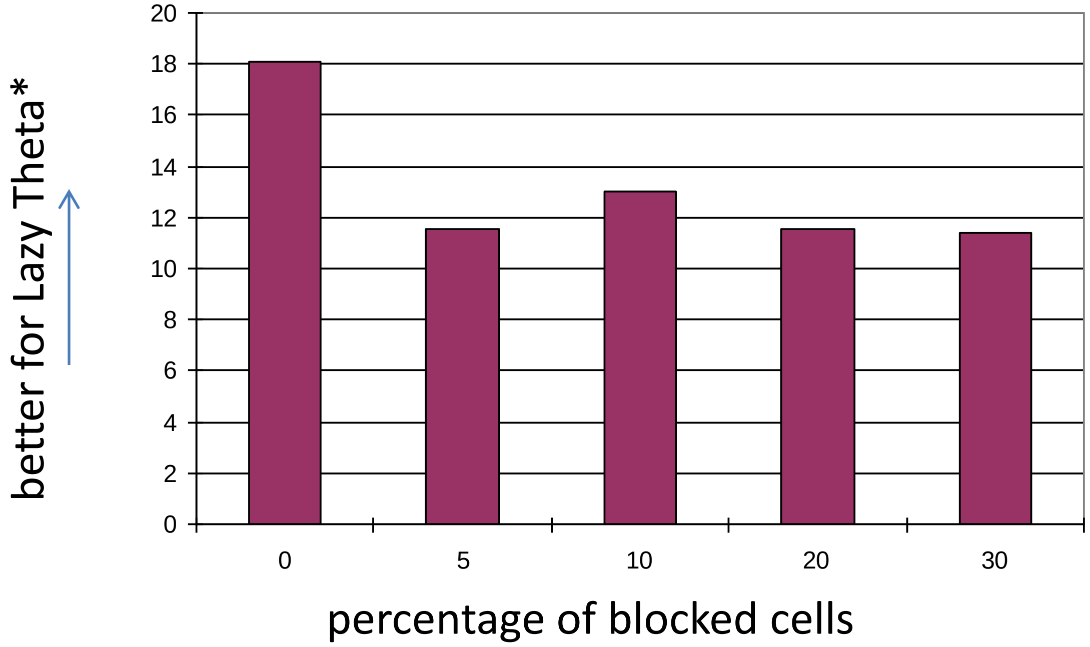
- 100 x 100 x 100 grids with randomly blocked cells



- Percentage by how much the path length of A* on grid graphs is worse than the one of Theta* (left) and Lazy Theta* (right)

Line-of-Sight Checks

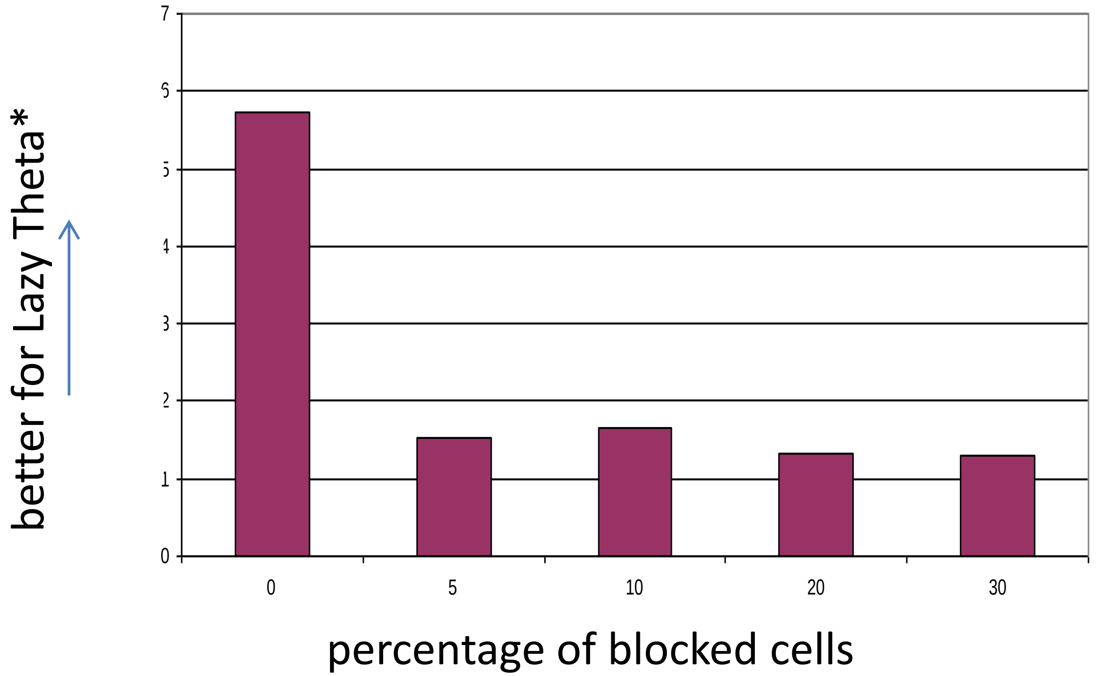
- 100 x 100 x 100 grids with randomly blocked cells



- Ratio of line-of-sight checks of Theta* and line-of-sight checks of Lazy Theta*

Computation Time

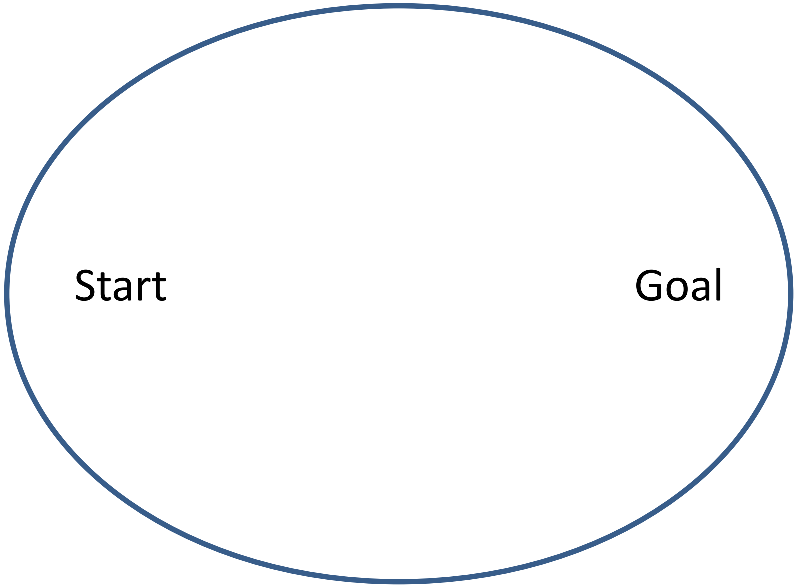
- 100 x 100 x 100 grids with randomly blocked cells



- Ratio of computation time of Theta* and computation time of Lazy Theta*

Weighted Lazy Theta*

- Lazy Theta* with A*: $f(s) = g(s) + h(s)$

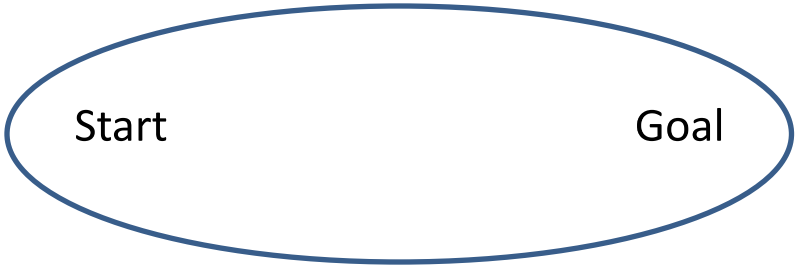


Start

Goal

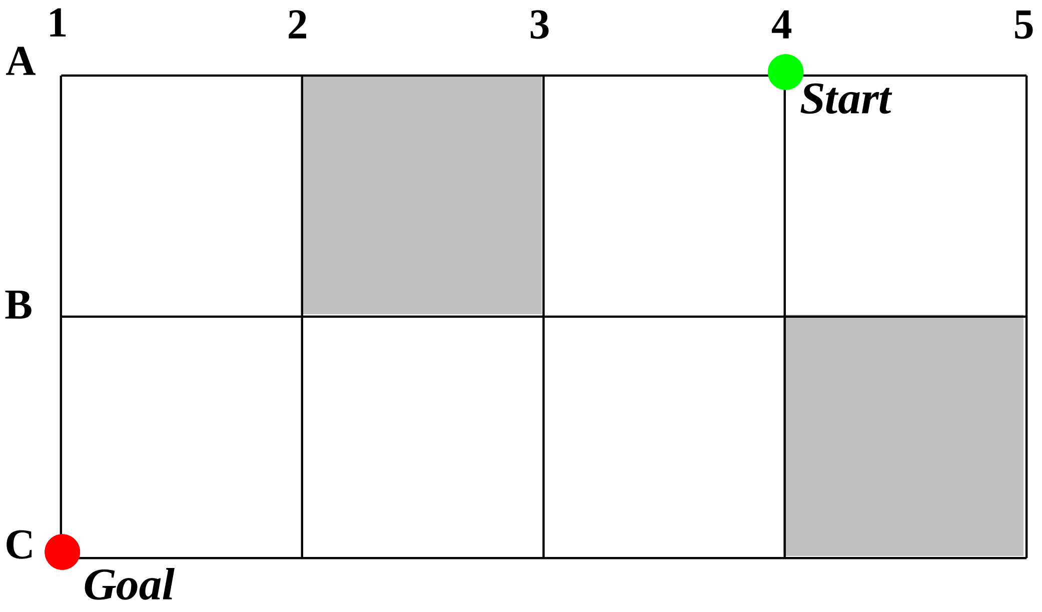
Weighted Lazy Theta*

- Lazy Theta* with A*: $f(s) = g(s) + \varepsilon \cdot h(s)$ with $\varepsilon > 1$
- Reduces vertex expansions and thus line-of-sight checks
- Both reductions reduce the computation time



Weighted Lazy Theta*

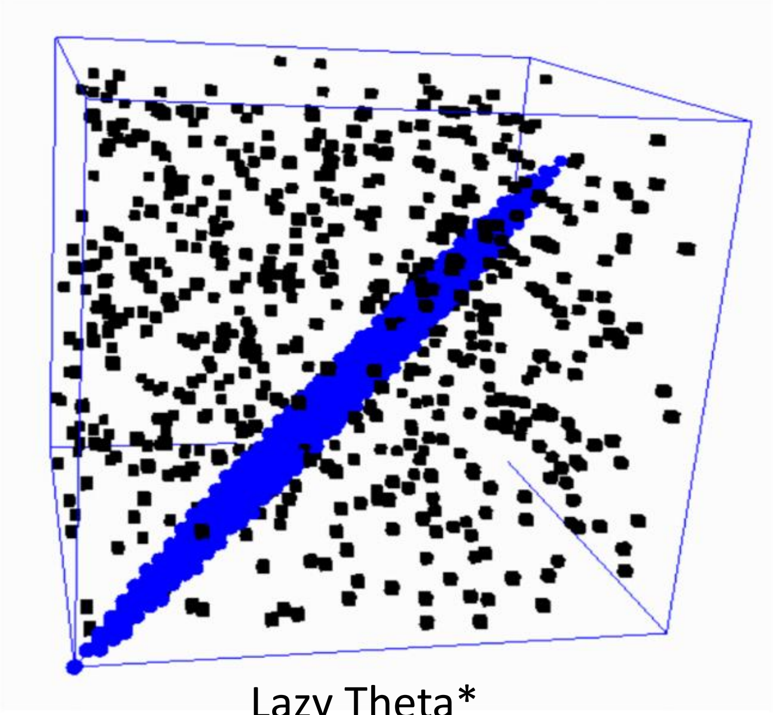
- Theta* and Lazy Theta* often do too much work



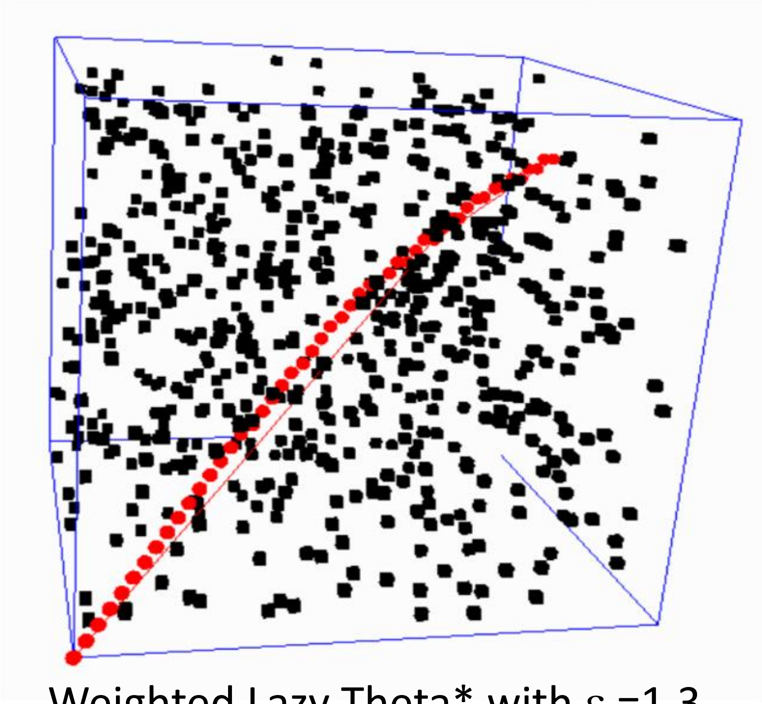
- It is sufficient if there is a single path along which the parent is propagated (when considering Path 2)

Efficiency

- Weighted Lazy Theta* expands 15 times fewer vertices but finds a path that is only 0.03% longer (which depends on how shallow the local minima are)



Lazy Theta*

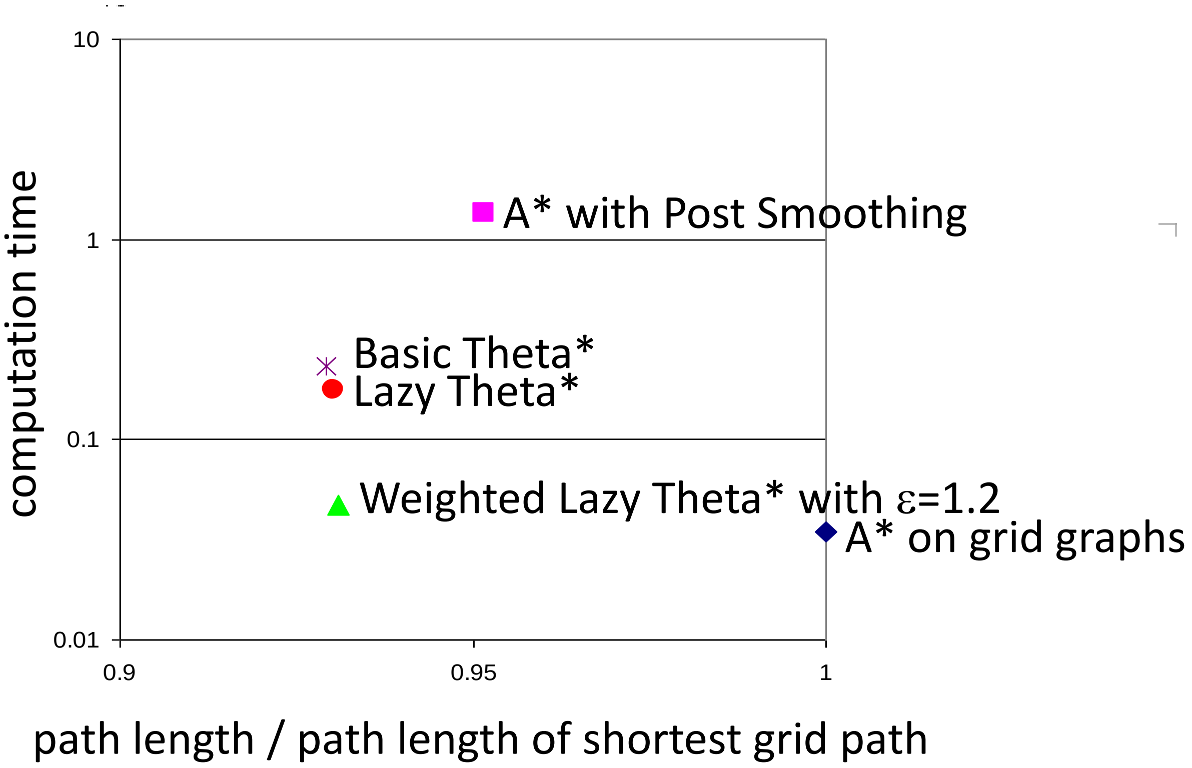


Weighted Lazy Theta* with ε =1.3

Known 3D Environments

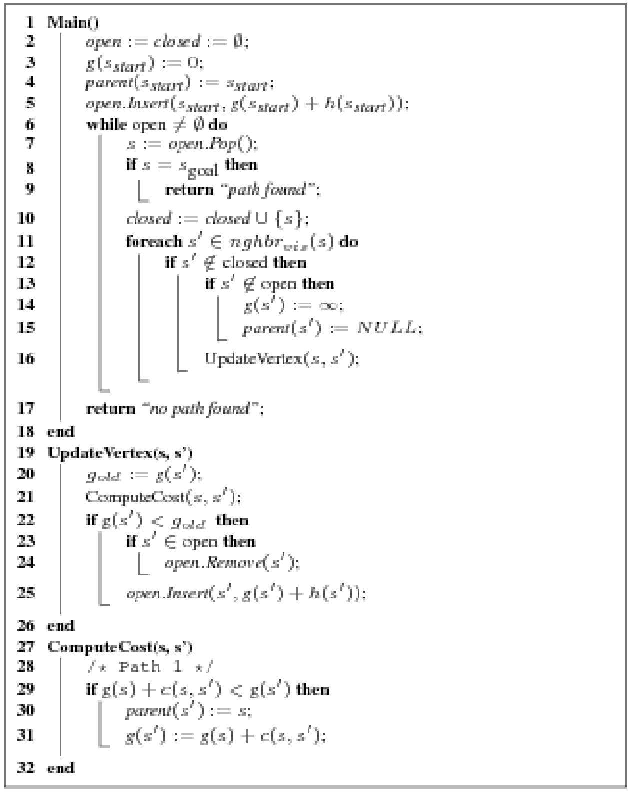
Efficiency

- 100 x 100 x 100 grids with 20% randomly blocked cells



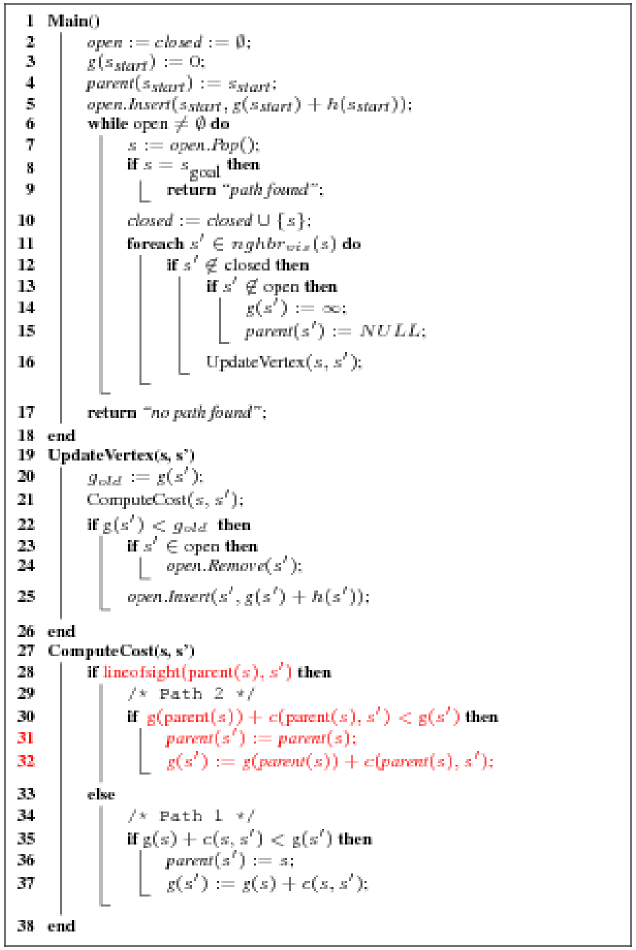
Known 3D Environments

Simplicity



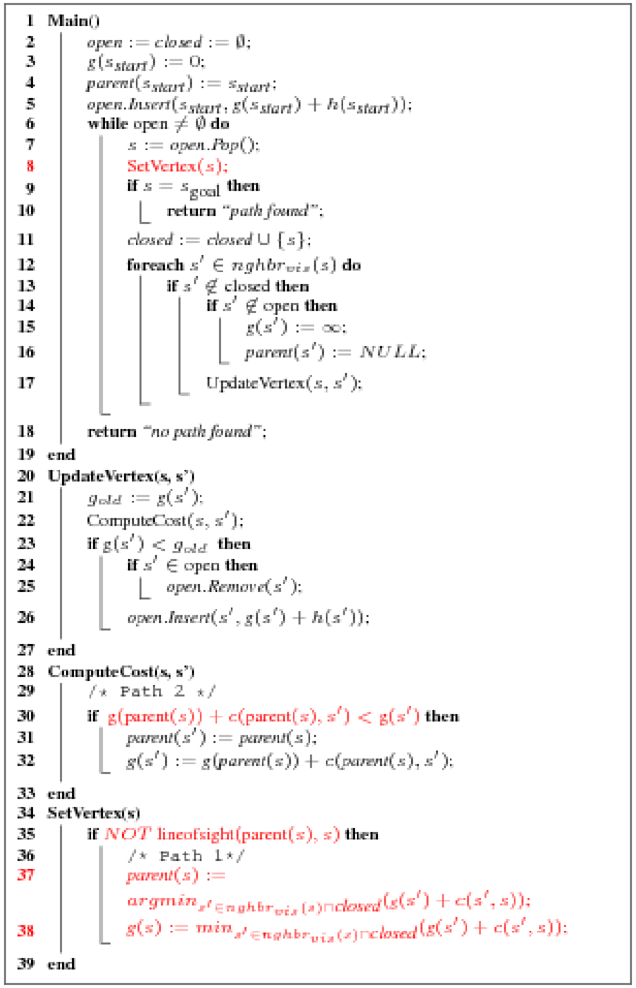
```
Main()
    $open := closed := \emptyset$;
    $g(s_{start}) := 0$;
    $parent(s_{start}) := s_{start}$;
    $open.Insert(s_{start}, g(s_{start}) + h(s_{start}))$;
    while $open \neq \emptyset$ do
        $s := open.Pop()$;
        if $s = s_{goal}$ then
            return "path found";
        $closed := closed \cup \{s\}$;
        foreach $s' \in nghbr_{vis}(s)$ do
            if $s' \notin closed$ then
                if $s' \notin open$ then
                    $g(s') := \infty$;
                    $parent(s') := NULL$;
                UpdateVertex$(s, s')$;
    return "no path found";
end
UpdateVertex(s, s')
    $g_{old} := g(s')$;
    ComputeCost$(s, s')$;
    if $g(s') < g_{old}$ then
        if $s' \in open$ then
            $open.Remove(s')$;
        $open.Insert(s', g(s') + h(s'))$;
end
ComputeCost(s, s')
    /* Path 1 */
    if $g(s) + c(s, s') < g(s')$ then
        $parent(s') := s$;
        $g(s') := g(s) + c(s, s')$;
end
```

A*



```
Main()
    $open := closed := \emptyset$;
    $g(s_{start}) := 0$;
    $parent(s_{start}) := s_{start}$;
    $open.Insert(s_{start}, g(s_{start}) + h(s_{start}))$;
    while $open \neq \emptyset$ do
        $s := open.Pop()$;
        if $s = s_{goal}$ then
            return "path found";
        $closed := closed \cup \{s\}$;
        foreach $s' \in nghbr_{vis}(s)$ do
            if $s' \notin closed$ then
                if $s' \notin open$ then
                    $g(s') := \infty$;
                    $parent(s') := NULL$;
                UpdateVertex$(s, s')$;
    return "no path found";
end
UpdateVertex(s, s')
    $g_{old} := g(s')$;
    ComputeCost$(s, s')$;
    if $g(s') < g_{old}$ then
        if $s' \in open$ then
            $open.Remove(s')$;
        $open.Insert(s', g(s') + h(s'))$;
end
ComputeCost(s, s')
    if lineofsight$(parent(s), s')$ then
        /* Path 2 */
        if $g(parent(s)) + c(parent(s), s') < g(s')$ then
            $parent(s') := parent(s)$;
            $g(s') := g(parent(s)) + c(parent(s), s')$;
    else
        /* Path 1 */
        if $g(s) + c(s, s') < g(s')$ then
            $parent(s') := s$;
            $g(s') := g(s) + c(s, s')$;
end
```

Theta*



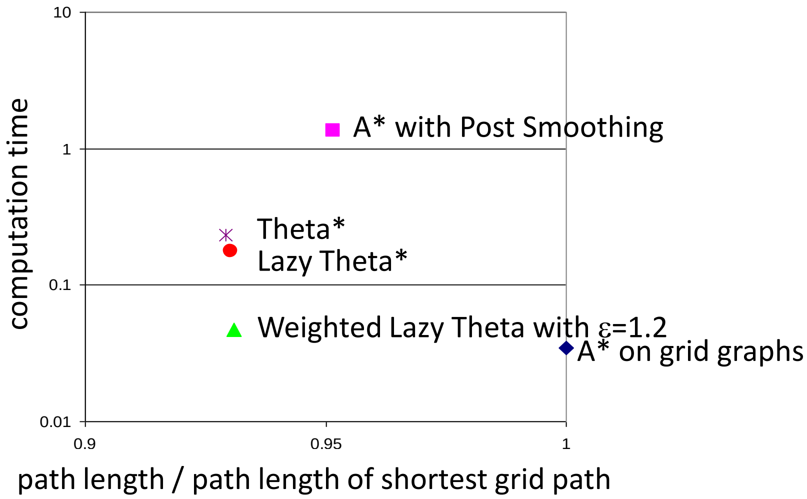
```
Main()
    $open := closed := \emptyset$;
    $g(s_{start}) := 0$;
    $parent(s_{start}) := s_{start}$;
    $open.Insert(s_{start}, g(s_{start}) + h(s_{start}))$;
    while $open \neq \emptyset$ do
        $s := open.Pop()$;
        SetVertex$(s)$;
        if $s = s_{goal}$ then
            return "path found";
        $closed := closed \cup \{s\}$;
        foreach $s' \in nghbr_{vis}(s)$ do
            if $s' \notin closed$ then
                if $s' \notin open$ then
                    $g(s') := \infty$;
                    $parent(s') := NULL$;
                UpdateVertex$(s, s')$;
    return "no path found";
end
UpdateVertex(s, s')
    $g_{old} := g(s')$;
    ComputeCost$(s, s')$;
    if $g(s') < g_{old}$ then
        if $s' \in open$ then
            $open.Remove(s')$;
        $open.Insert(s', g(s') + h(s'))$;
end
ComputeCost(s, s')
    /* Path 2 */
    if $g(parent(s)) + c(parent(s), s') < g(s')$ then
        $parent(s') := parent(s)$;
        $g(s') := g(parent(s)) + c(parent(s), s')$;
end
SetVertex(s)
    if $NOT$ lineofsight$(parent(s), s)$ then
        /* Path 1*/
        $parent(s) :=$
        $argmin_{s' \in nghbr_{vis}(s) \cap closed}(g(s') + c(s', s))$;
        $g(s) := min_{s' \in nghbr_{vis}(s) \cap closed}(g(s') + c(s', s))$;
end
```

Lazy Theta*

Known 3D Environments

Any-Angle Search

Property 1: Efficiency



Any-Angle Search Method	Property 2: Simplicity	Property 3: Generality (Any Euclidean Graph)
A* with Post Smoothing	Yes	Yes
3D Field D*	No	No
Theta*	Yes	Yes
(Weighted) Lazy Theta*	Yes	Yes

Table of Contents

○ Introduction

○ Analysis of Path Lengths

○ Any-Angle Search Methods

  ○ Known 2D Environments

  ○ Known 3D Environments

  ○ Unknown 2D Environments (1 slide only)

○ Conclusion

Replanning

- Field D* and Theta* can both use incremental search to replan faster than from scratch (Field D* was designed with this in mind) but neither of them works for every graph embedded in 2D Euclidean space

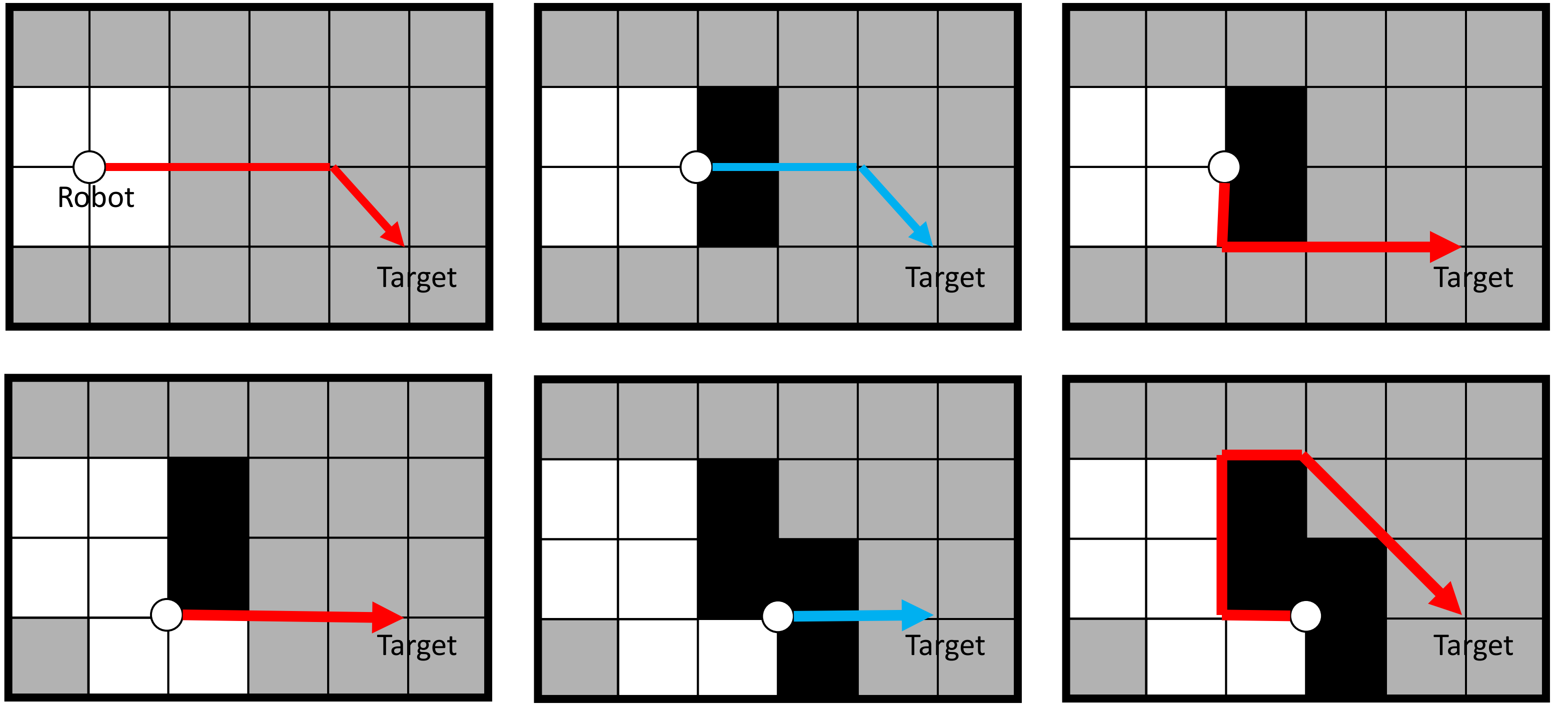
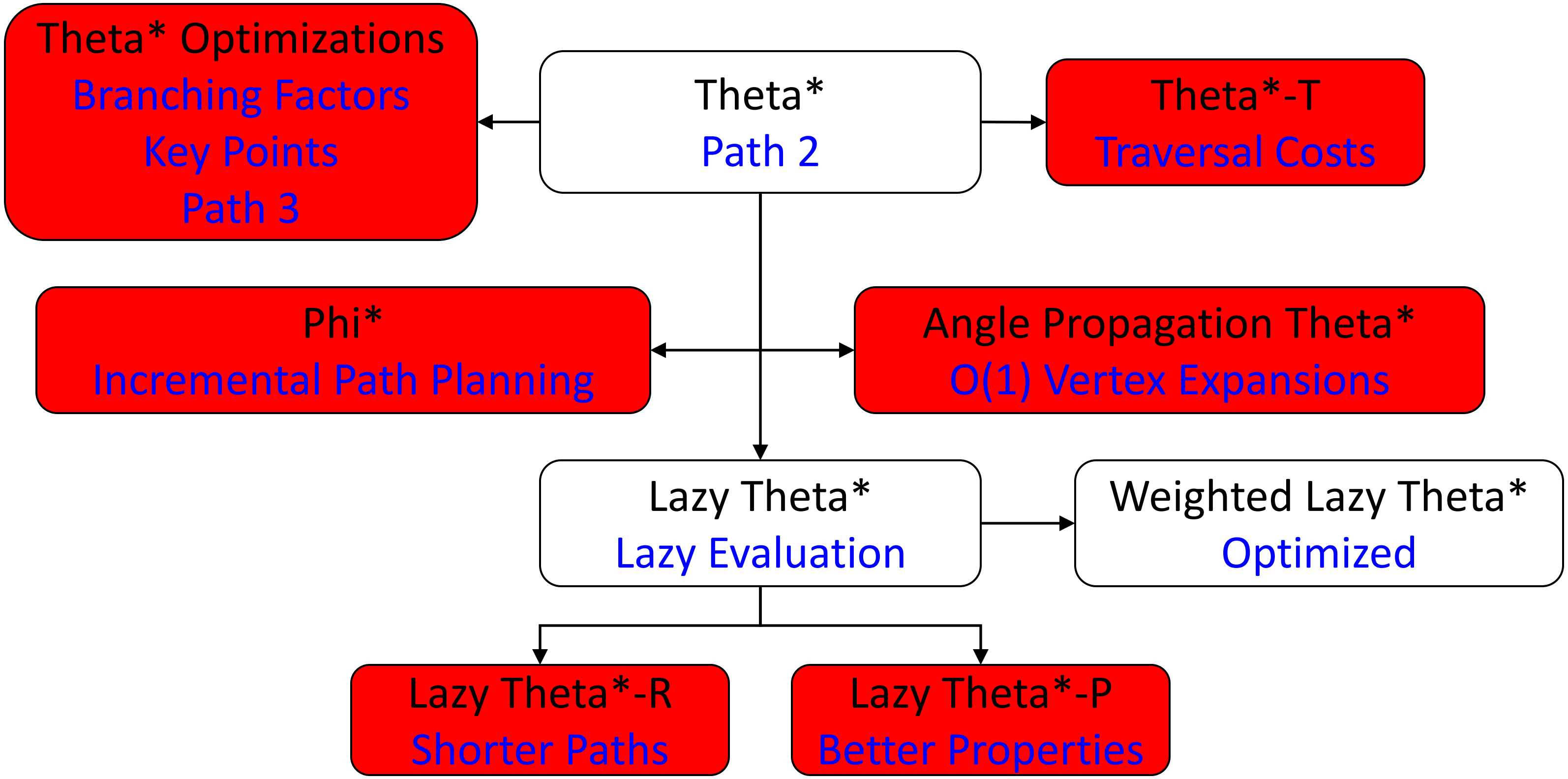


Table of Contents

- Introduction
- Analysis of Path Lengths
- Any-Angle Search Methods
  - Known 2D Environments
  - Known 3D Environments
  - Unknown 2D Environments (1 slide only)
- Conclusion

Theta*



Theta* Optimizations
Branching Factors
Key Points
Path 3

Theta*
Path 2

Theta*-T
Traversal Costs

Phi*
Incremental Path Planning

Angle Propagation Theta*
O(1) Vertex Expansions

Lazy Theta*
Lazy Evaluation

Weighted Lazy Theta*
Optimized

Lazy Theta*-R
Shorter Paths

Lazy Theta*-P
Better Properties

🟥 not covered in this talk

Other Any-Angle Search Methods

- There are any-angle search methods for several trade offs between computation time and path length

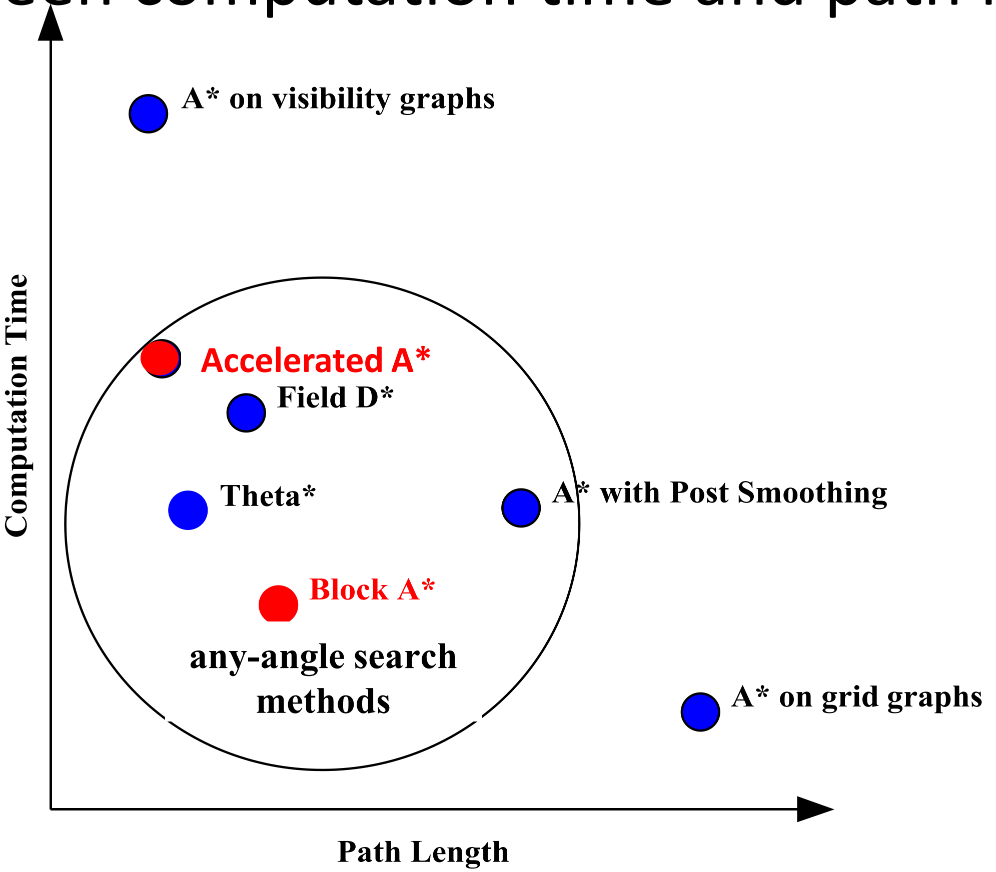


figure is notional
extensive experimental comparisons have not been performed yet

Block A*

- Block A*
  [Yap et al. (2011)]
  - partitions a square grid into blocks of equal size
  - uses an A* search that expands blocks rather than cells
  - pre-computes (edge-constrained or any-angle) paths within each block to speed up the A* search

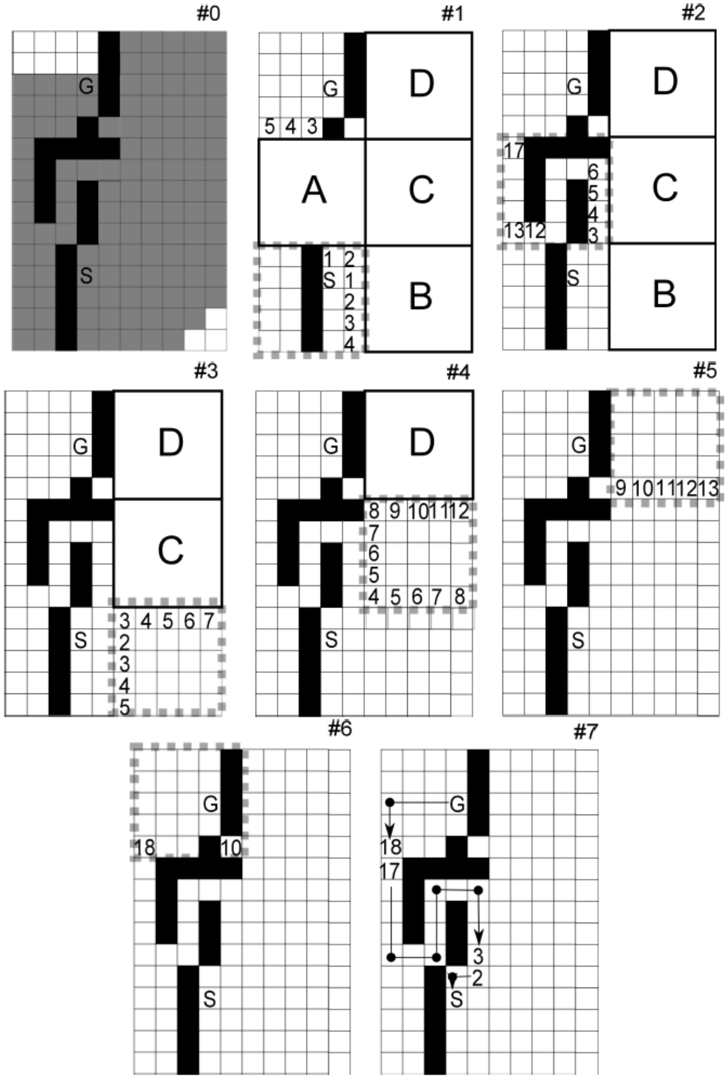
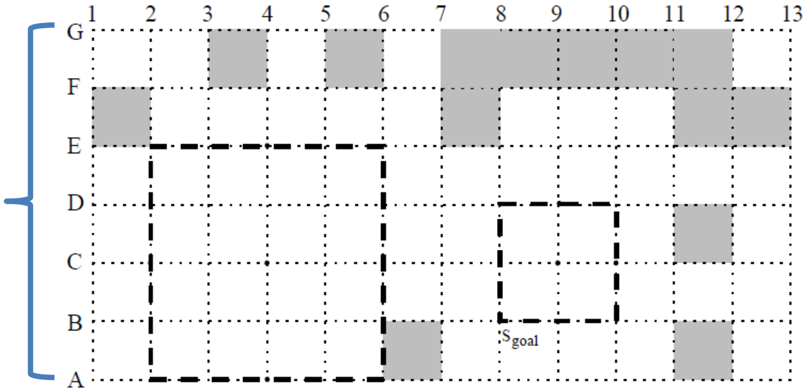


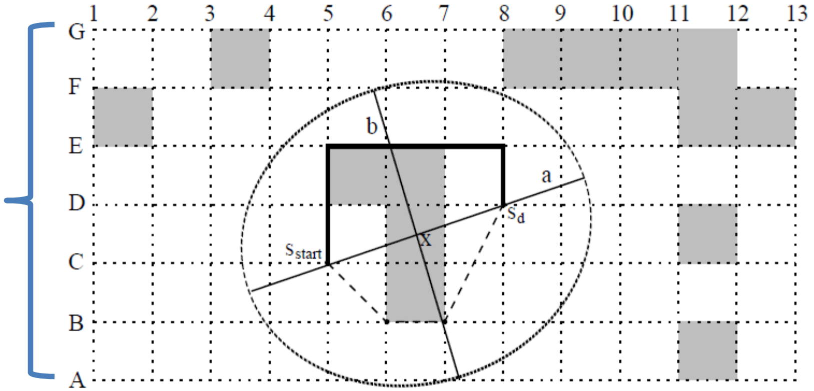
figure from the cited paper

Accelerated A*

- Accelerated A* [Sislak et al. (2009)]
  - Uses an adaptive step size to determine the neighbors of a vertex
  - Considers more parents than Theta*, namely all expanded vertices (using a sufficiently large ellipse to prune unpromising expanded vertices)



adaptive step size



ellipse

figures from the cited paper

Additional Information

- Theta* Dissertation
  - A. Nash. Any-Angle Path Planning. Dissertation. Computer Science Department, University of Southern California, 2012.
    (can be retrieved from idm-lab.org soon)

Additional Information

- Theta* Publications
  - K. Daniel, A. Nash, S. Koenig and A. Felner. Theta*: Any-Angle Path Planning on Grids. *Journal of Artificial Intelligence Research,* 39, 533-579, 2010.
  - A. Nash, S. Koenig and C. Tovey. Lazy Theta*: Any-Angle Path Planning and Path Length Analysis in 3D. *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI),* 2010.
  - A. Nash, S. Koenig and M. Likhachev. Incremental Phi*: Incremental Any-Angle Path Planning on Grids. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI),* 1824-1830, 2009.
  - S. Koenig, K. Daniel and A. Nash. A Project on Any-Angle Path Planning for Computer Games for 'Introduction to Artificial Intelligence' Classes. *Technical Report, Department of Computer Science. University of Southern California, Los Angeles (California)*, 2008.
  - A. Nash, K. Daniel, S. Koenig and A. Felner. Theta*: Any-Angle Path Planning on Grids. *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI),* 1177-1183, 2007.

Additional Information

- Field D* and 3D Field D* Publications
  - J. Carsten, A. Rankin, D. Ferguson and A. Stentz: Global Planning on the Mars Exploration Rovers: Software Integration and Surface Testing. *Journal of Field Robotics*, 26, 337-357, 2009.
  - J. Carsten, D. Ferguson and A. Stentz. 3D Field D*: Improved Path Planning and Replanning in Three Dimensions. *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 3381-3386, 2006.
  - D. Ferguson and A. Stentz. Using Interpolation to Improve Path Planning: The Field D* Algorithm. *Journal of Field Robotics*, 23(2), 79-101, 2006.
  - A more general closed form linear interpolation equation that can be used on triangular meshes was introduced in L. Sapronov and A. Lacaze: Path Planning for Robotic Vehicles using Generalized Field D*. *Proceedings of the SPIE*, 6962, 2010.

Additional Information

- Block A* Publications
  - P. Yap, N. Burch, R. Holte and J. Schaeffer: Block A*: Database-Driven Search with Applications in Any-Angle Path Planning. *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2011.
  - P. Yap, N. Burch, R. Holte and J. Schaeffer: Any-Angle Path Planning for Computer Games. *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2011.
- Accelerated A* Publications
  - D. Sislak, P. Volf and M. Pechoucek: Accelerated A* Trajectory Planning: Grid-Based Path Planning Comparison. *Proceedings of the ICAPS 2009 Workshop on Planning and Plan Execution for Real-World Systems*, 2009.
  - D. Sislak, P. Volf and M. Pechoucek: Accelerated A* Path Planning. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2009.

Additional Information

- Web Pages
  - http://idm-lab.org/project-o.html
    (lots of information, including a class project)
  - http://aigamedev.com/open/tutorials/theta-star-any-angle-paths/
    (tutorial)
- People
  - Alex Nash (anash@usc.edu / alexwnash@gmail.com)
  - Sven Koenig (skoenig@usc.edu)

Acknowledgements

- Ideas and Technical Comments
  - Kenny Daniel
  - Ariel Felner
  - Maxim Likhachev
  - Xiaoxun Sun
  - Craig Tovey
  - Peter Yap
  - William Yeoh
  - Xiaoming Zheng

Acknowledgements

- Research Support
  - NSF (in part while serving at NSF)
  - ARO under grant number W911NF-08-1-0468
  - ONR under grant number N00014-09-1-1031
  - Northrop Grumman via a fellowship to Alex Nash
  - Nathan Sturtevant by providing game maps at movingai.com

- The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, companies or the U.S. government.