USC Viterbi
School of Engineering

Single-Robot and Multi-Robot Path Planning with Quality Guarantees

Cognitive Robotics Summer School 2017

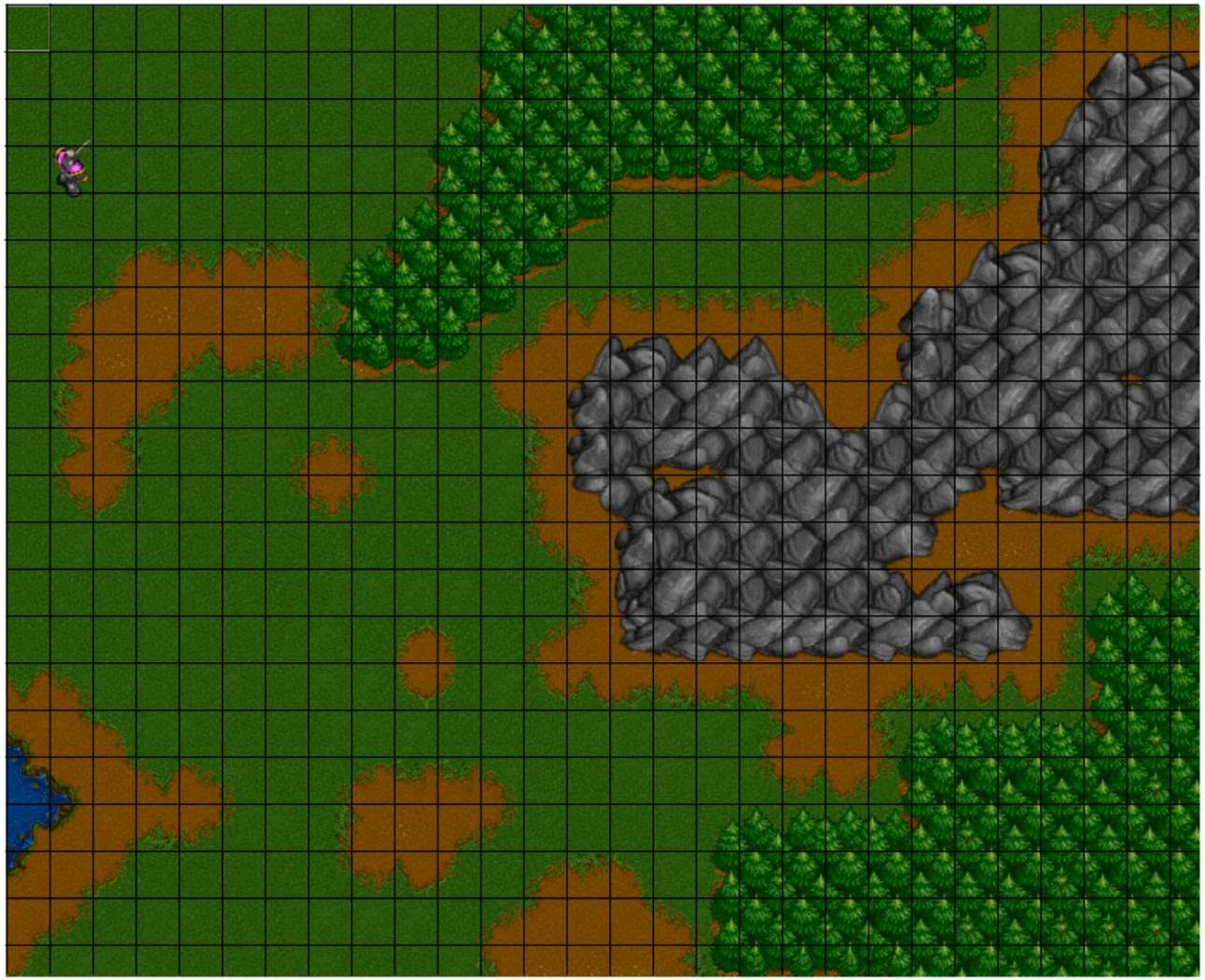
Sven Koenig,
the awesome people in his research group
and their awesome collaborators
University of Southern California

idm-lab.org
skoenig@usc.edu

Background: Path Planning

Warcraft II

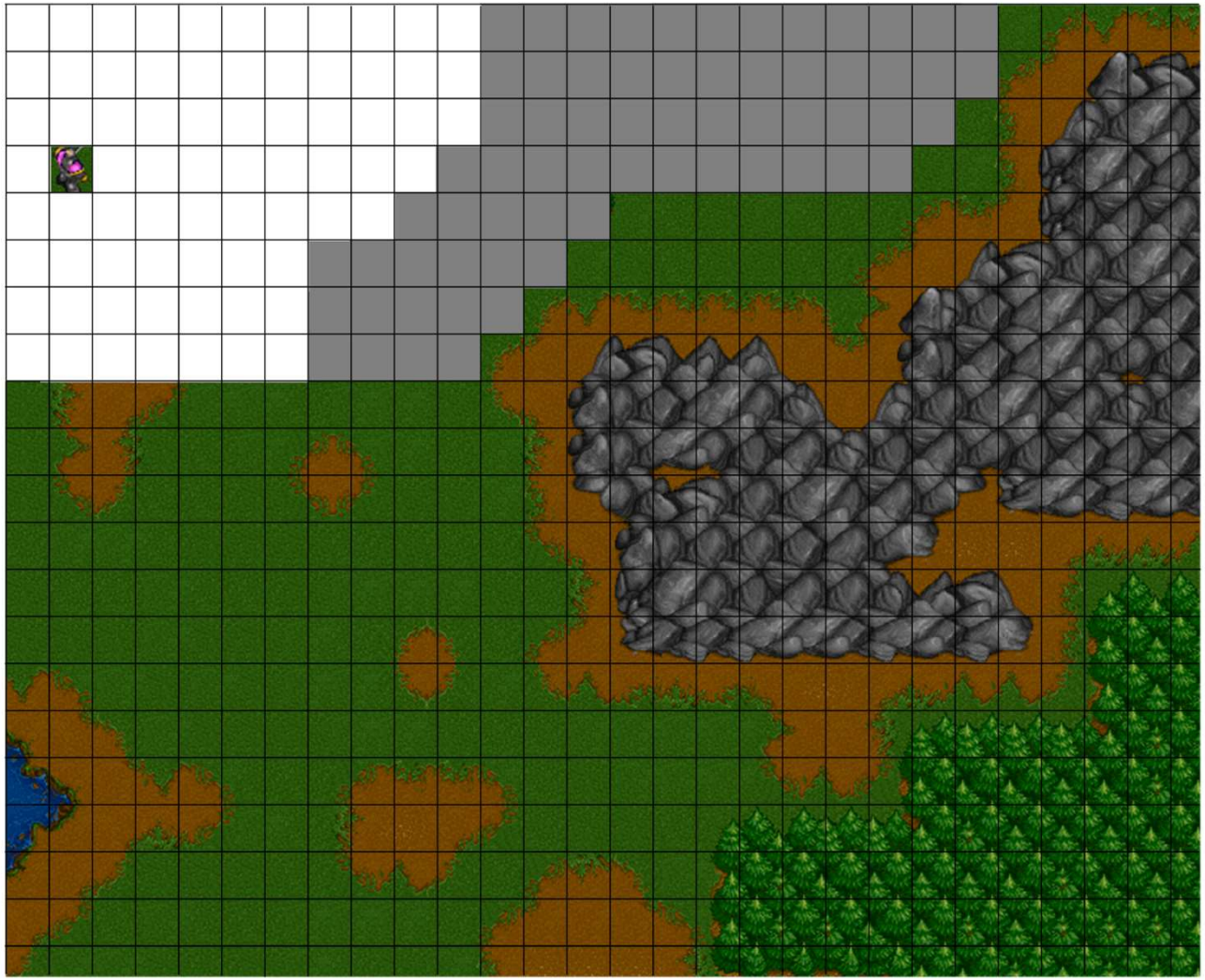
Background: Path Planning



Warcraft II

8-neighbor grid

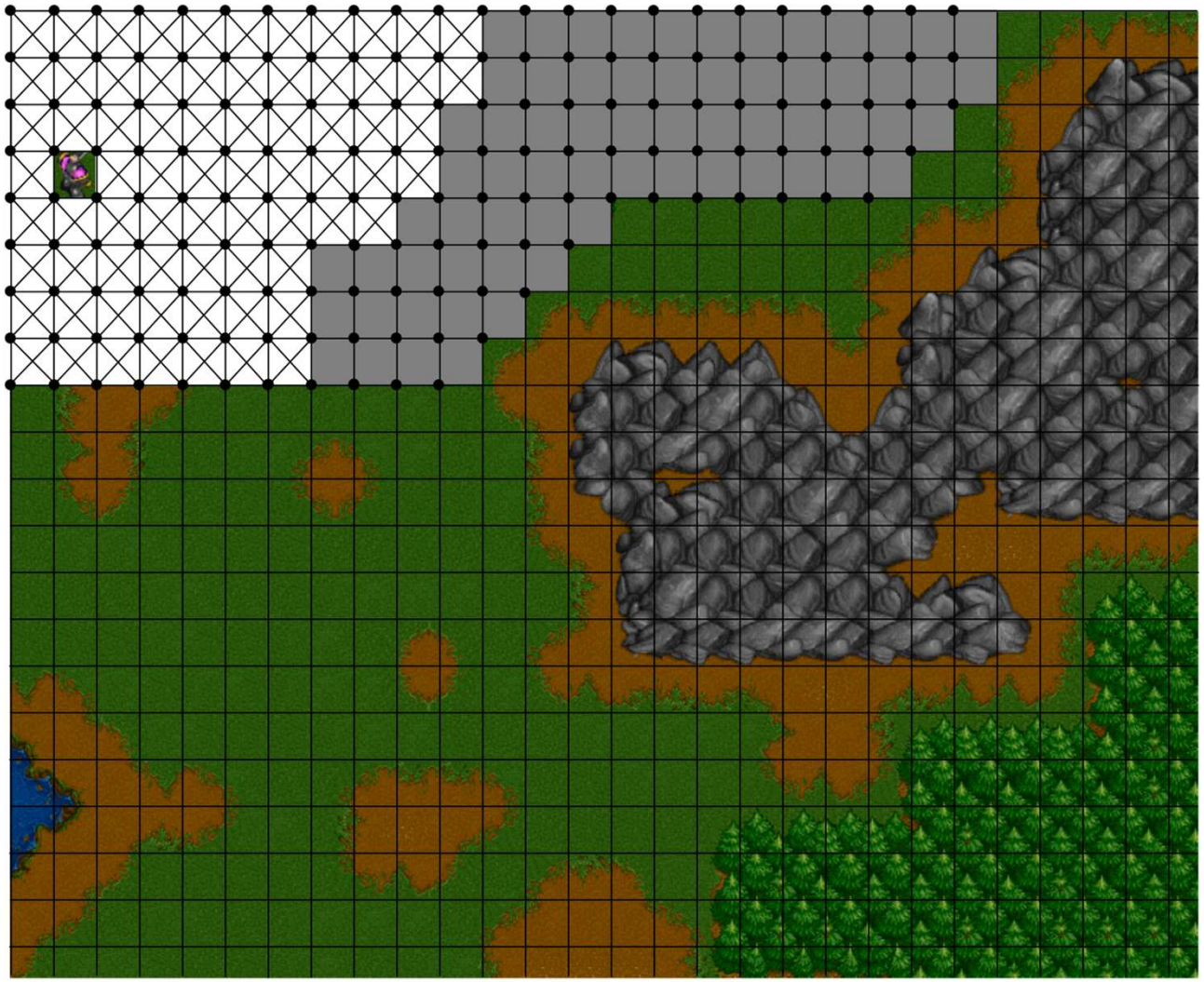
Background: Path Planning



Warcraft II

8-neighbor grid

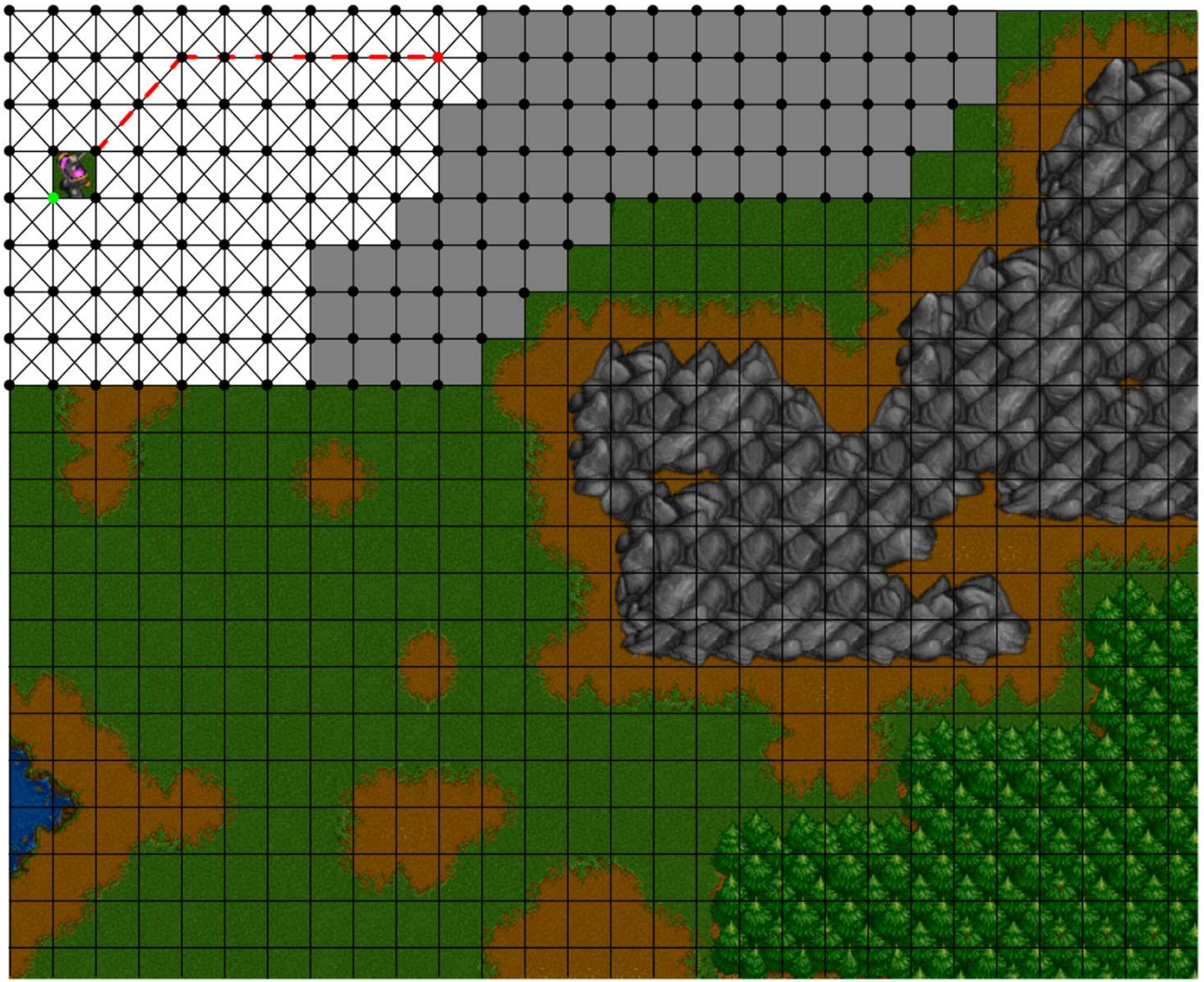
Background: Path Planning



Warcraft II

8-neighbor grid

Background: Path Planning



Warcraft II

8-neighbor grid

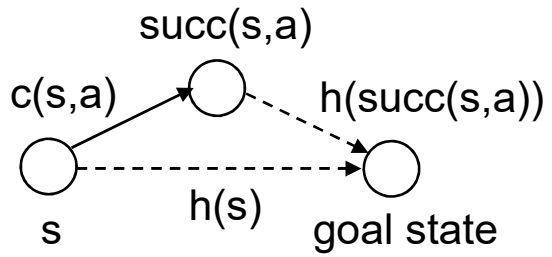
Background: Path Planning

Warcraft II

8-neighbor grid

Background: A*

- A* uses user-supplied h-values to focus its search
- The h-values approximate the goal distances
- We always assume that the h-values are consistent!
- The h-values h(s) are consistent
  if they satisfy the triangle inequality:
  h(s) = 0 if s is the goal state
  h(s) ≤ c(s,a) + h(succ(s,a)) otherwise
- Consistent h-values are admissible.
- The h-values h(s) are admissible
  if they do not overestimate the goal distances.



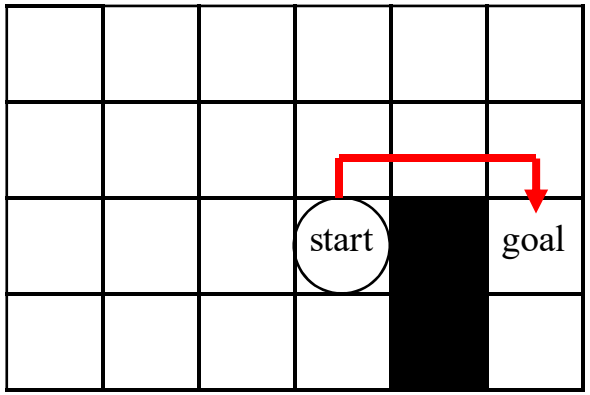
Background: A*

- Finds optimal (shortest) paths

1. Create a search tree that contains only the start state
2. Pick a generated but not yet expanded state s with the smallest f-value, where f(s) = g(s) + h(s)
3. If state s is a goal state: stop
4. Expand state s
5. Go to 2

Background: A*

- Search problem with uniform cost



4-neighbor grid

Background: A*

- Possible consistent h-values

7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0
6	5	4	3	2	1

Manhattan Distance

5	4	3	2	2	2
5	4	3	2	1	1
5	4	3	2	1	0
5	4	3	2	1	1

Octile Distance

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

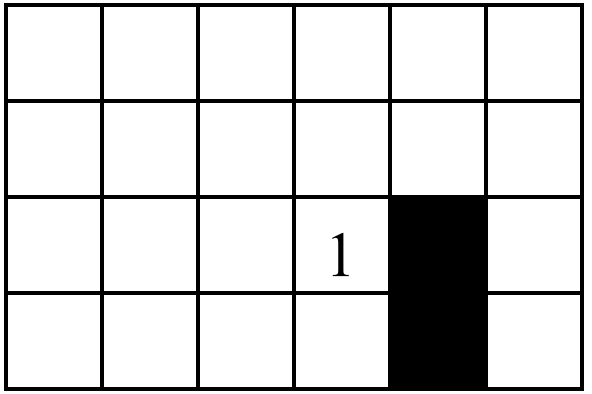
Zero h-values

more informed (dominating)

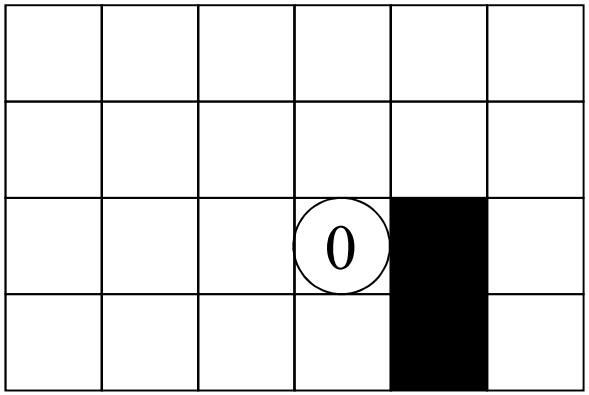
4-neighbor grid

Background: A*

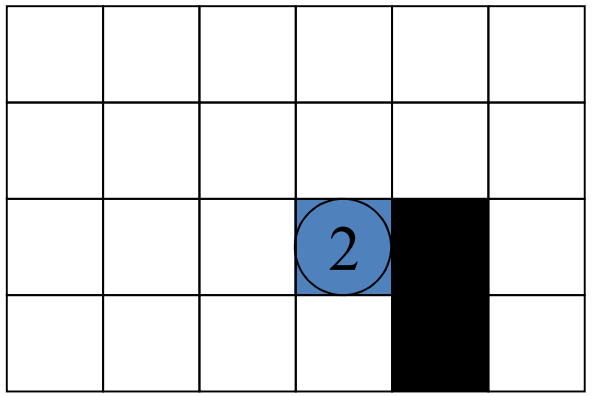
order of expansions



- First iteration of A*



7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0
6	5	4	3	2	1



g-values + h-values = f-values

cost of the shortest path in the search tree from the start state to the given state

generated but not expanded state (OPEN list)

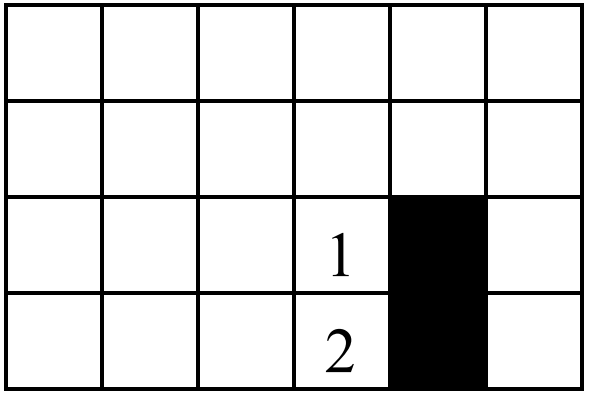
expanded state (CLOSED list)

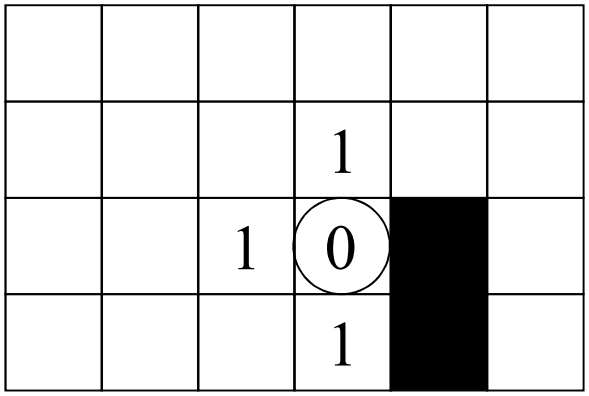
4-neighbor grid

Background: A*

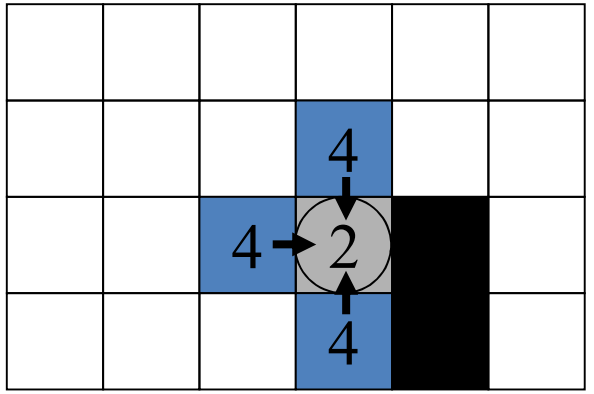
- Second iteration of A*

order of expansions





7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0
6	5	4	3	2	1



g-values + h-values = f-values

cost of the shortest path in the search tree from the start state to the given state

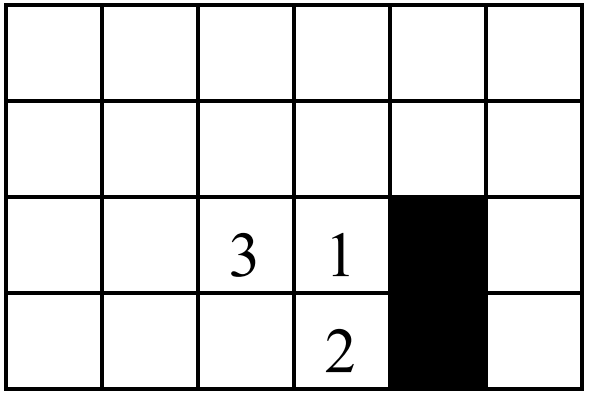
generated but not expanded state (OPEN list)

expanded state (CLOSED list)

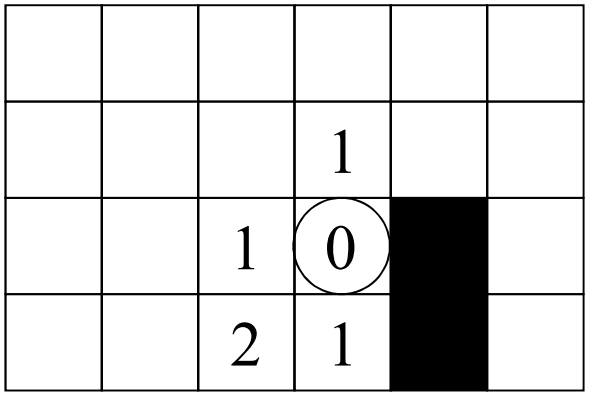
4-neighbor grid

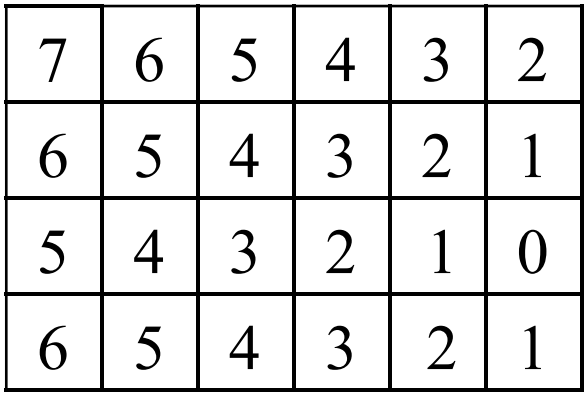
Background: A*

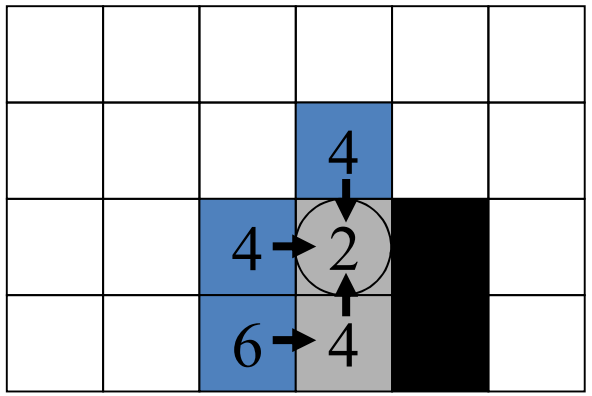
order of expansions



- Third iteration of A*







g-values + h-values = f-values

cost of the shortest path in the search tree from the start state to the given state

generated but not expanded state (OPEN list)

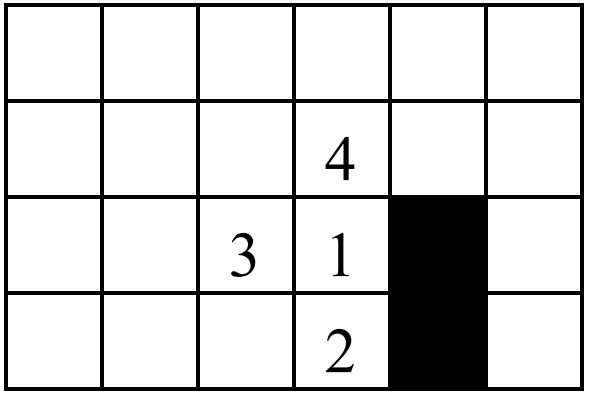
expanded state (CLOSED list)

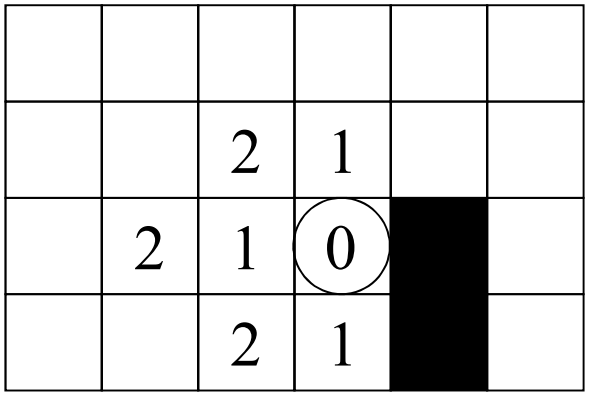
4-neighbor grid

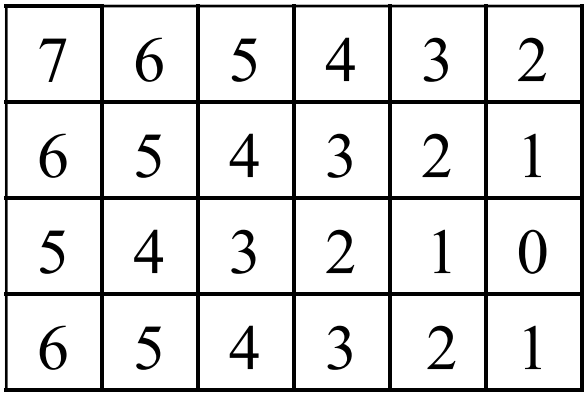
Background: A*

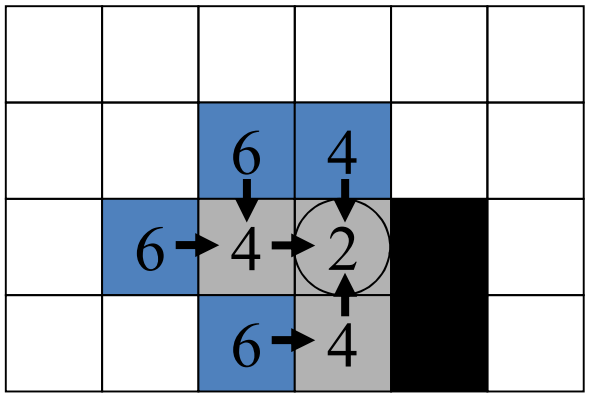
- Fourth iteration of A*

order of expansions









g-values + h-values = f-values

cost of the shortest path in the search tree from the start state to the given state

generated but not expanded state (OPEN list)

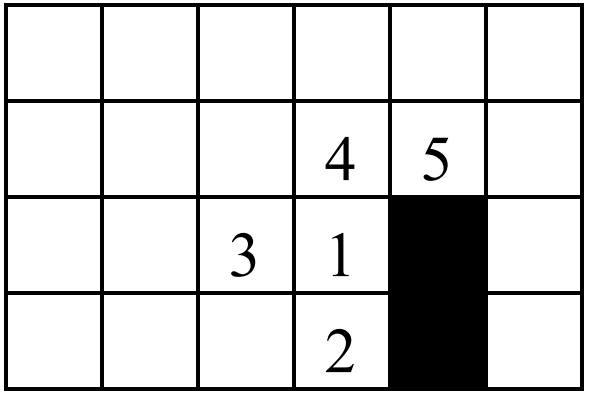
expanded state (CLOSED list)

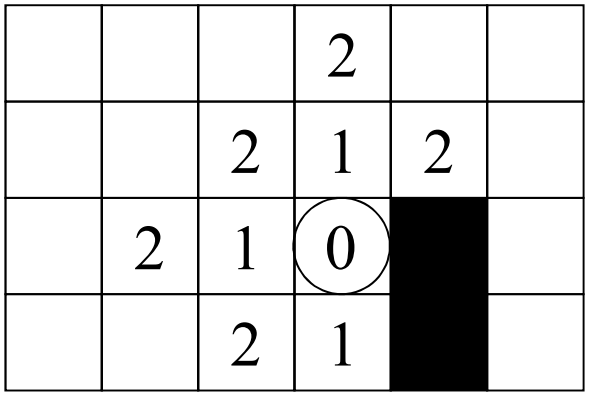
4-neighbor grid

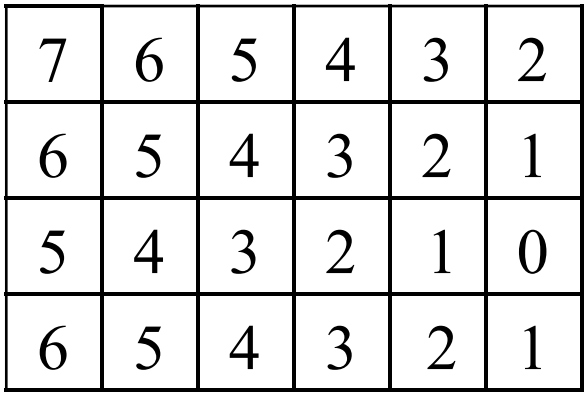
Background: A*

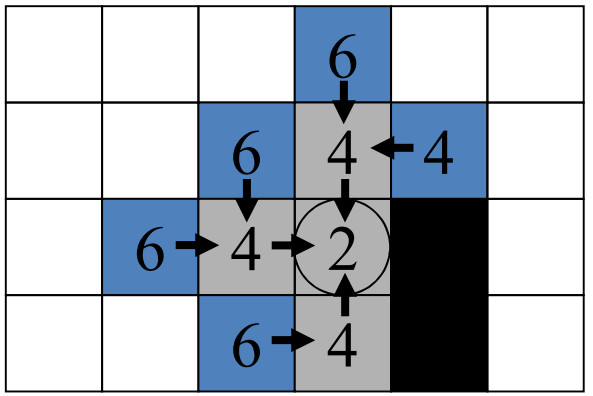
- Fifth iteration of A*

order of expansions









g-values + h-values = f-values

cost of the shortest path in the search tree from the start state to the given state

generated but not expanded state (OPEN list)

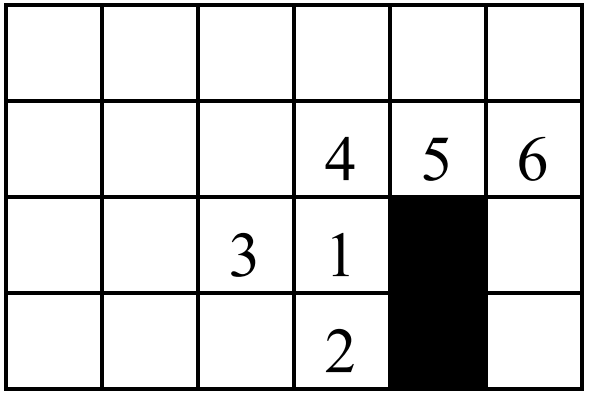
expanded state (CLOSED list)

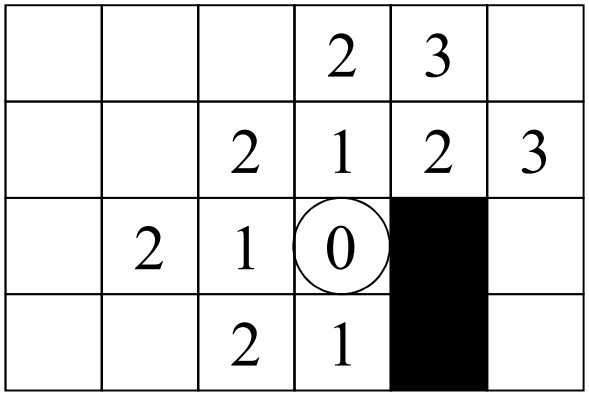
4-neighbor grid

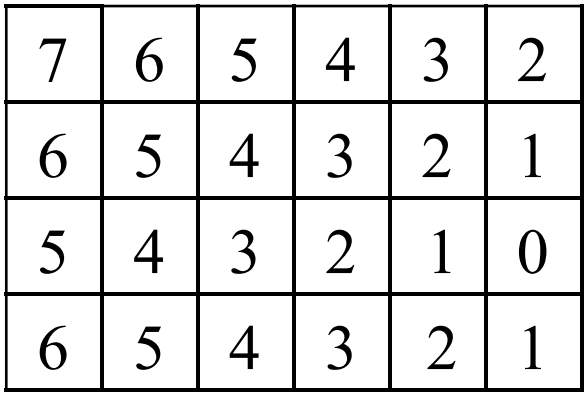
Background: A*

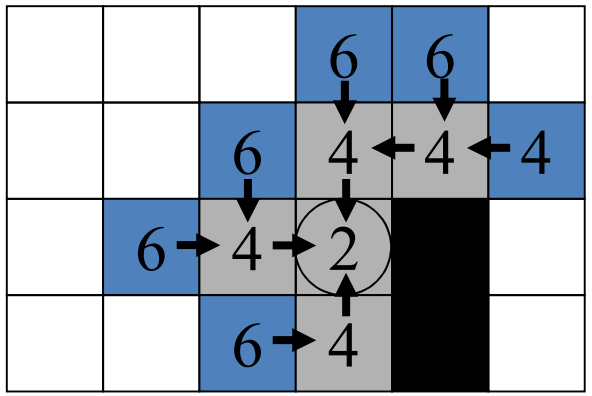
- Sixth iteration of A*

order of expansions









g-values + h-values = f-values

cost of the shortest path in the search tree from the start state to the given state

generated but not expanded state (OPEN list)

expanded state (CLOSED list)

4-neighbor grid

Background: A*

order of expansions

			4	5	6
		3	1	■	(7)
			2	■	

- Seventh and last iteration of A*

			2	3	4
		2	1	2	3
	2	1	0	■	4
		2	1	■	

7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0
6	5	4	3	2	1

			6	6	6
		6	4	4	4
	6	4	2	■	(4)
		6	4	■	

g-values + h-values = f-values

cost of the shortest path in the search tree from the start state to the given state

generated but not expanded state (OPEN list)

expanded state (CLOSED list)

4-neighbor grid

Background: A*

Uniform-cost search

Breadth-first search

7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0
6	5	4	3	2	1

5	4	3	2	2	2
5	4	3	2	1	1
5	4	3	2	1	0
5	4	3	2	1	1

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Manhattan Distance

			4	5	6
		3	1		(7)
			2		

Octile Distance

			6		
			3	4	7
		5	1		(8)
			2		

Zero h-values

	18	13	8	14	19
17	12	7	4	9	15
11	6	3	1		(20)
16	10	5	2		

more informed (dominating)

4-neighbor grid

Background: A*

- We say that h-values $h_1(s)$ dominate h-values $h_2(s)$ iff $h_1(s) \geq h_2(s)$ for all states s.
- A* with consistent h-values h(s)
  - expands every state at most once
  - has found a shortest path from the start state to a state when it is about to expand the state
  - has found a shortest path from the start state to the goal state when it terminates
  - expands no more states than with consistent h-values dominated by the h-values h(s)

Background: Weighted A*

- Finds paths that are at most by a factor of w longer than optimal (= bounded-suboptimal paths)

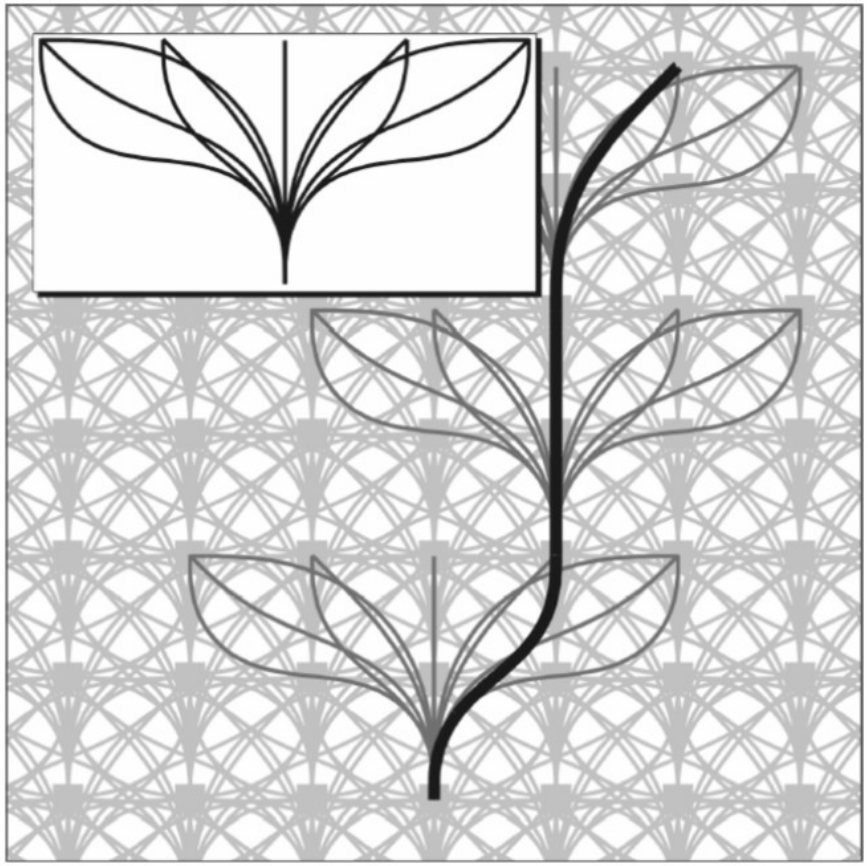
1. Create a search tree that contains only the start state
2. Pick a generated but not yet expanded state s with the smallest f-value, where f(s) = g(s) + w h(s) for a weight parameter w ≥ 1
3. If state s is a goal state: stop
4. Expand state s
5. Go to 2

Background: Weighted A*

- For consistent h-values, weighted A*
  - typically expands many fewer states the larger w is
  - finds a path from the start state to the goal state that is at most a factor of w longer than minimal

Background: Motion Planning

- State lattices



[Pivtoraiko]

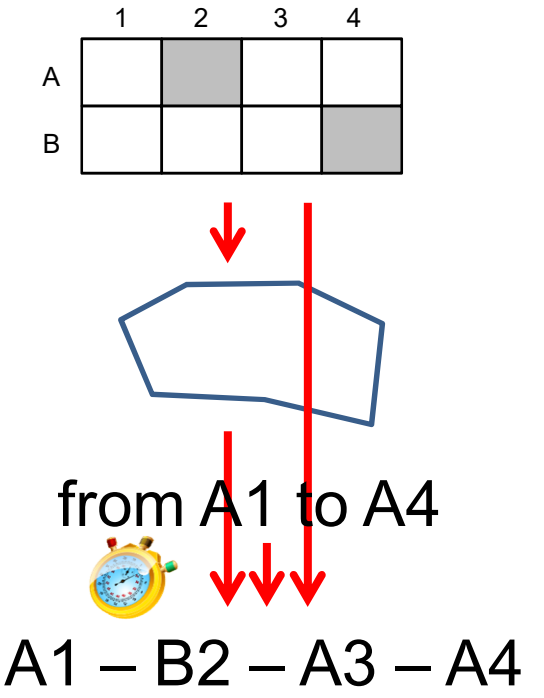
TOC

- Single-robot path planning
  - Runtime matters
    - Hierarchical search (via preprocessing)
    - Incremental search (via using experience with previous similar searches)
  - Quality-runtime tradeoff matters
    - Any-angle search
- Multi-robot path planning and execution

Grid-Based Path Planning Competition

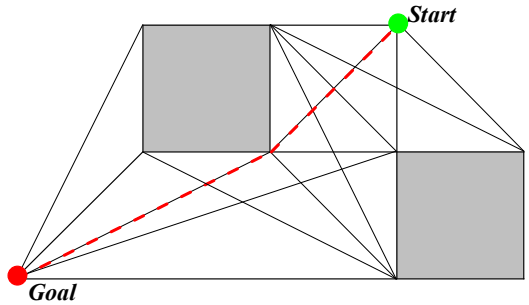
movingai.com/GPPC

- Offline phase
  - Map is given (without start and goal locations)
  - Preprocessing is performed for 30 minutes
- Online phase
  - Start and goal locations are given
  - Path-planning problem is solved
- Main evaluation criteria
  - How close to optimal is the path?
  - How fast is the path found?
  - How much memory is used?



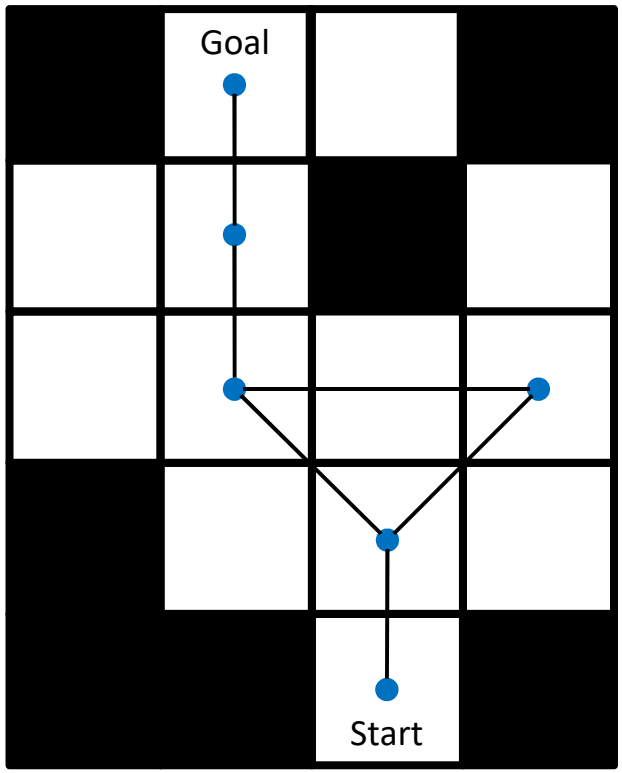
[work by University of Denver, not me]

Subgoal Graphs

continuous terrain	grid graphs
	
visibility graphs →	(simple) subgoal graphs

Subgoal Graphs

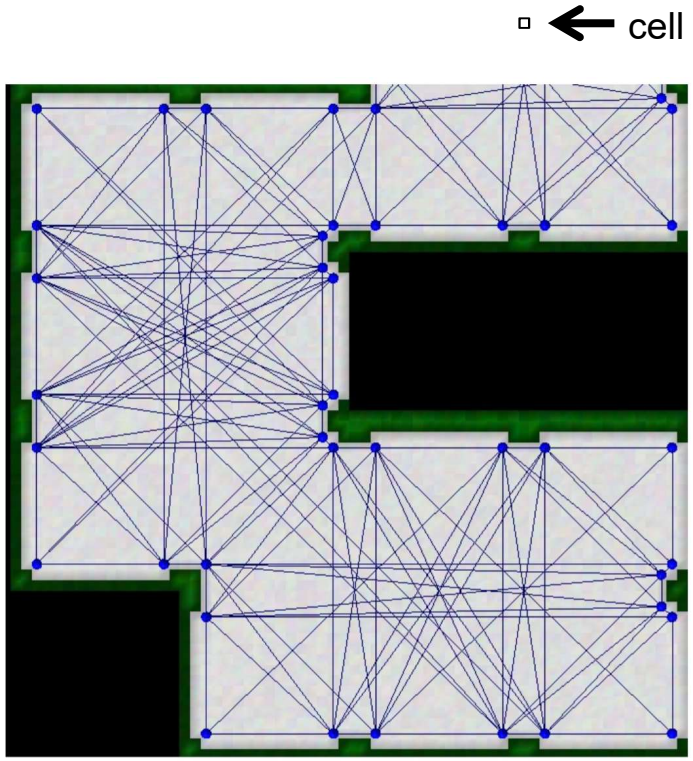
- Place states at the convex corners of blocked cells and in the start and goal cells
- Connect two states iff all shortest grid paths between them on an empty grid neither pass through blocked cells nor through other states
  (= h-reachability)



8-neighbor grid

Subgoal Graphs

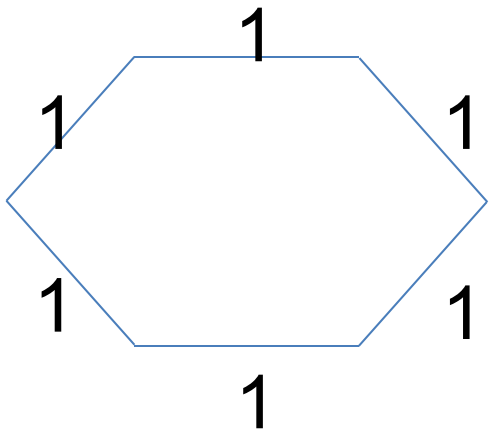
- Place states at the convex corners of blocked cells and in the start and goal cells
- Connect two states iff all shortest grid paths between them on an empty grid neither pass through blocked cells nor through other states
  (= h-reachability)



8-neighbor grid

Searching Subgoal Graphs

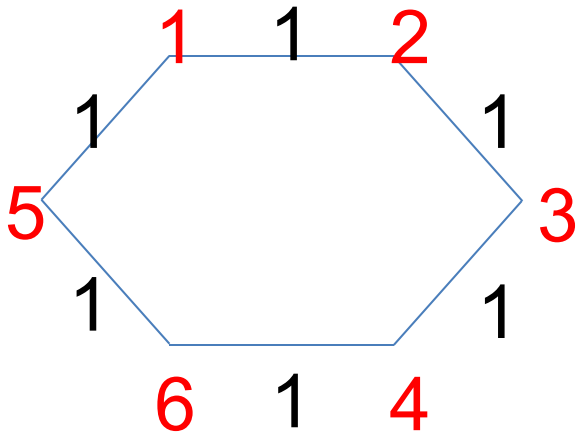
- Contraction hierarchies [Geisberger et al.]
- Offline phase:
  - Transform the graph into a contraction hierarchy



[work by Universität Karlsruhe, not me]

Searching Subgoal Graphs

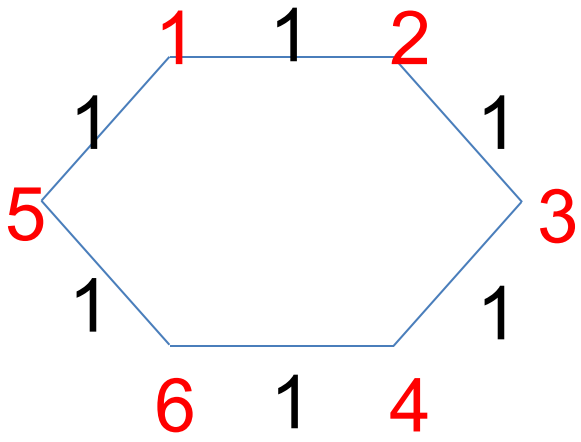
- Contraction hierarchies [Geisberger et al.]
- Offline phase:
  - Transform the graph into a contraction hierarchy

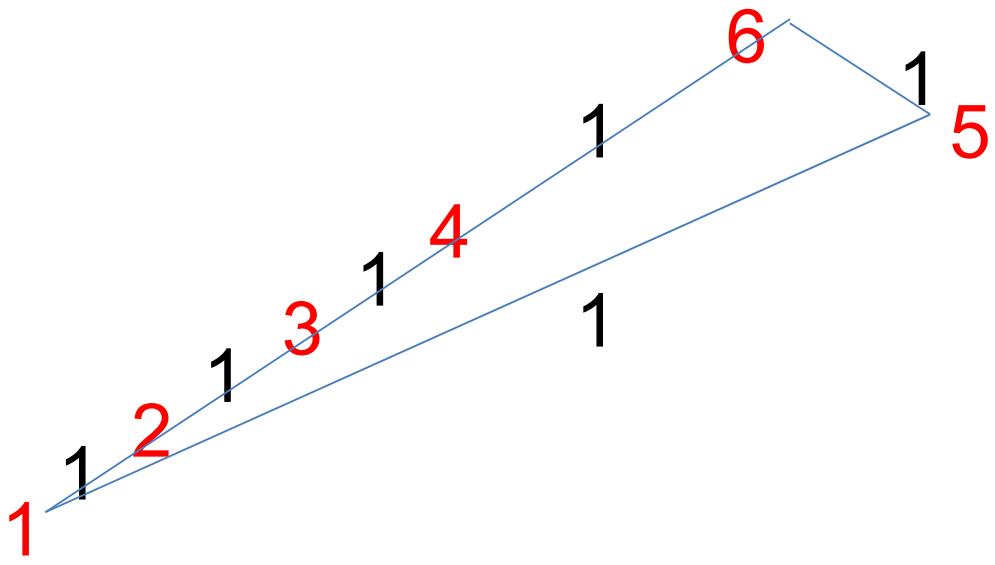


[work by Universität Karlsruhe, not me]

Searching Subgoal Graphs

- Contraction hierarchies [Geisberger et al.]
- Offline phase:
  - Transform the graph into a contraction hierarchy

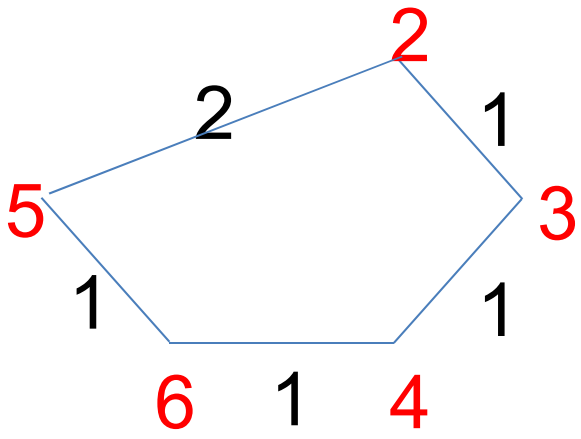


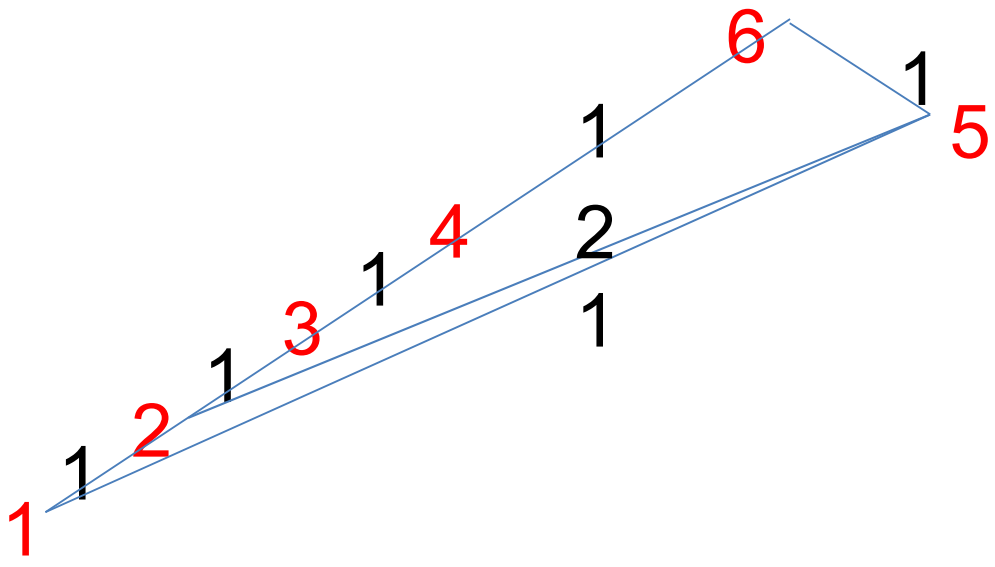


[work by Universität Karlsruhe, not me]

Searching Subgoal Graphs

- Contraction hierarchies [Geisberger et al.]
- Offline phase:
  - Transform the graph into a contraction hierarchy

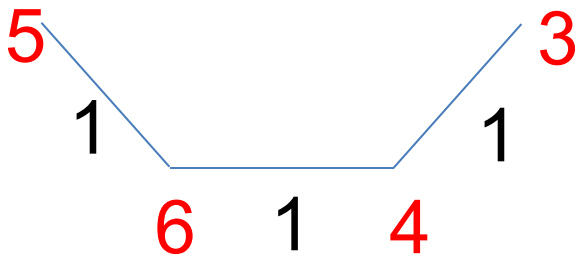


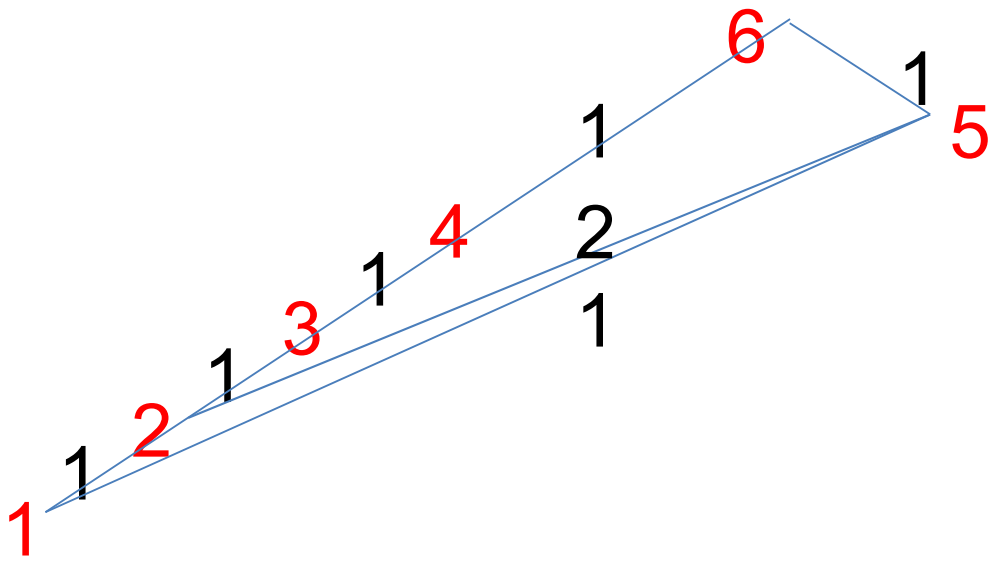


[work by Universität Karlsruhe, not me]

Searching Subgoal Graphs

- Contraction hierarchies [Geisberger et al.]
- Offline phase:
  - Transform the graph into a contraction hierarchy

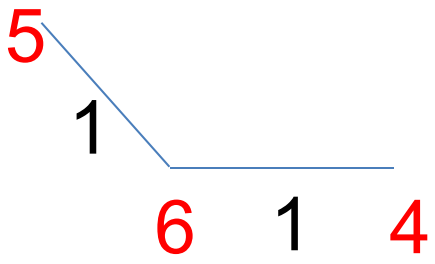


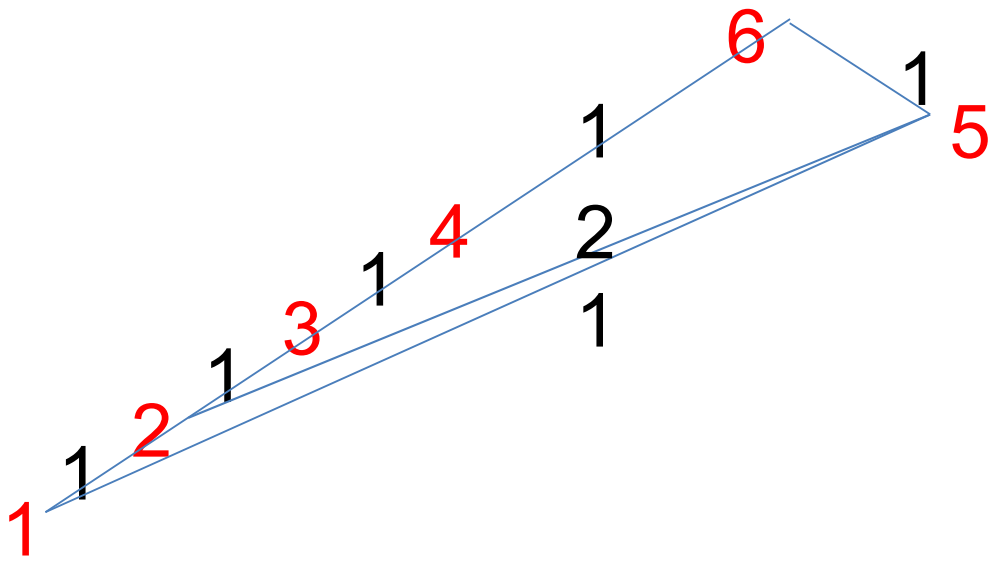


[work by Universität Karlsruhe, not me]

Searching Subgoal Graphs

- Contraction hierarchies [Geisberger et al.]
- Offline phase:
  - Transform the graph into a contraction hierarchy

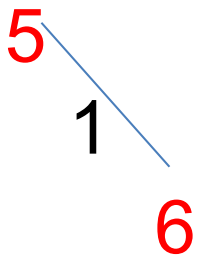


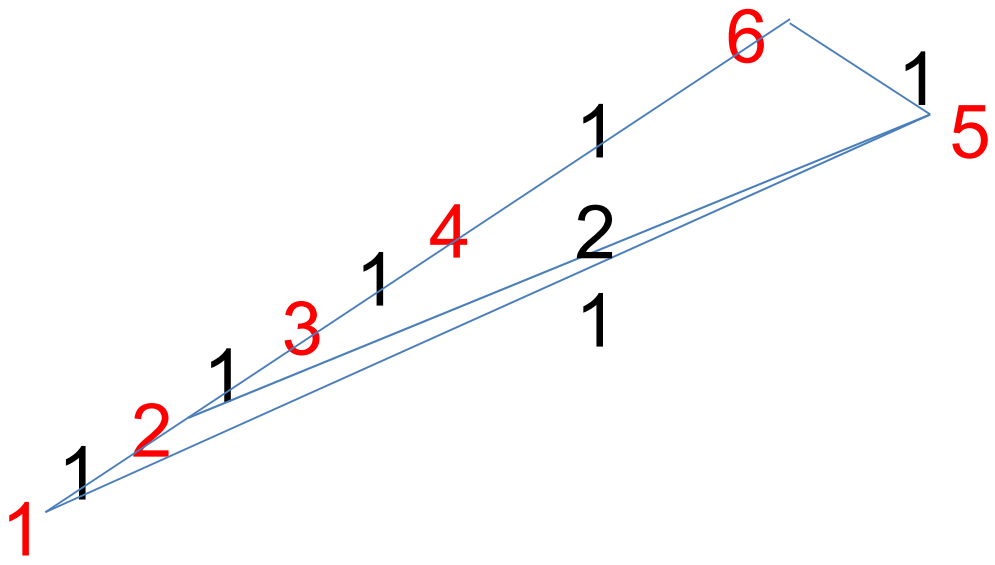


[work by Universität Karlsruhe, not me]

Searching Subgoal Graphs

- Contraction hierarchies [Geisberger et al.]
- Offline phase:
  - Transform the graph into a contraction hierarchy

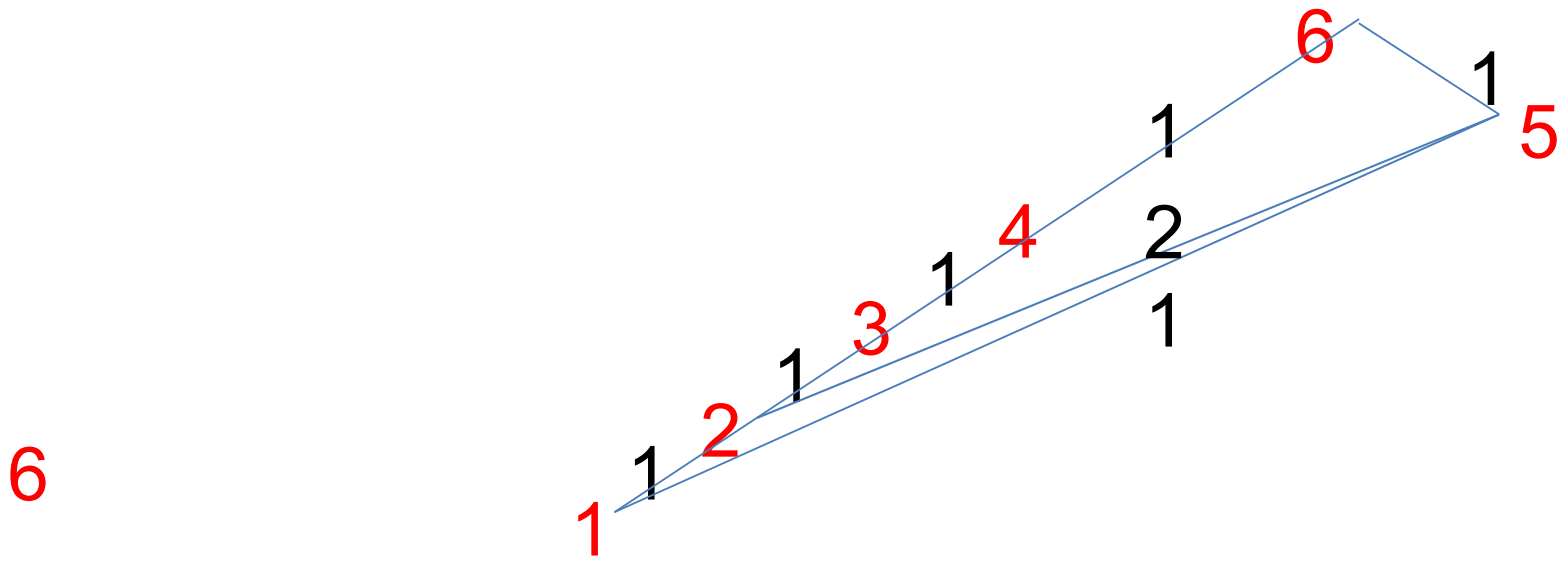




[work by Universität Karlsruhe, not me]

Searching Subgoal Graphs

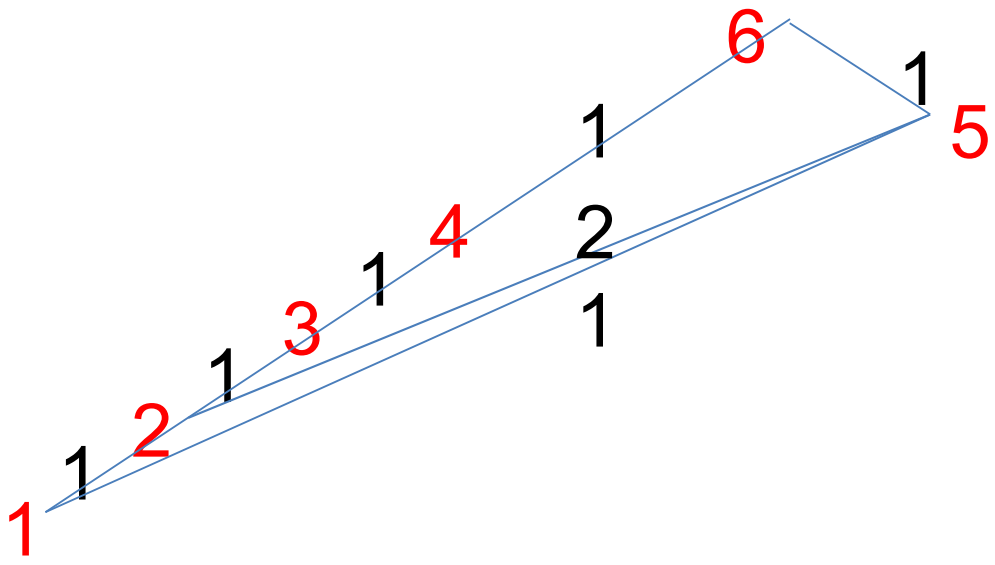
- Contraction hierarchies [Geisberger et al.]
- Offline phase:
  - Transform the graph into a contraction hierarchy



[work by Universität Karlsruhe, not me]

Searching Subgoal Graphs

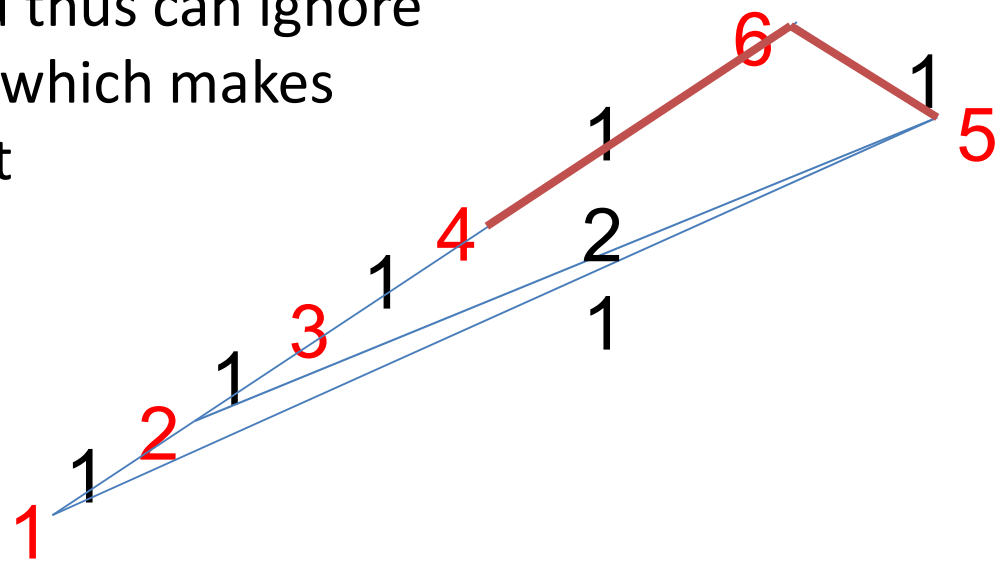
- Contraction hierarchies [Geisberger et al.]
- Offline phase:
  - Transform the graph into a contraction hierarchy



[work by Universität Karlsruhe, not me]

Searching Subgoal Graphs

- Contraction hierarchies [Geisberger et al.]
- Online phase: Find a shortest path from 4 to 5
  - One only needs to consider all shortest up-down paths and thus can ignore most of the graph, which makes the search very fast



[work by Universität Karlsruhe, not me]

TOC

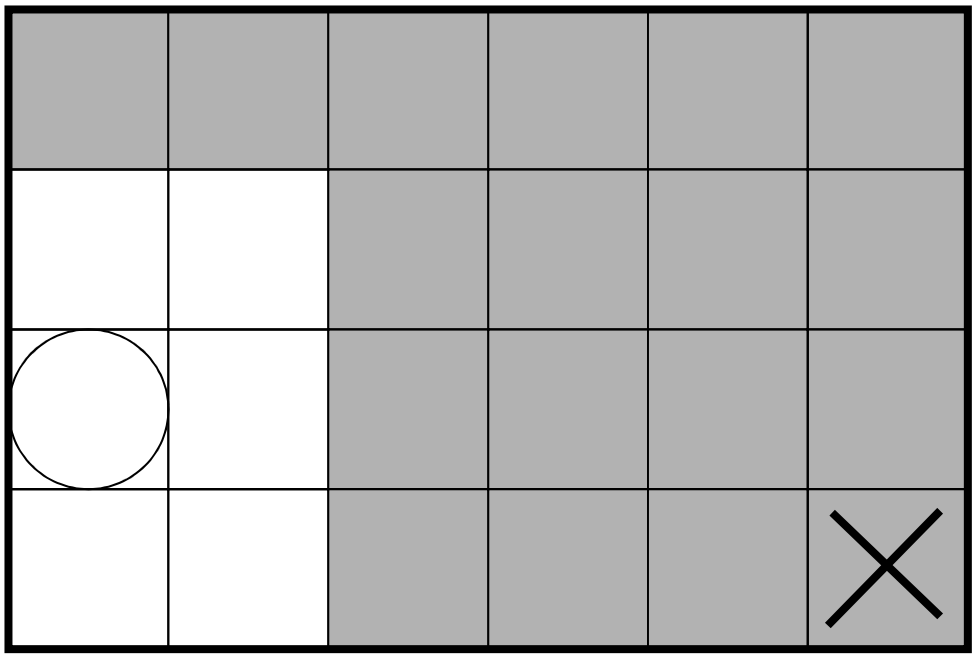
- Single-robot path planning
  - Runtime matters
    - Hierarchical search (via preprocessing)
    - Incremental search (via using experience with previous similar searches)
  - Quality-runtime tradeoff matters
    - Any-angle search
- Multi-robot path planning and execution

Incremental Search

Total Annihilation by Cavedog Entertainment

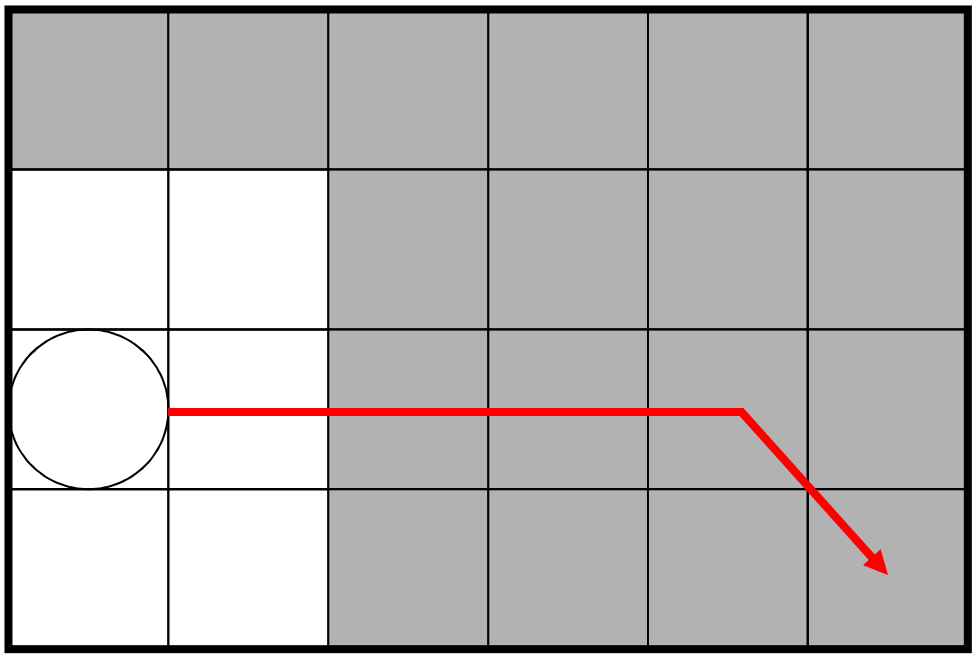
Incremental Search

- Goal-directed navigation
  with the freespace assumption



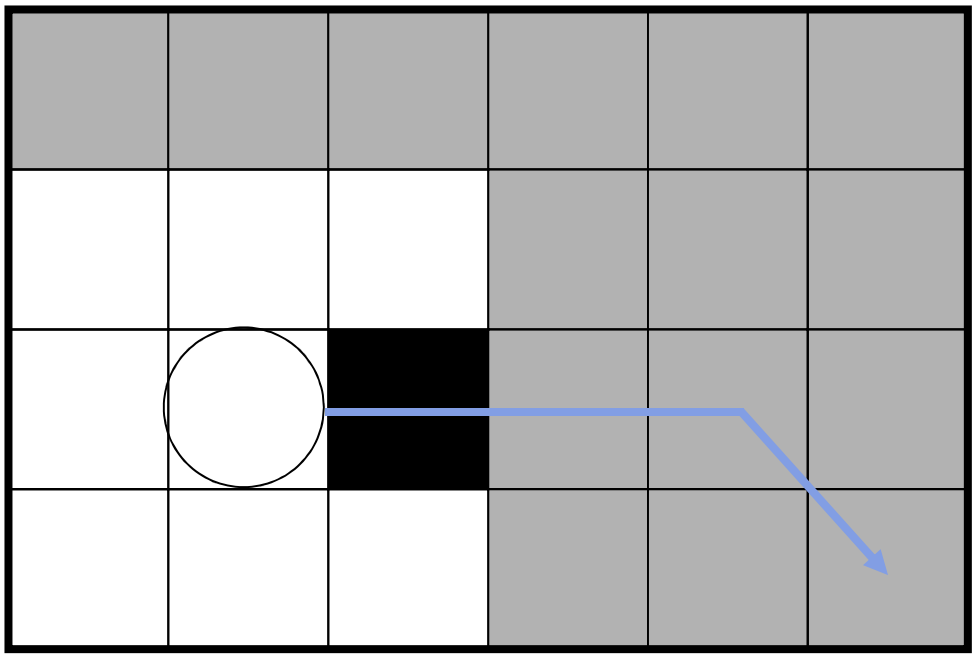
Incremental Search

- Goal-directed navigation
  with the freespace assumption



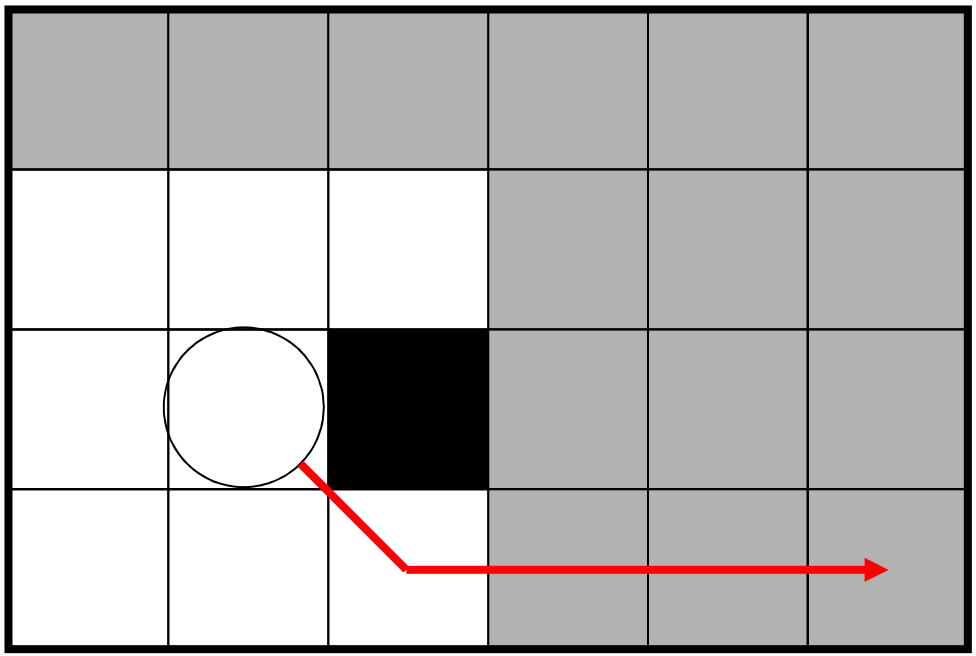
Incremental Search

- Goal-directed navigation
  with the freespace assumption



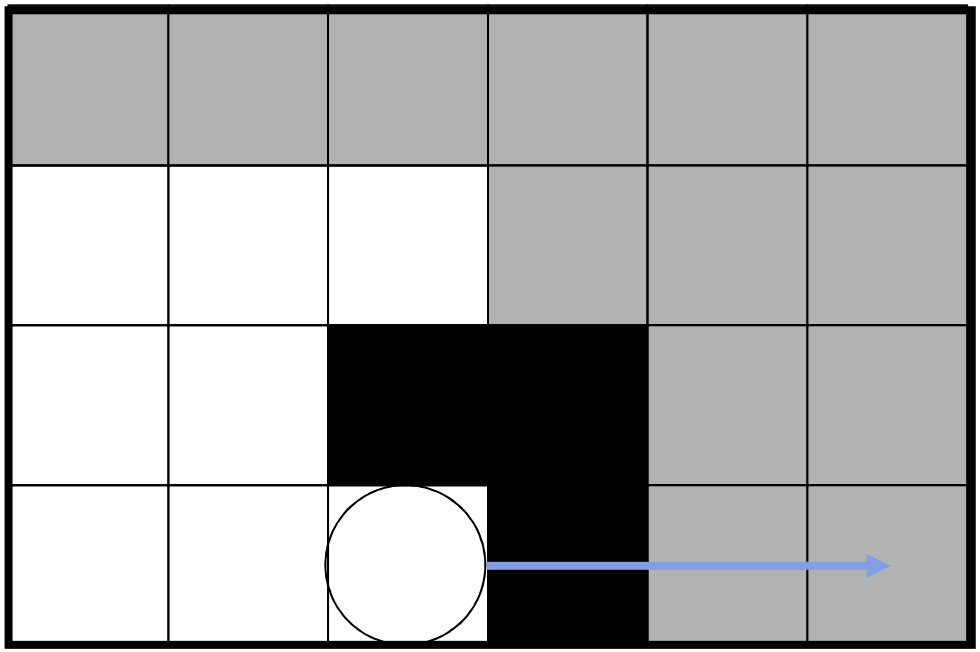
Incremental Search

- Goal-directed navigation
  with the freespace assumption



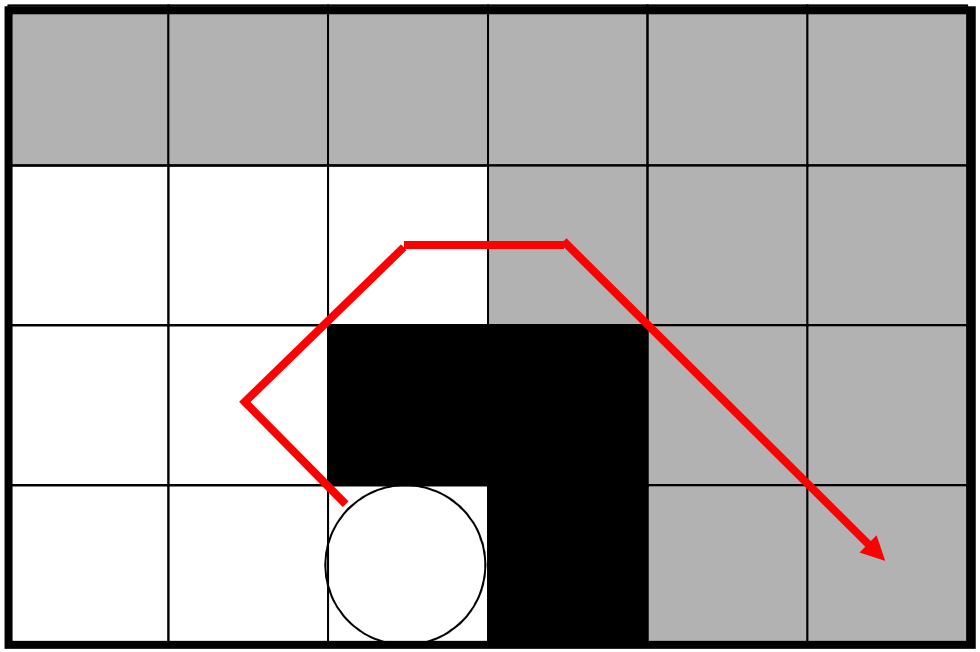
Incremental Search

- Goal-directed navigation
  with the freespace assumption



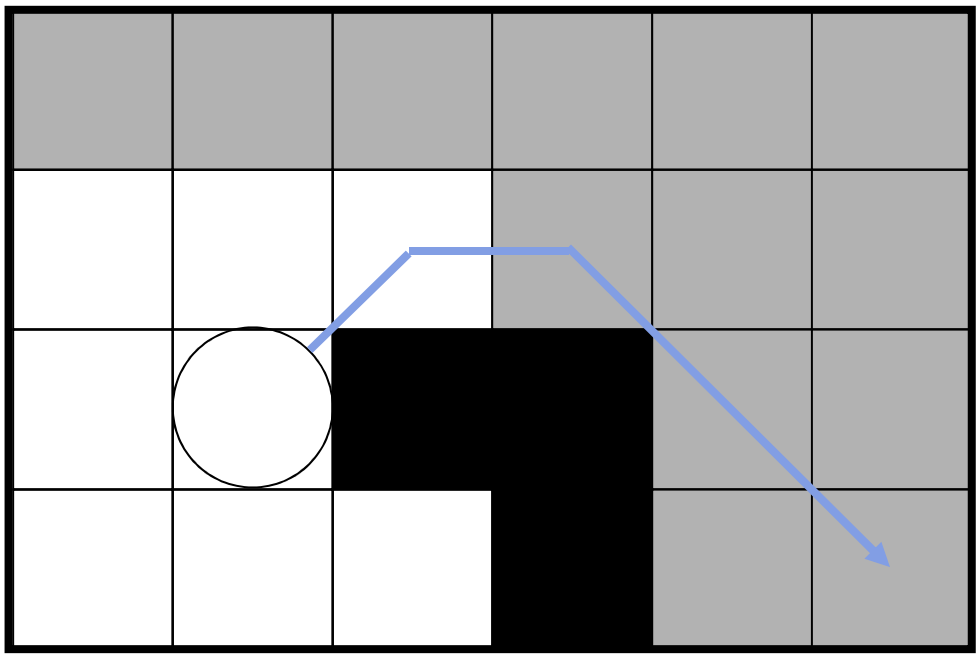
Incremental Search

- Goal-directed navigation
  with the freespace assumption



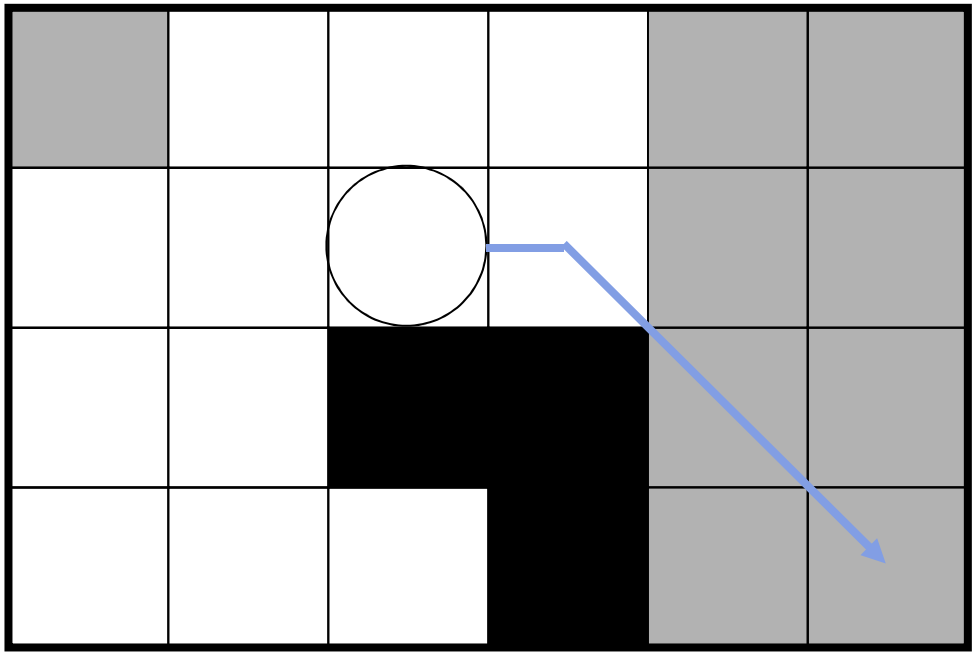
Incremental Search

- Goal-directed navigation
  with the freespace assumption



Incremental Search

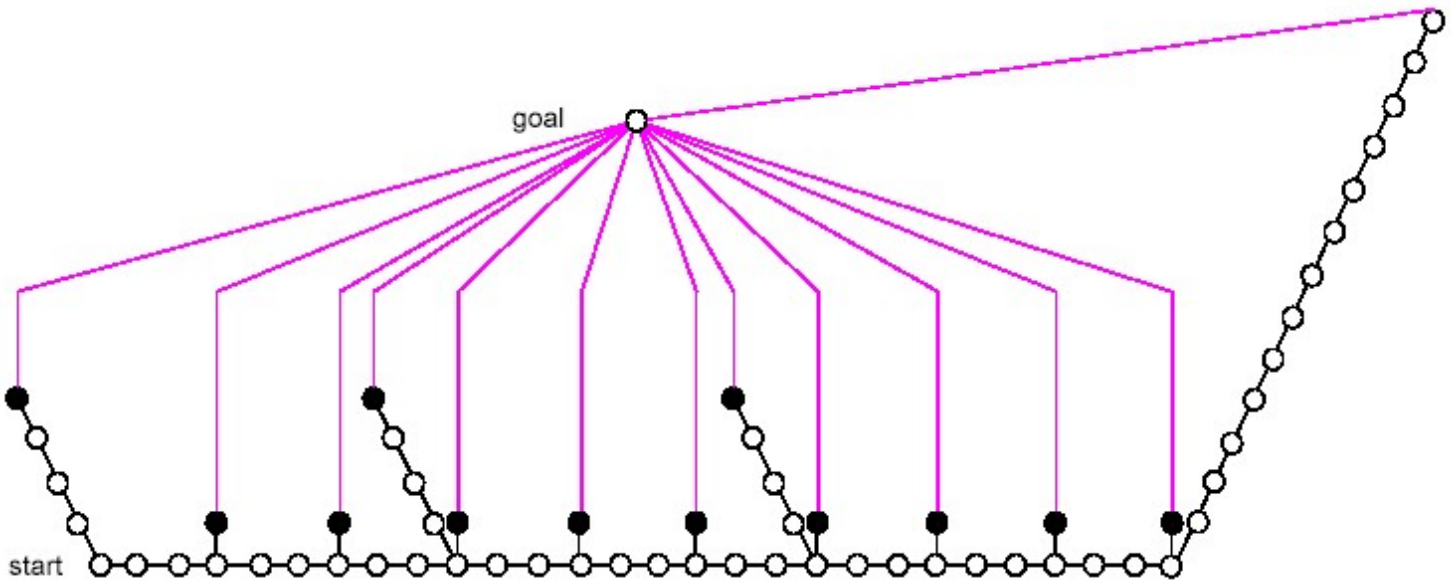
- Goal-directed navigation
  with the freespace assumption



Incremental Search

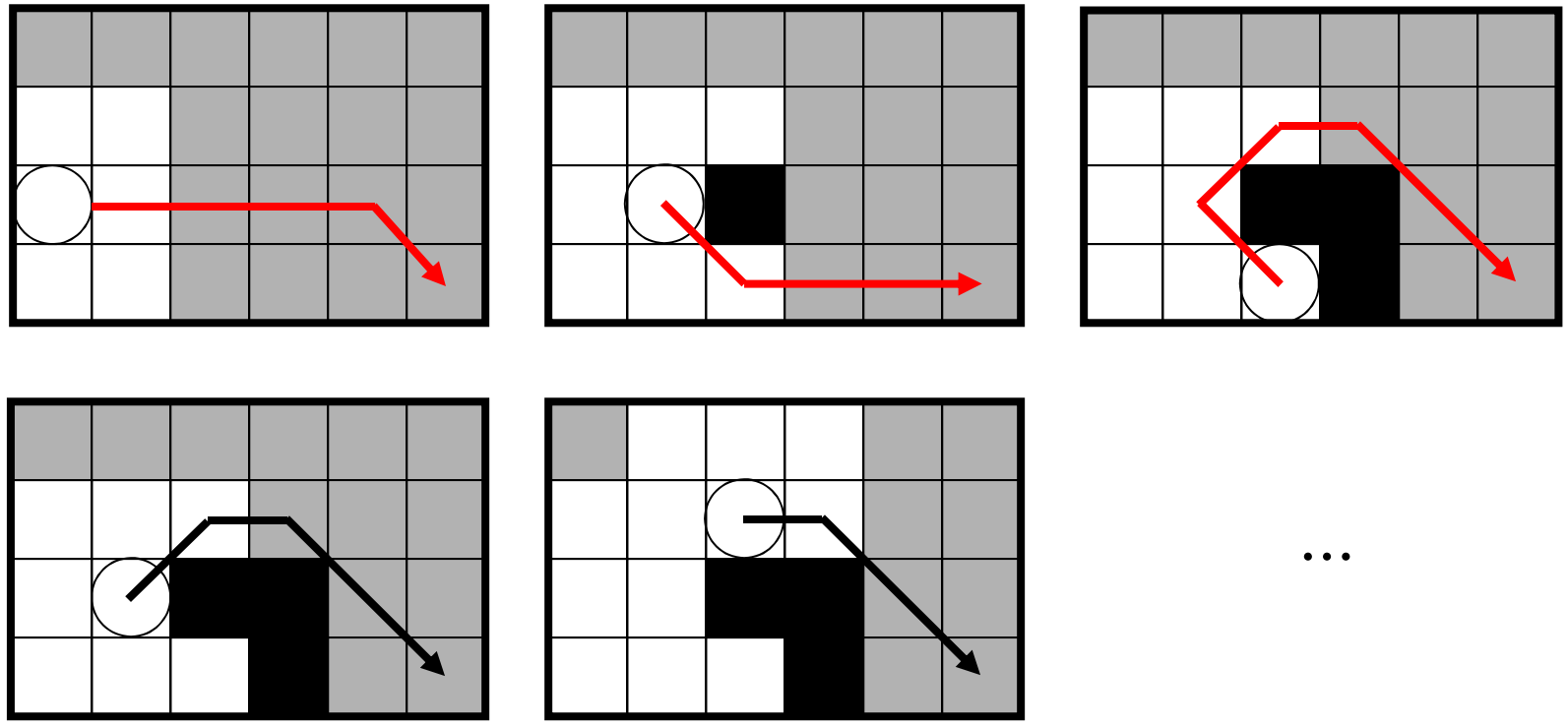
Incremental Search

- Theorem
  The worst-case number of movements is between
  $\Omega(\log(n)/\log\log(n) \times n)$ and $O(\log(n) \times n)$
  on grids, where n is the number of unblocked cells.



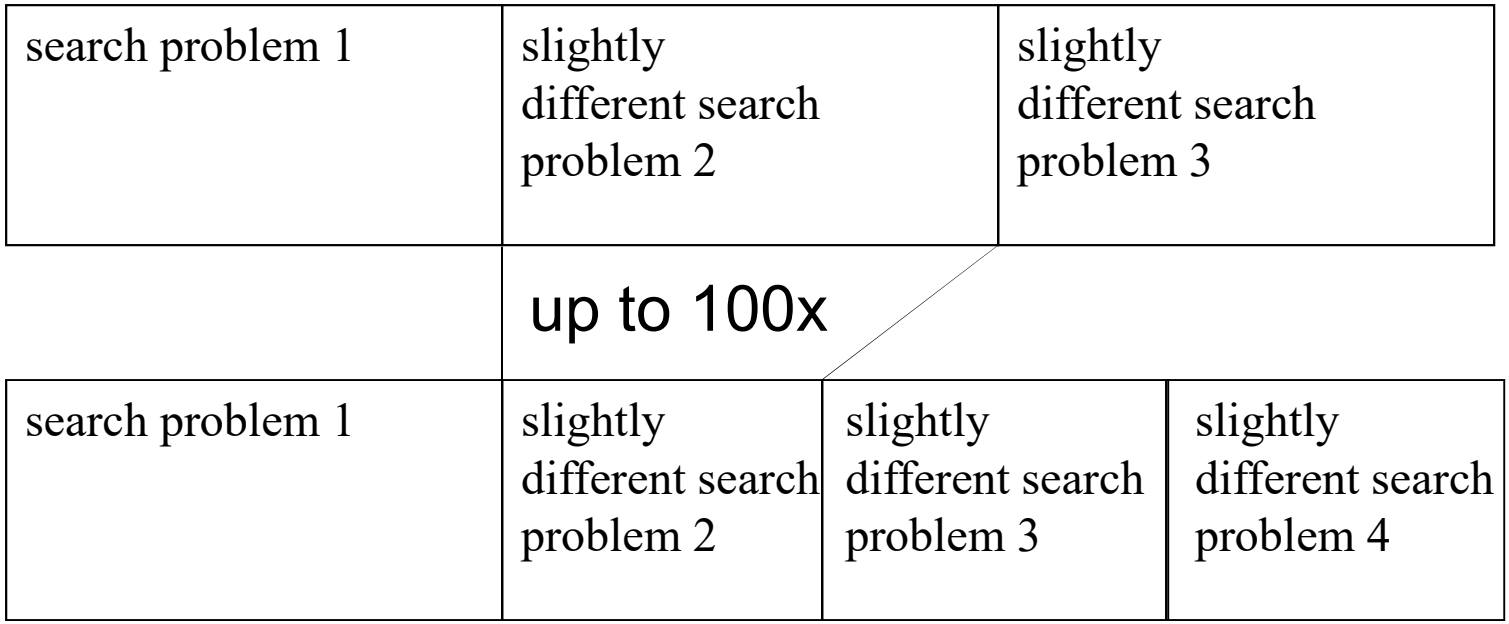
Incremental Search

- Goal-directed navigation
  with the freespace assumption



Incremental Search

- We want to find shortest paths faster than with individual searches from scratch.



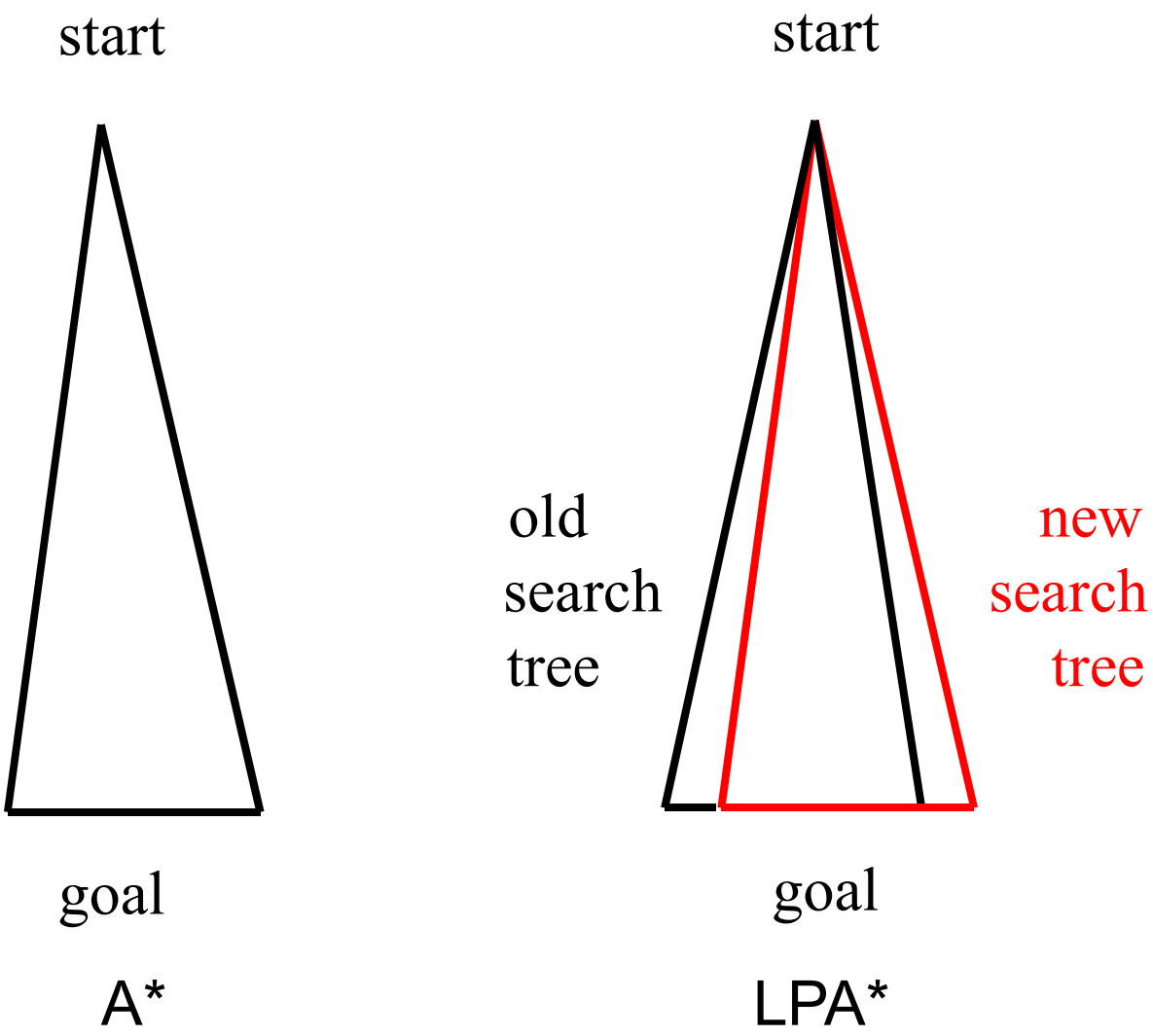
Incremental Search with LPA*

- Lifelong Planning A* (LPA*) speeds up A* searches for a sequence of similar search problems by recalculating only those g-values in the current search that are important for finding a shortest path **and** have changed from the previous search.
- This can often be understood as transforming the search tree from the previous search to the one of the current search.

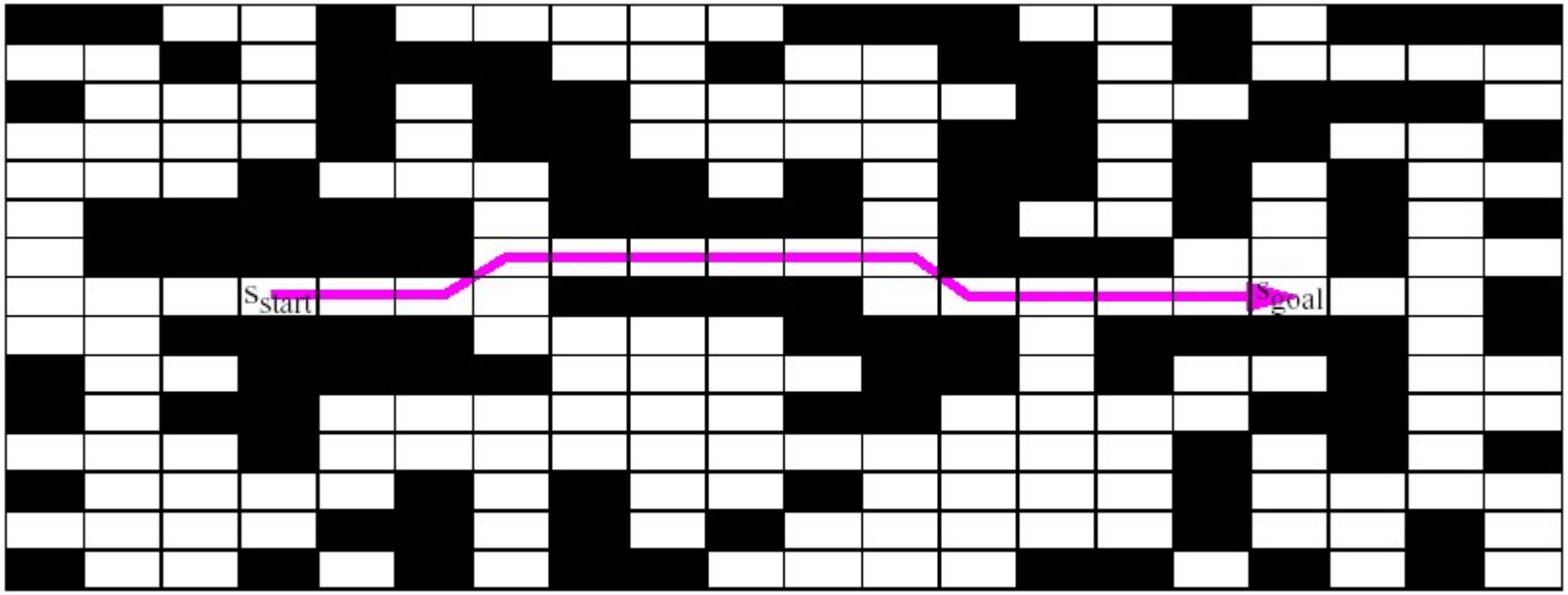
Incremental Search with LPA*

- Anthony Stentz (CMU) and his students:
  D* and extensions; applications
- Sven Koenig (USC) and his students:
  LPA* / D* Lite, FSA* and AA* extension; analysis
- Maxim Likhachev (CMU)
  some of the above, ARA*
- Groups that work on algorithms (e.g. University of Alberta and University of Amsterdam)
- Groups that work on applications, often together with the CMU group (e.g. JPL)

Incremental Search with LPA*

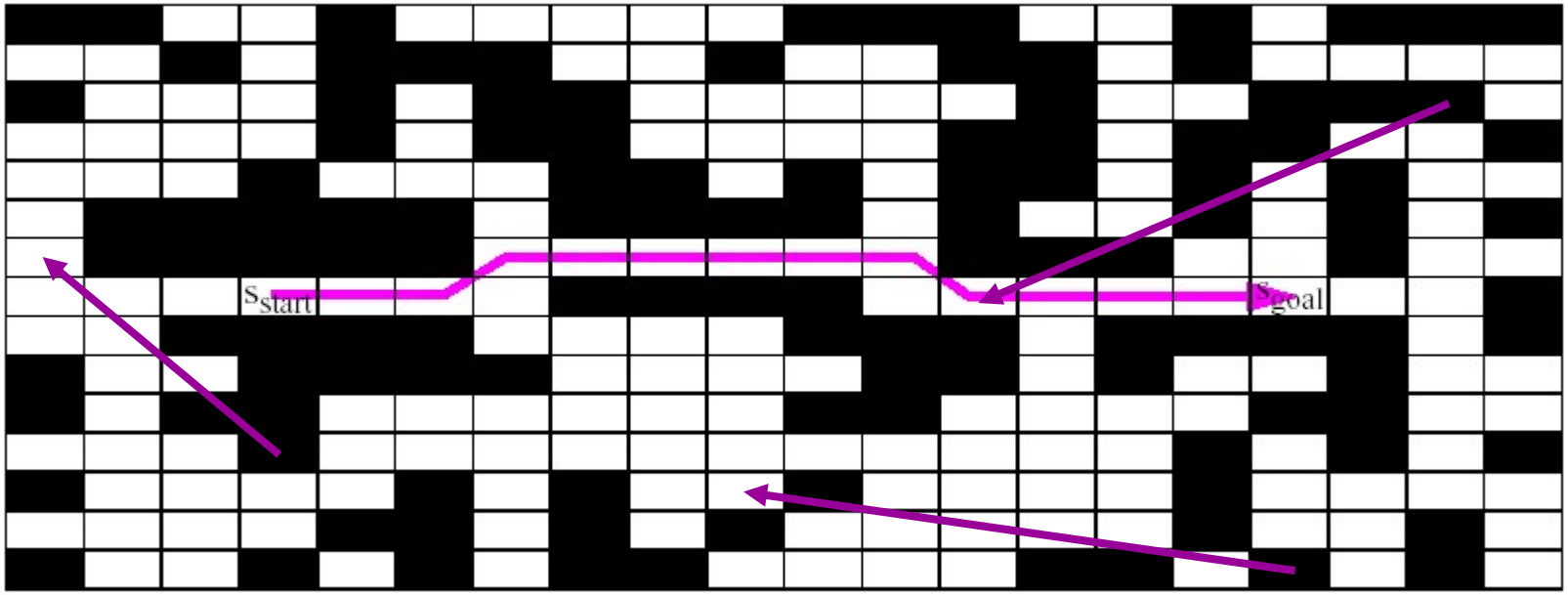


Incremental Search with LPA*



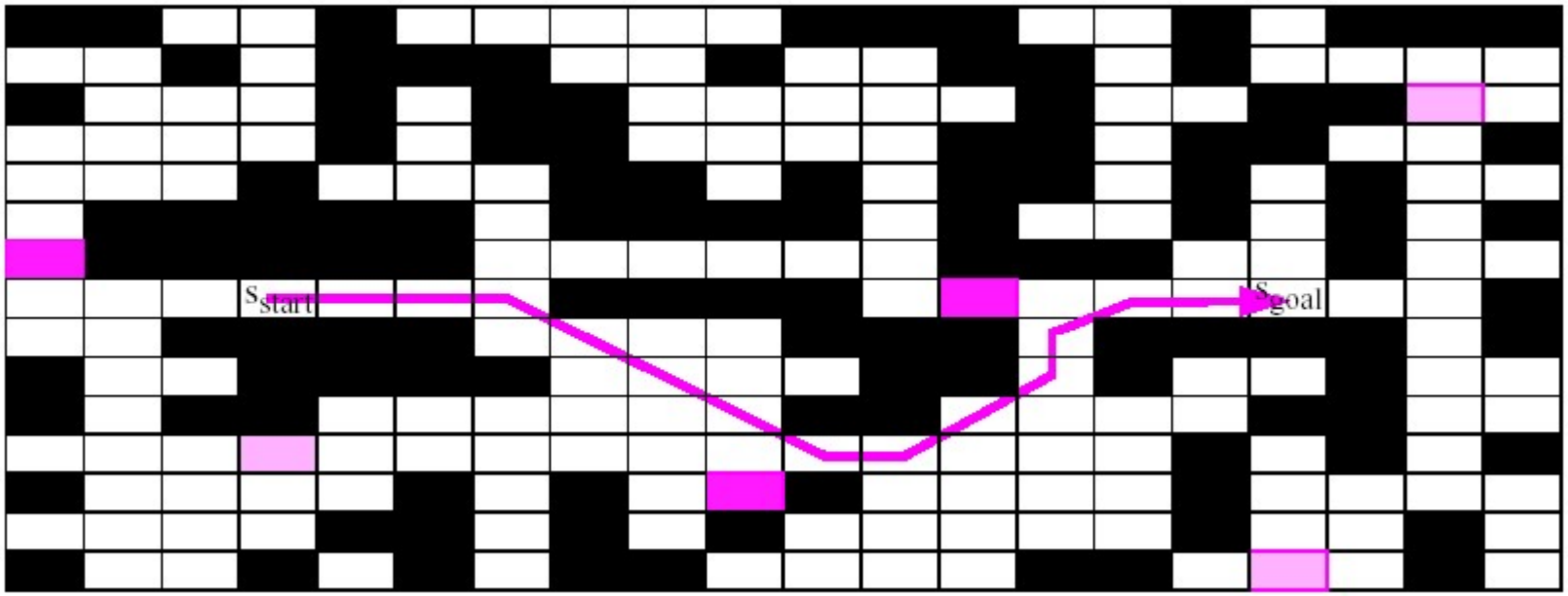
first search

Incremental Search with LPA*



first search

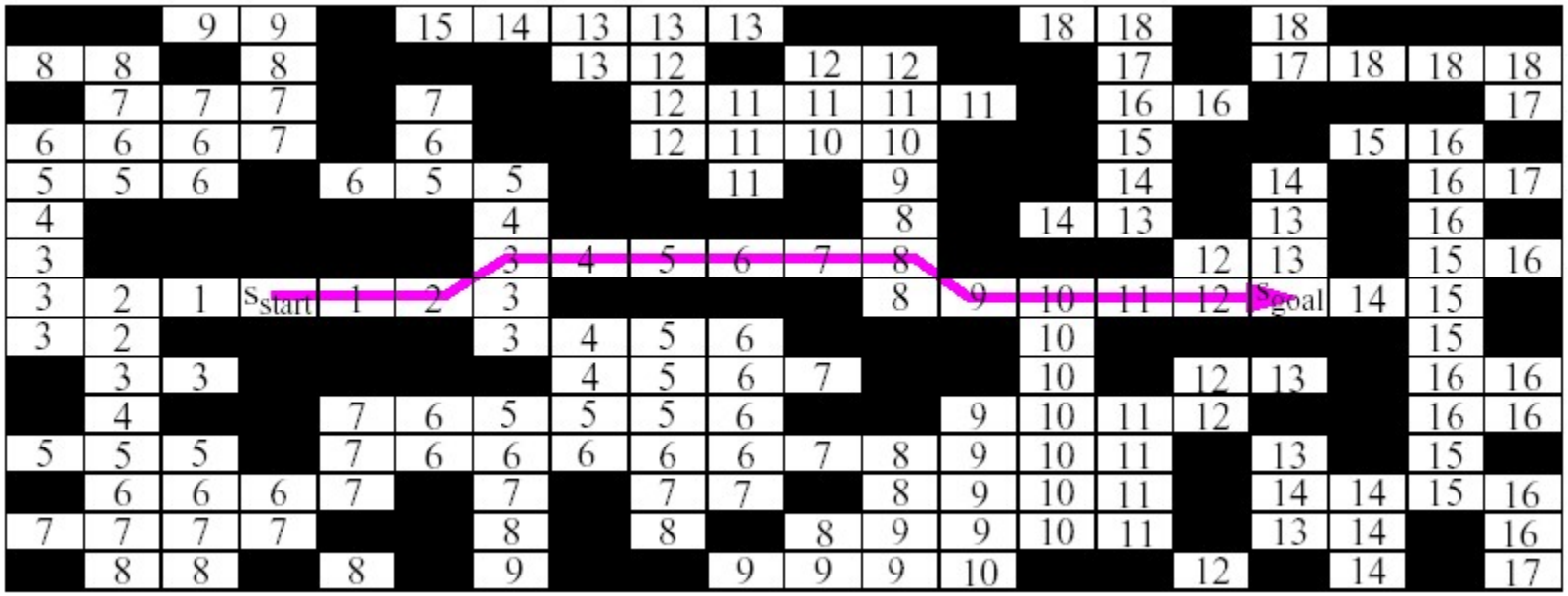
Incremental Search with LPA*



second search

Incremental Search with LPA*

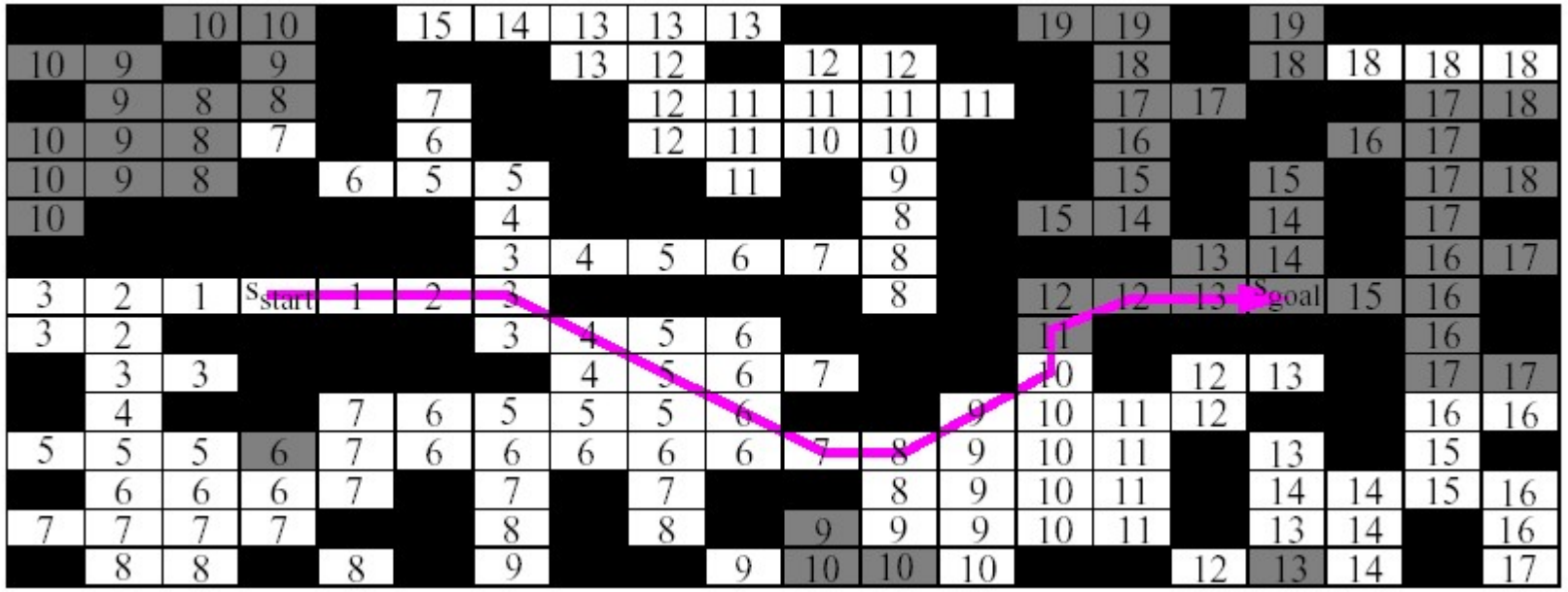
g



first search

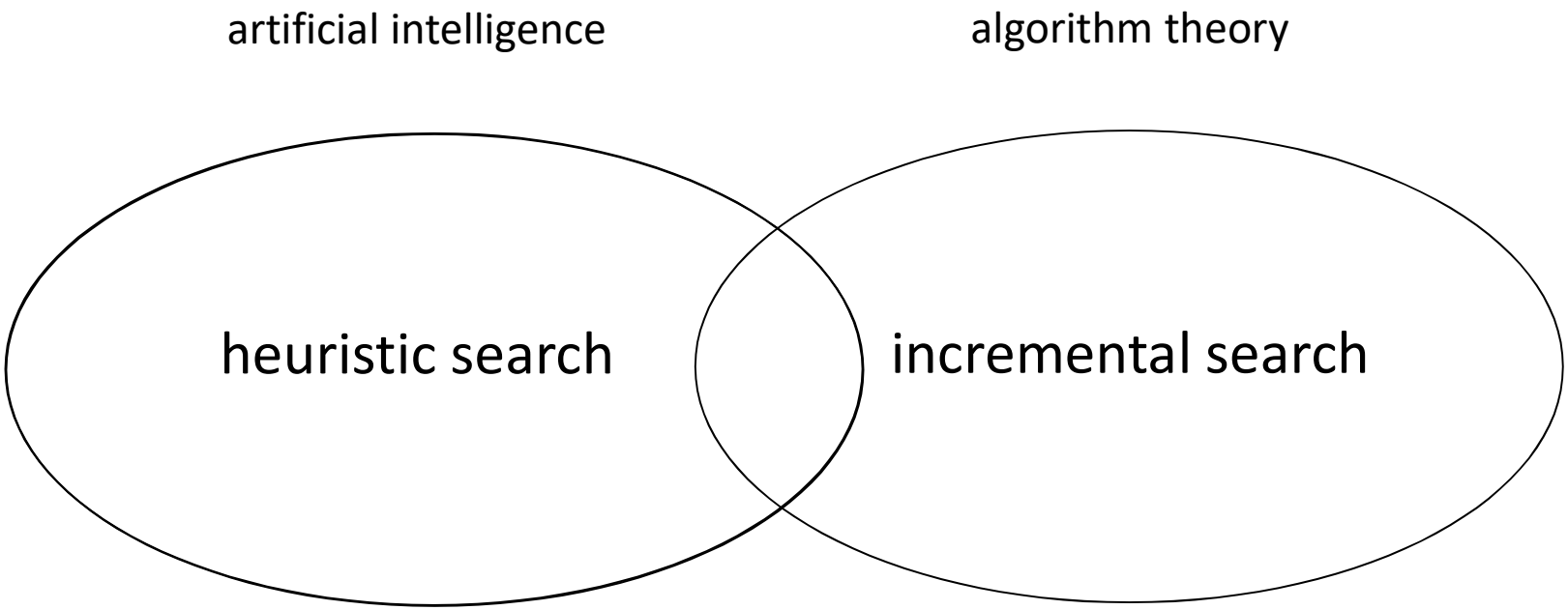
Incremental Search with LPA*

g



second search

Incremental Search with LPA*



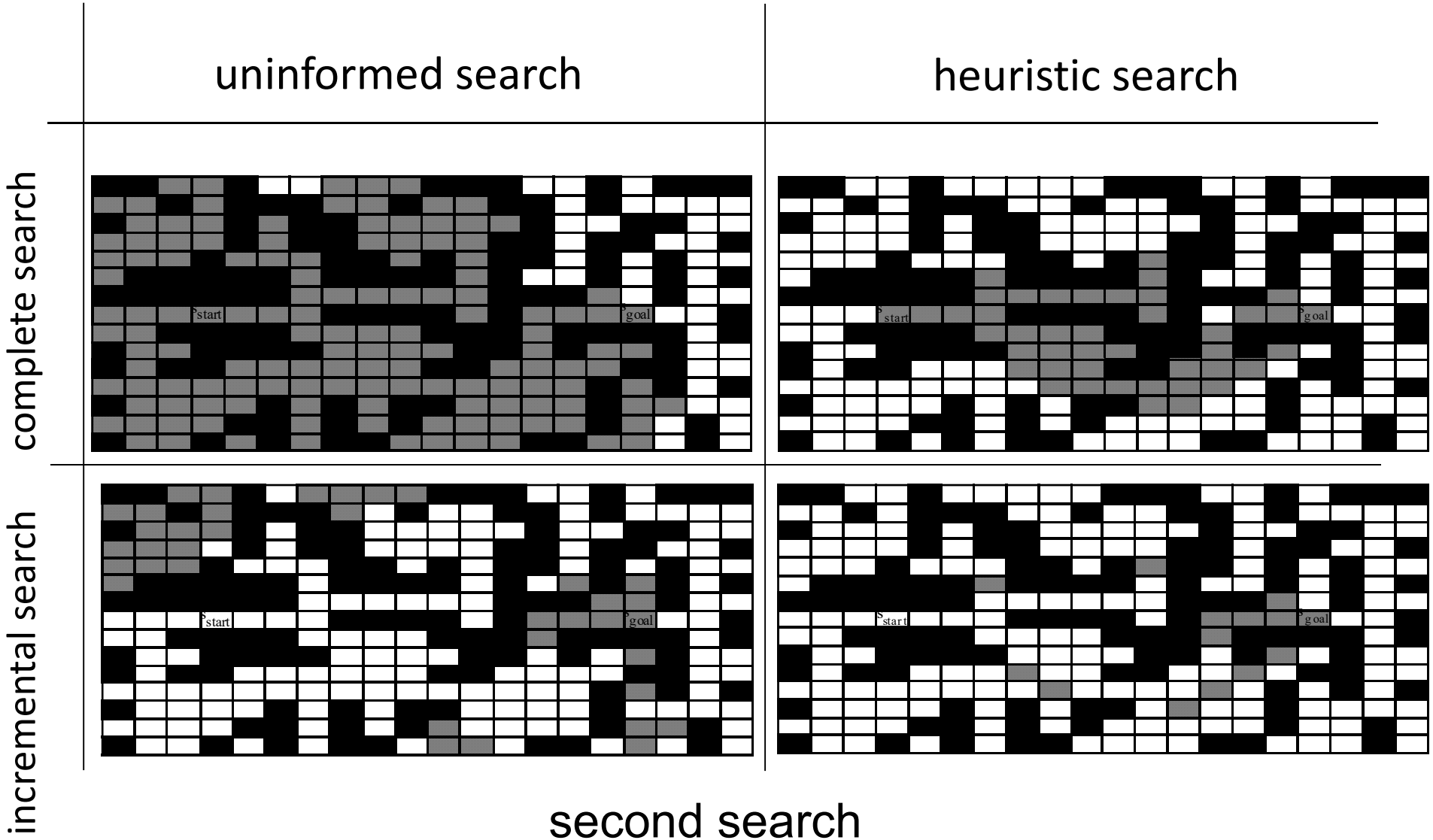
how to search efficiently using h-values to focus the search

how to search efficiently by reusing information from previous similar searches

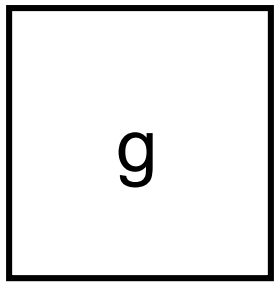
Incremental Search with LPA*

	uninformed search	heuristic search
complete search	breadth-first search	A* [Hart, Nilsson, Raphael, 1968]
incremental search	DynamicSWSF-FP with early termination (our addition) [Ramalingam and Reps, 1996]	Lifelong Planning A* (LPA*)

Incremental Search with LPA*



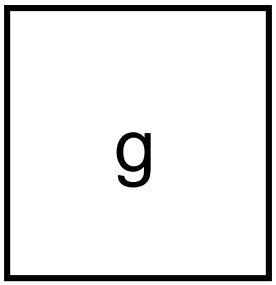
Incremental Search with LPA*



	1	2	start	4	5	6
A	2	1	0	1	2	3
B	3		1			4
C	4		2			5
D	5	4	3	4	5	6
		goal				

4-neighbor grid

Incremental Search with LPA*

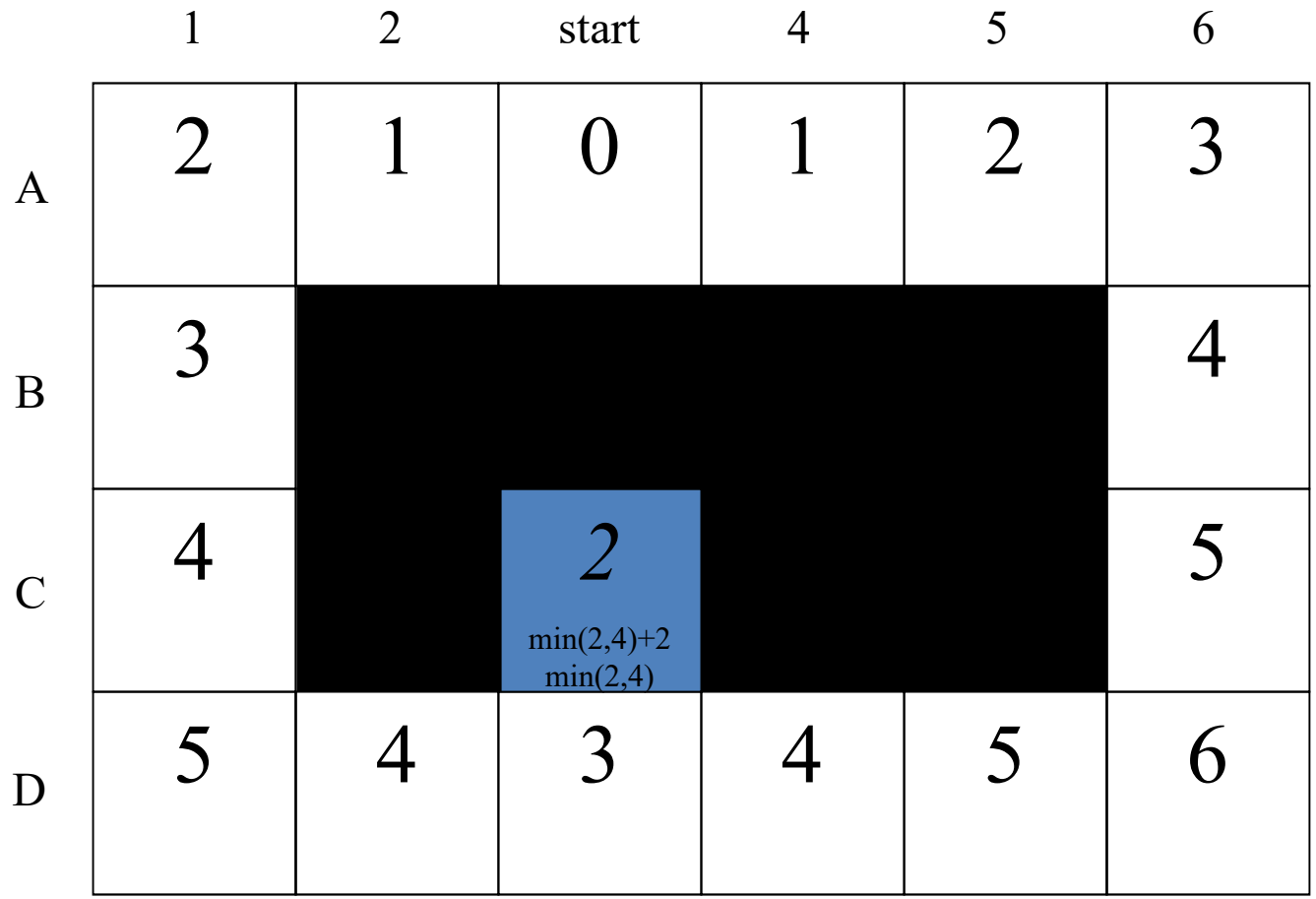


	1	2	start	4	5	6
A	2	1	0	1	2	3
B	3					4
C	4		2			5
D	5	4	3	4	5	6
		goal				

4-neighbor grid

Incremental Search with LPA*

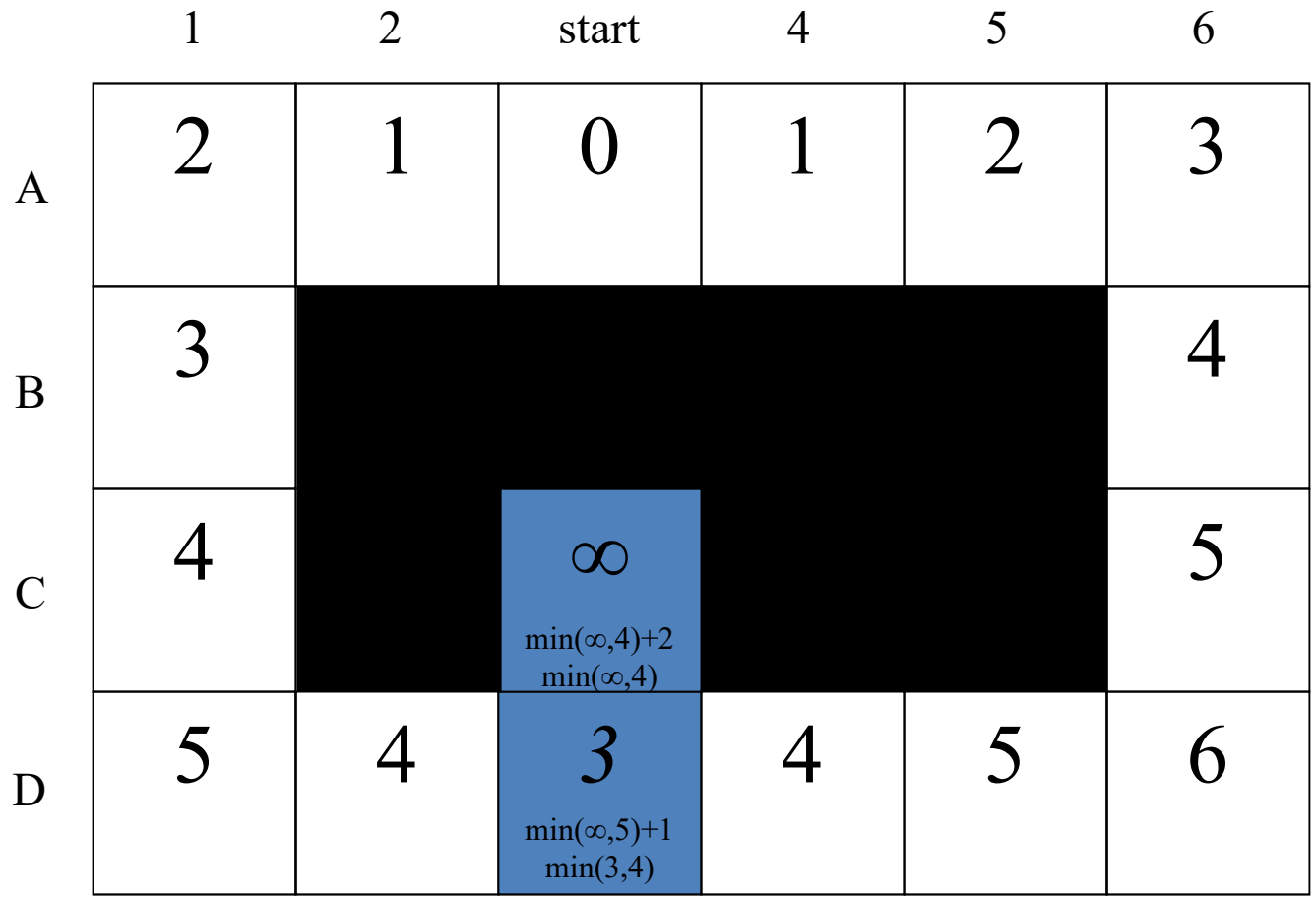
g



4-neighbor grid

Incremental Search with LPA*

g



goal

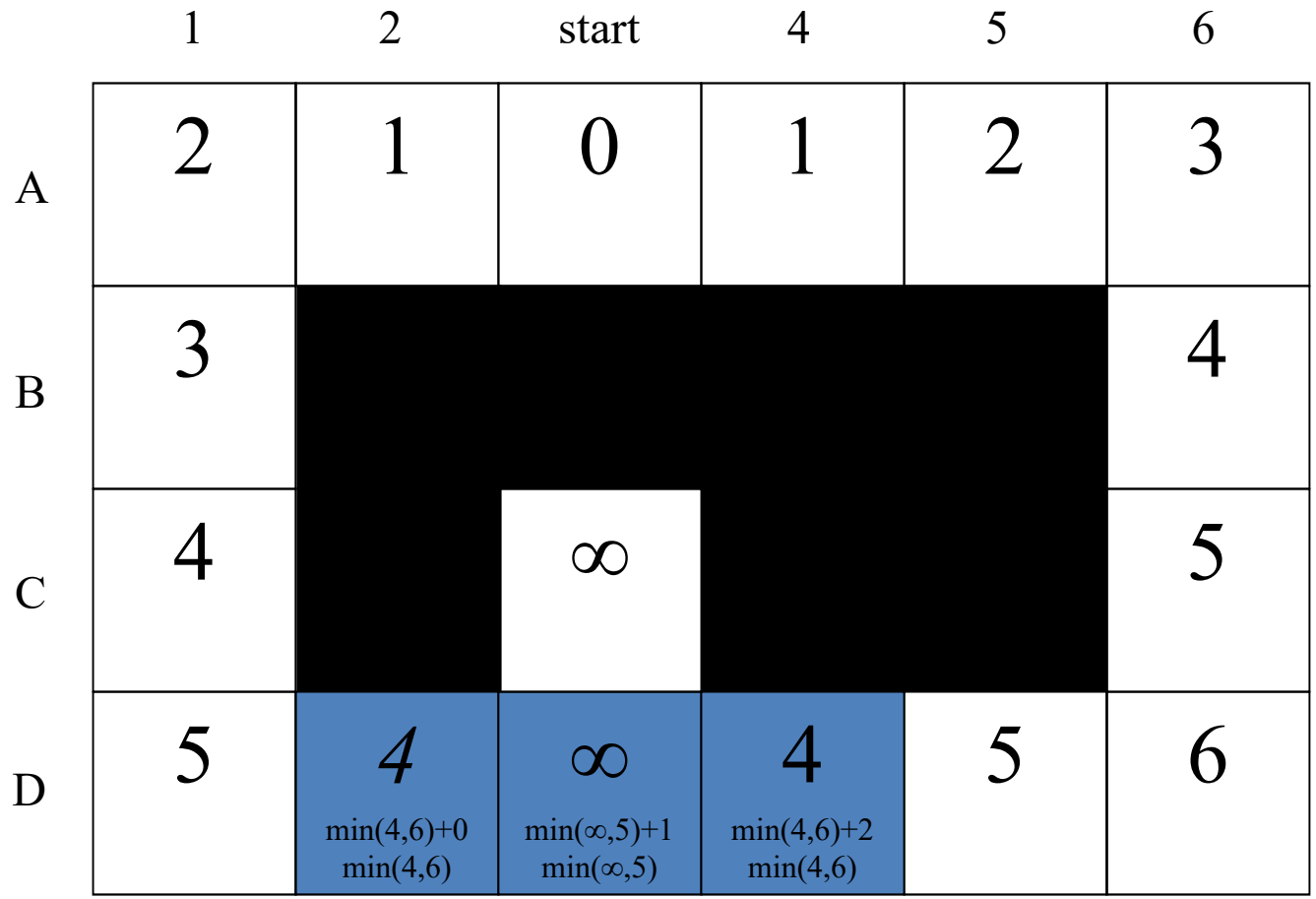
priority queue

D3:[4;3]; C3:[6;4]

4-neighbor grid

Incremental Search with LPA*

g



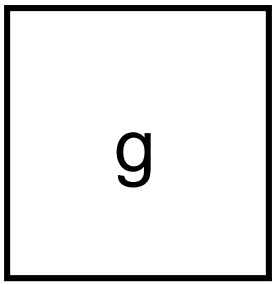
goal

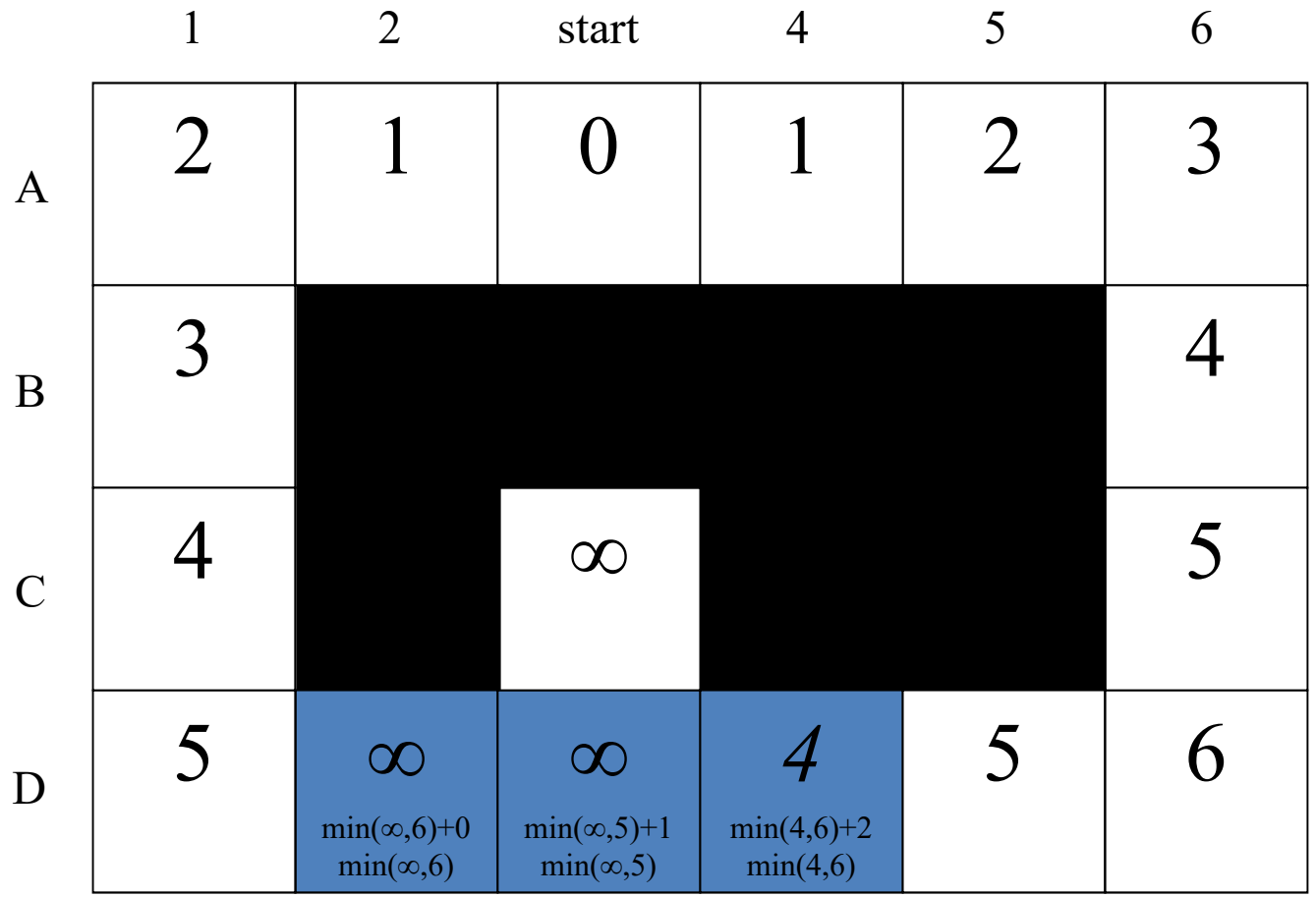
priority queue

D2:[4;4]; D4:[6;4]; D3:[6;5]

4-neighbor grid

Incremental Search with LPA*





goal

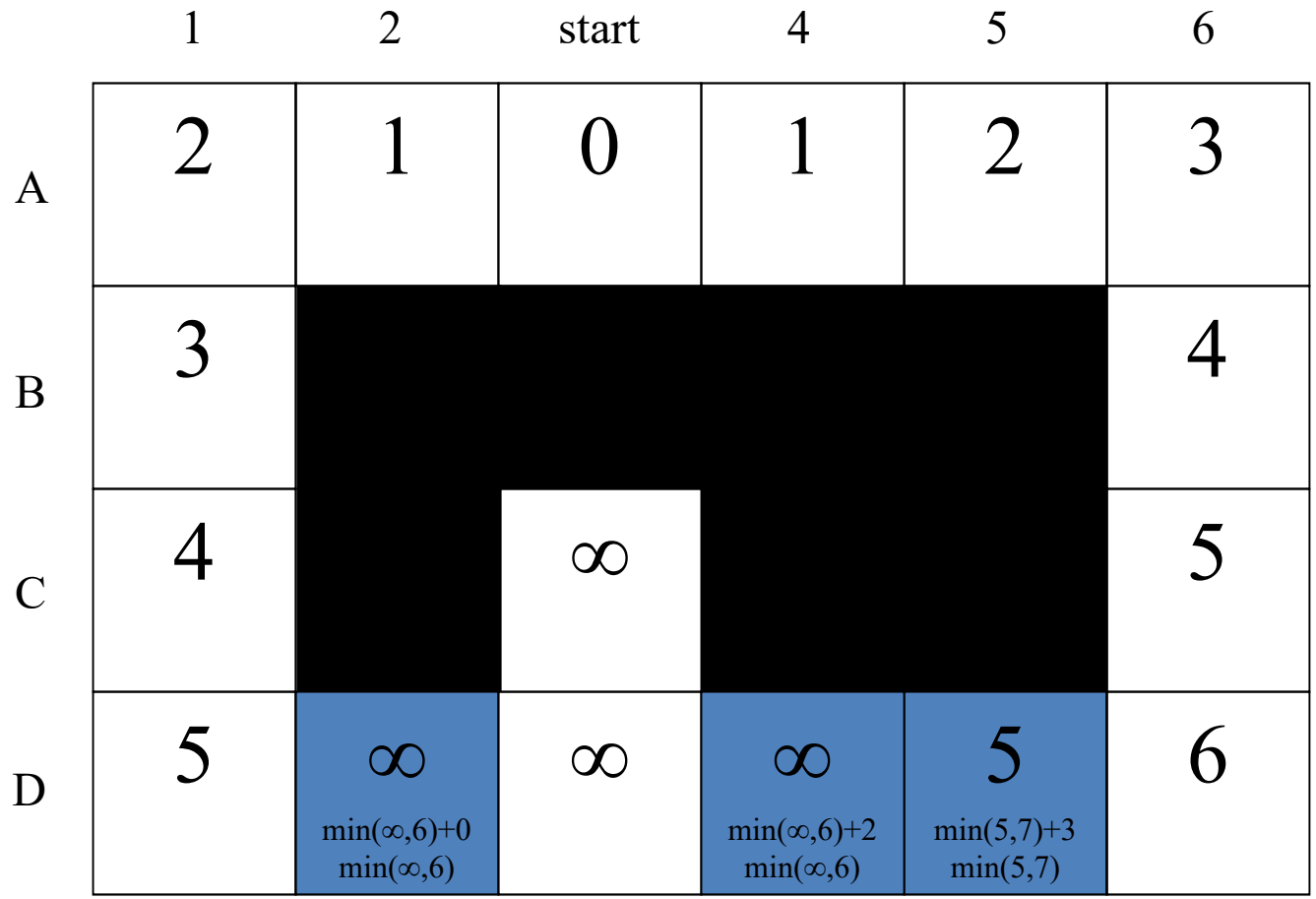
priority queue

D4:[6;4]; D3:[6;5]; D2:[6;6]

4-neighbor grid

Incremental Search with LPA*

g



goal

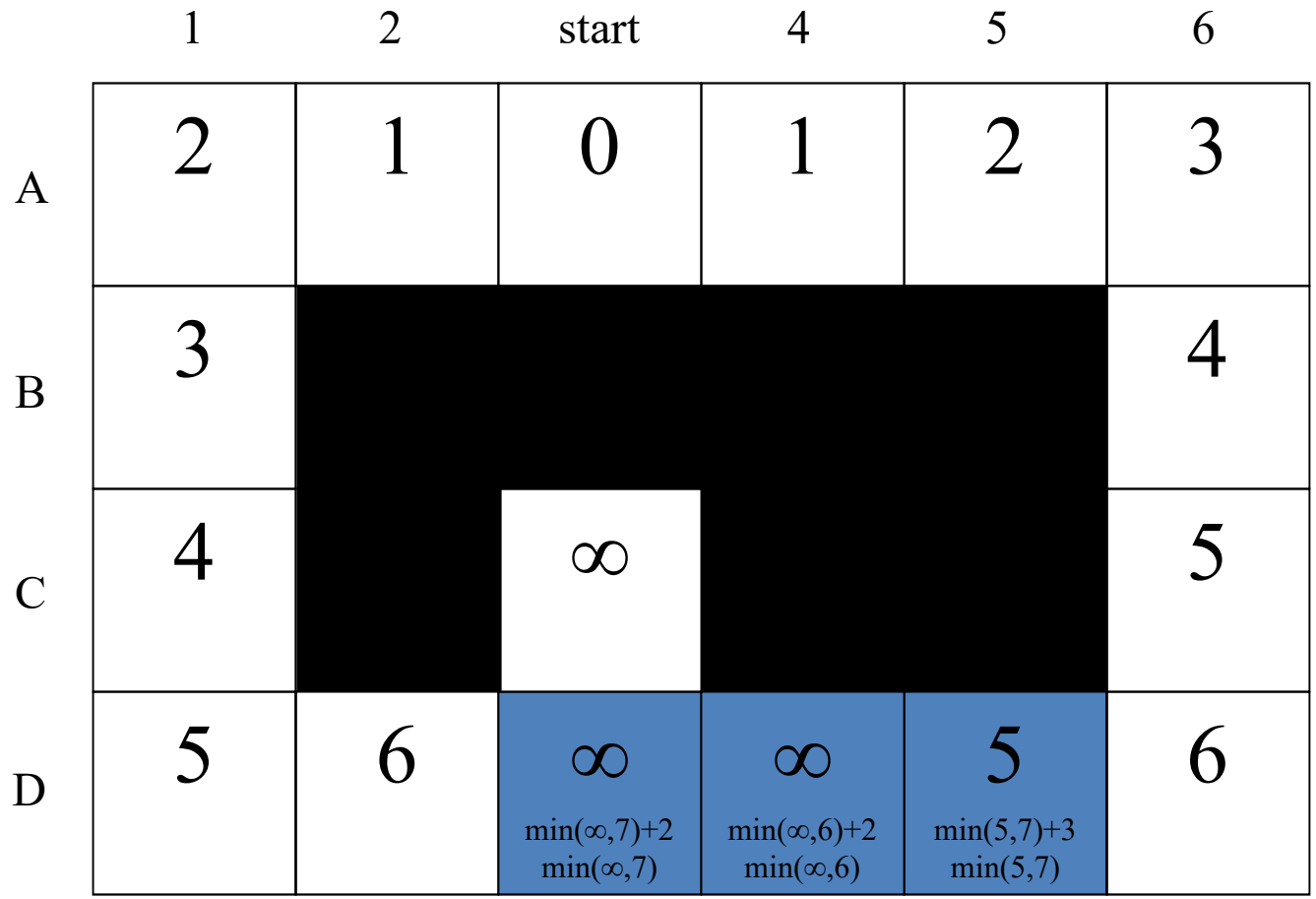
priority queue

D2:[6;6]; D5:[8;5]; D4:[8;6]

4-neighbor grid

Incremental Search with LPA*

g



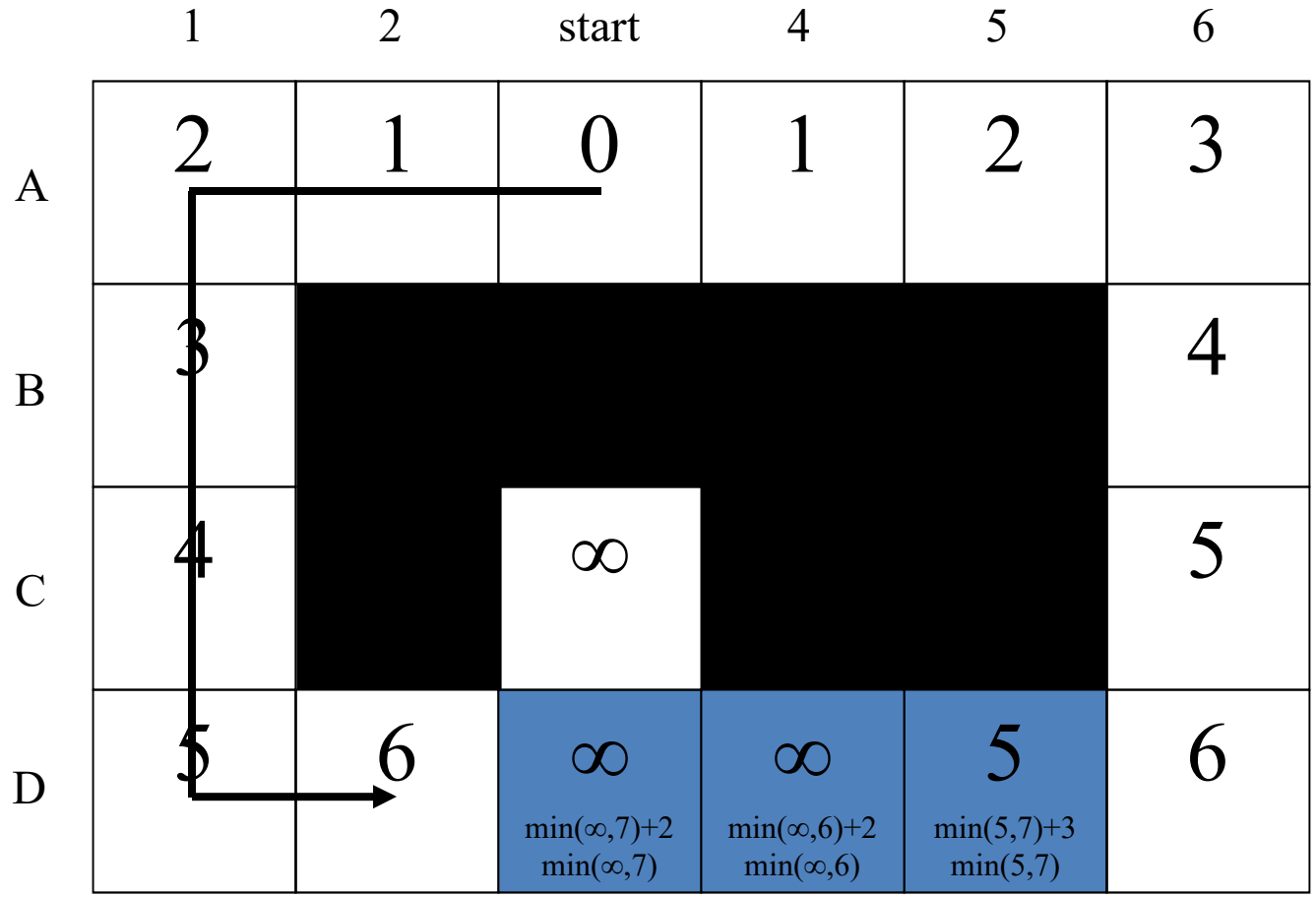
goal

priority queue

4-neighbor grid

D5:[8;5]; D4:[8;6]; D3:[8;7]

Incremental Search with LPA*



4-neighbor grid

Incremental Search with LPA*

```
procedure CalculateKey(s)
return [min(g(s), rhs(s)) + h(s), min(g(s), rhs(s))];
procedure Initialize()
U := Ø;
for all s ∈ S rhs(s) = g(s) = ∞
rhs(s_start) = 0;
U.Insert(s_start, [h(s_start);0];
procedure UpdateVertex(u)
if (u ≠ s_start) rhs(u) = min_{s' in Pred(u)} (g(s')+c(s',u));
if (u ∈ U) U.Remove(u);
if (g(u) ≠ rhs(u)) U.Insert(u, CalculateKey(u));
procedure ComputeShortestPath()
while (U.TopKey < CalculateKey(s_goal) OR rhs(s_goal) ≠ g(s_goal))
    u = U.Pop();
    if (g(u) > rhs(u))
        g(u) = rhs(u);
        for all s ∈ Succ(u) UpdateVertex(s);
    else
        g(u) = rhs(u);
        for all s ∈ { Succ(u) ∪ u } UpdateVertex(s);
procedure Main()
Initialize();
forever
    ComputeShortestPath();
    Wait for changes in edge costs;
    for all directed edges (u, v) with changed edge costs
        Update the edge cost c(u,v);
        UpdateVertex(v);
```

Incremental Search with LPA*

- Theorem

  Each search expands every state at most twice and thus terminates.

  = LPA* terminates

- Theorem

  After a search terminates, one can trace back a shortest path from the start to the goal by always moving from the current state s, starting at the goal, to any predecessor s' that minimizes g(s') + c(s',s) until the start is reached.

  = LPA* is correct

Incremental Search with LPA*

- Theorem

  No search expands a state whose g-value before the search was already equal to its start distance.

  = LPA* is efficient because it uses incremental search

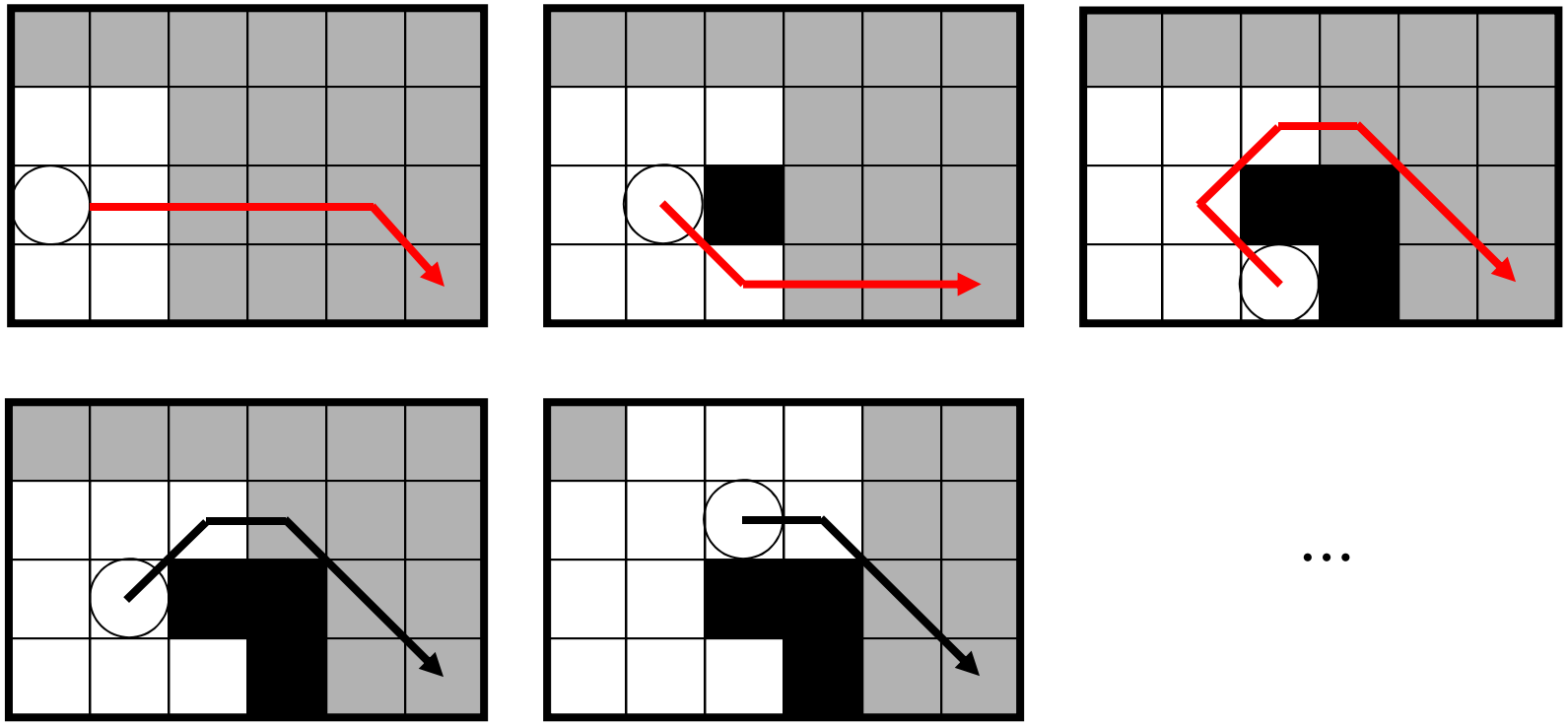
- Theorem

  Each search expands at most those states s with [f(s);g*(s)] ≤ [f(goal); g*(goal)] or [g_{old}(s) + h(s); g_{old}(s)] ≤ [f(goal); g*(goal)], where f(s) = g*(s) + h(s) and g_{old}(s) is the g-value of s before the search.

  = LPA* is efficient because it uses heuristic search

Incremental Search with D* Lite

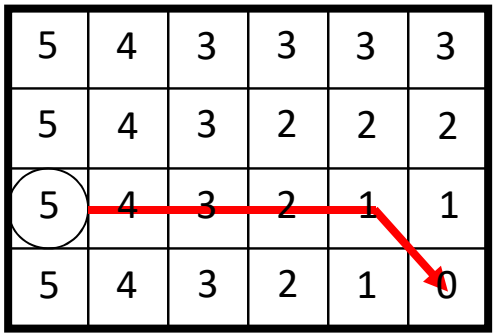
- Goal-directed navigation
  with the freespace assumption

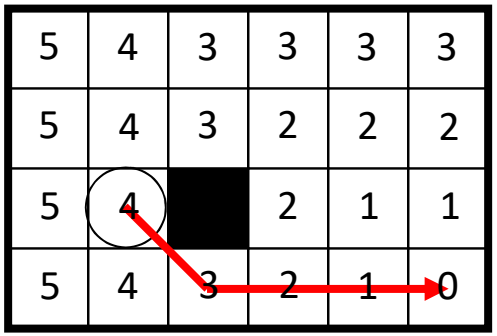


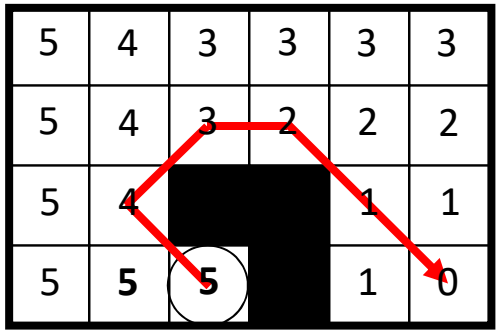
Incremental Search with D* Lite

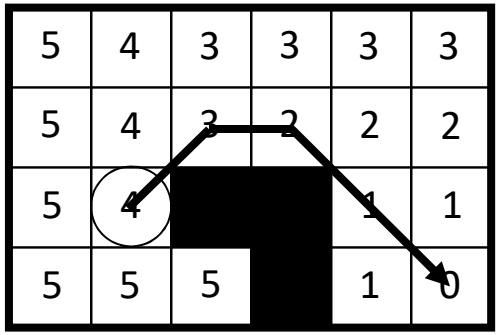
goal distance

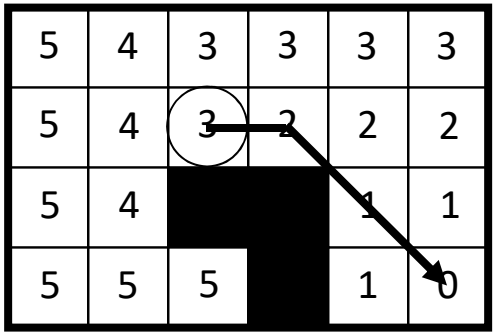
- Goal-directed navigation with the freespace assumption











…

Incremental Search with D* Lite

- D* Lite: Basic Version
- If the agent moves from $s_{oldagent}$ to $s_{newagent}$, then the goal of the search moves from $s_{oldagent}$ to $s_{newagent}$. This changes the priorities of the states in the priority queue

  from $[\min(g(s), rhs(s)) + h(s_{oldagent}, s), \min(g(s), rhs(s))]$

  to $[\min(g(s), rhs(s)) + h(s_{newagent}, s), \min(g(s), rhs(s))]$

  (but not which states are in the priority queue).
- Thus, one needs to reorder the priority queue.

Incremental Search with D* Lite

- D* Lite: Basic Version
- Priority queue: A [8,5]; B [8,6]; C [8,7]
- Agent moves
- Priority queue: C [7,7]; B [8,6]; A [9,5]

Incremental Search with D* Lite

- D* Lite: Final Version
- One uses lower bounds on the new priorities instead of the new priorities themselves

  $[\min(g(s), rhs(s)) + h(s_{oldagent}, s), \min(g(s), rhs(s))]$
  $\leq [\min(g(s), rhs(s)) + h(s_{oldagent}, s_{newagent}) + h(s_{newagent}, s), \min(g(s), rhs(s))]$
  $[\min(g(s), rhs(s)) + h(s_{oldagent}, s) - h(s_{oldagent}, s_{newagent}), \min(g(s), rhs(s))]$
  $\leq [\min(g(s), rhs(s)) + h(s_{newagent}, s), \min(g(s), rhs(s))]$

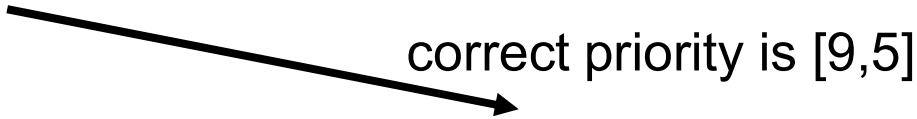
- The term $h(s_{oldagent}, s_{newagent})$ is the same across all states in the priority queue. Instead of deleting it from all states in the priority queue, we add it to all states added to the priority queue in the future [Stentz].

Incremental Search with D* Lite

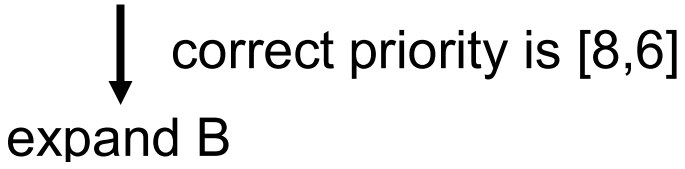
- D* Lite: Final Version
- When one selects a state for expansion, one first checks whether its priority is correct.
- If so, then one expands the state.
- If not (= it is a lower bound), then one re-inserts the state into the priority queue with the correct priority.

Incremental Search with D* Lite

- D* Lite: Final Version
- Priority queue: A [8,5]; B [8,6]; C [8,7]
- Agent moves: $h(s_{oldstart}, s_{newstart}) = 2$ (changes accumulate)
- Priority queue: A [8,5]; B [8,6]; C [8,7]
- Add state D with priority [10,5]
- Priority queue: A [8,5]; B [8,6]; C [8,7]; D [12,5]



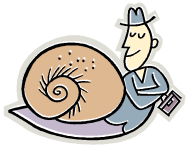
- Priority queue: B [8,6]; C [8,7]; A [9,5]; D[12,5]



Applications

- Used in robotics

JPL



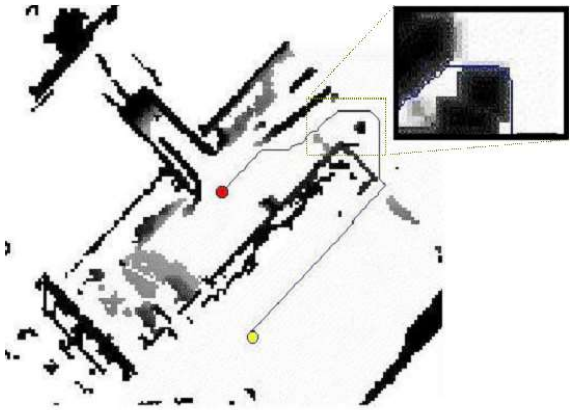
2006/2007

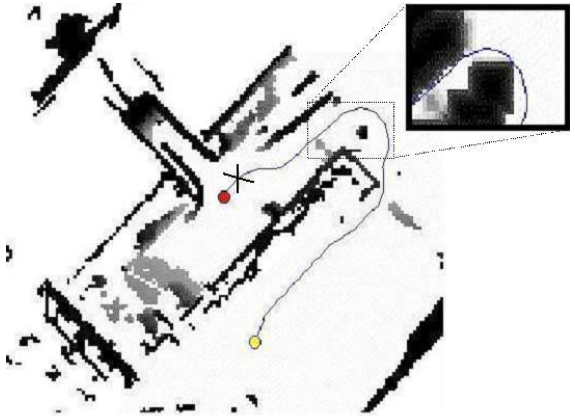
20(!) megahertz RAD6000 processor

[work done by JPL/CMU, not me]

Applications

- Used in robotics





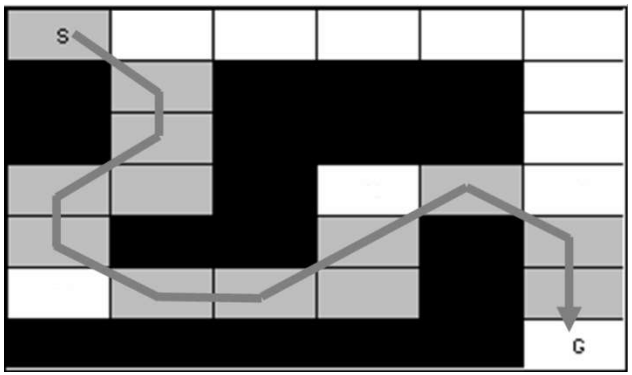
2d (x, y) planning
- 54,000 states
- Fast planning
- Slow execution

4d (x, y, Θ, v) planning
- > 20,000,000 states
- Slow planning
- Fast execution

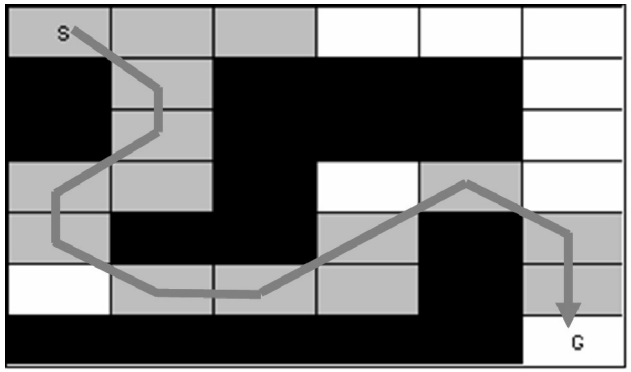
[work done by CMU, not me]

Incremental Search with ARA*

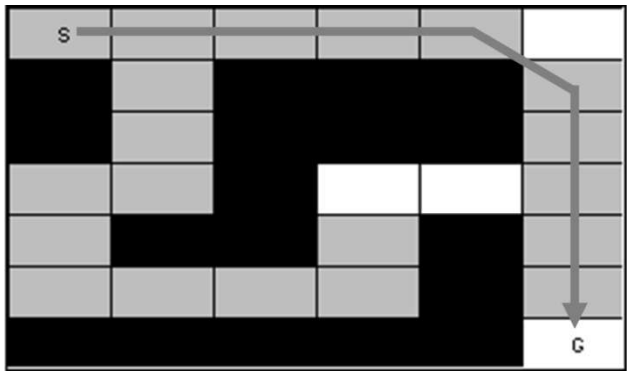
- Anytime search with weighted A* [Likhachev et al.]:
  f(s) = g(s) + w h(s)



w = 2.5
13 expansions
11 movements



w = 1.5
15 expansions
11 movements



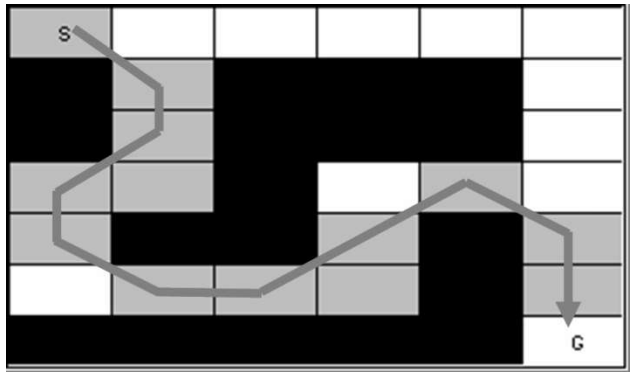
w = 1.0
20 expansions
10 movements

8-neighbor grid

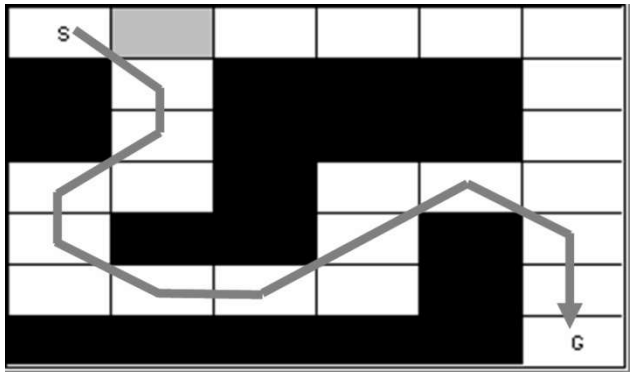
[work done by CMU, not me]

Incremental Search with ARA*

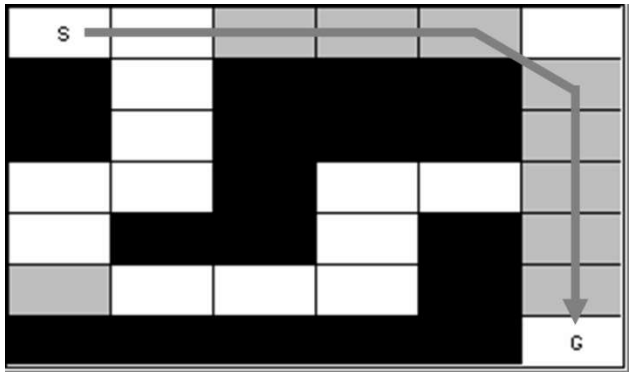
- Anytime search with weighted A* [Likhachev et al.]:
  f(s) = g(s) + w h(s)



w = 2.5
13 expansions
11 movements



w = 1.5
1 expansion
11 movements



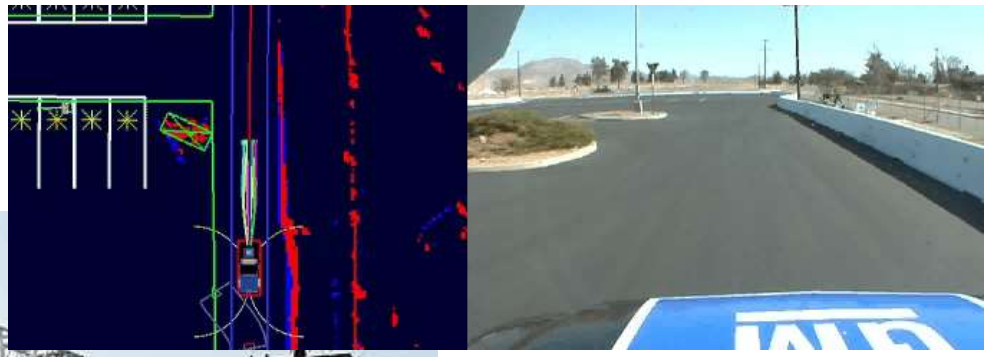
w = 1.0
9 expansions
10 movements

8-neighbor grid

[work done by CMU, not me]

Applications

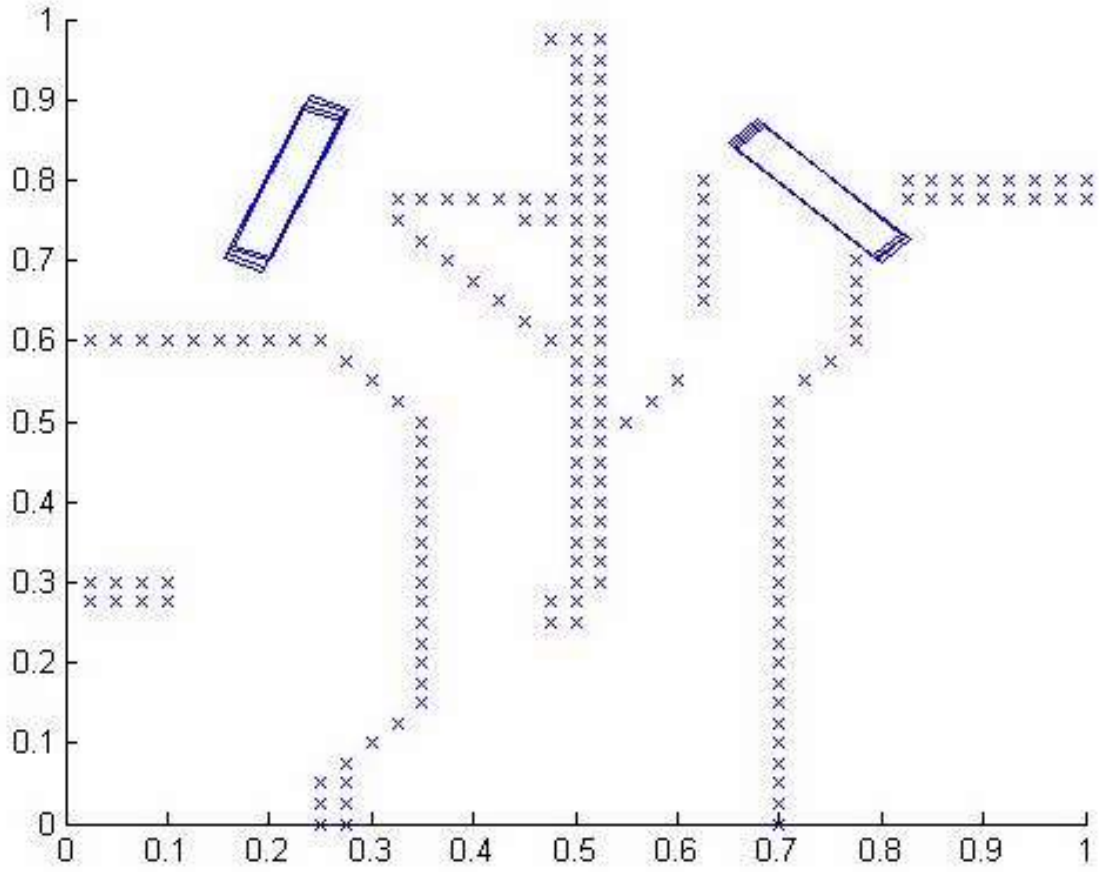
- Used in robotics



[work done by CMU, not me]

Applications

- Moving target search



Applications

- High-level replanning and plan reuse when
  - World changes over time
  - Model of the world changes over time
  - What-if analyses need to be performed

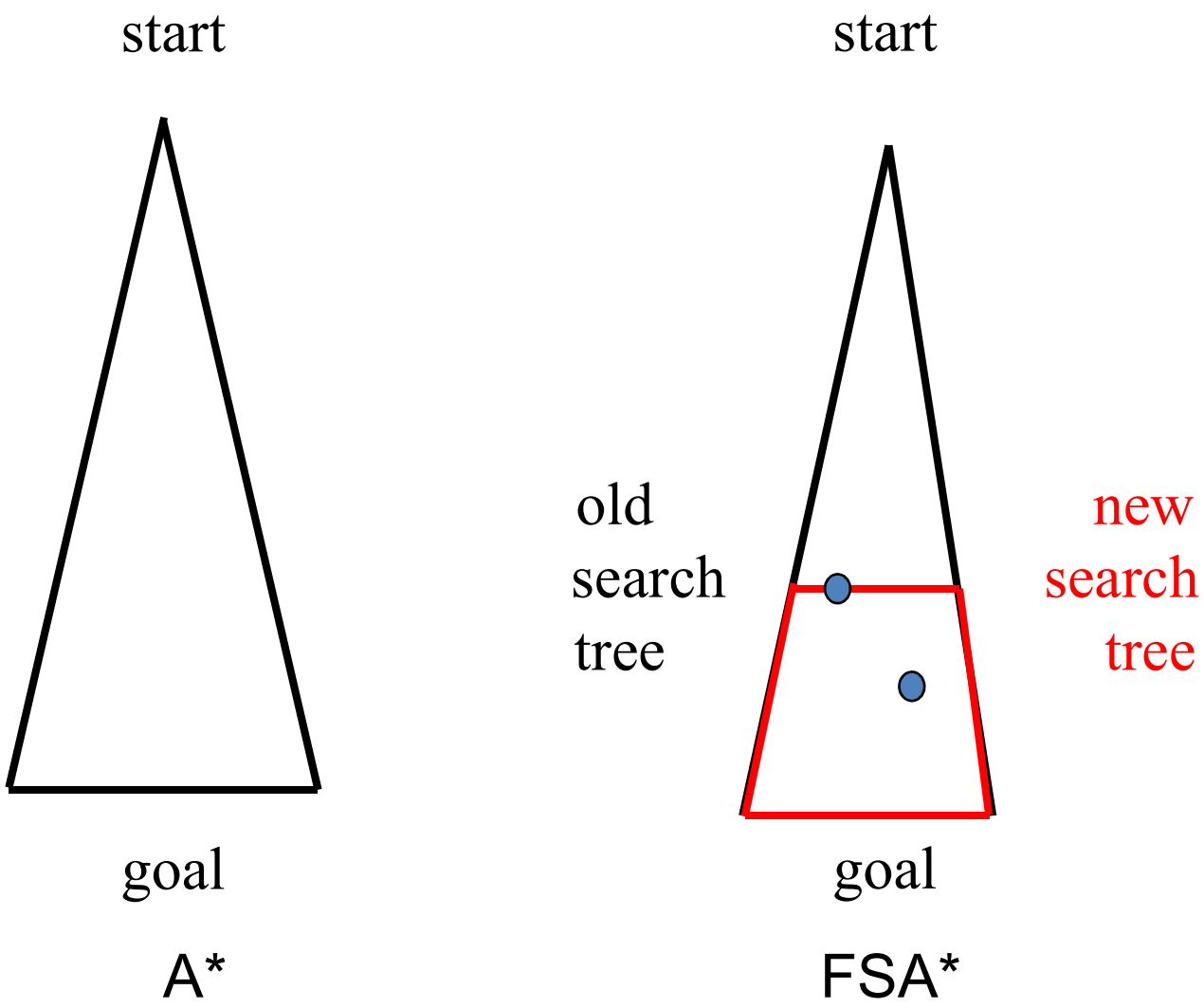
www.rca.org

www.airpartner.com

Alternative: FSA*

- Fringe Saving A* (FSA*) speeds up A* searches for a sequence of similar search problems by starting each search at the point where it could differ from the previous one

Alternative: FSA*



Alternative: FSA*

- Seventh and last iteration of A*

order of expansions

			4	5	6
		3	1		(7)
			2		

			2	3	4
		2	1	2	3
	2	1	0		4
		2	1		

7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0
6	5	4	3	2	1

			6	6	6
		6	4	4	4
	6	4	2		(4)
		6	4		

g-values + h-values = f-values

cost of the shortest path in the search tree from the start state to the given state

generated but not expanded state (OPEN list)

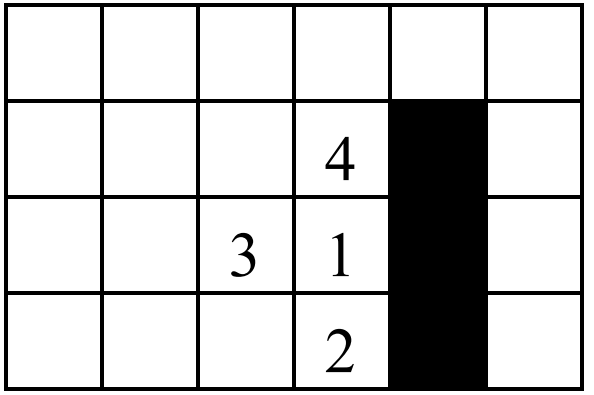
expanded state (CLOSED list)

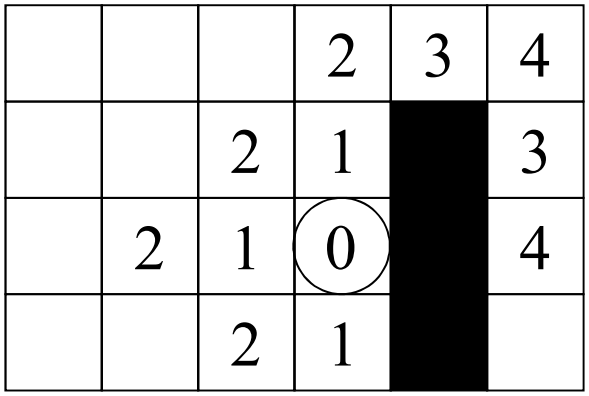
4-neighbor grid

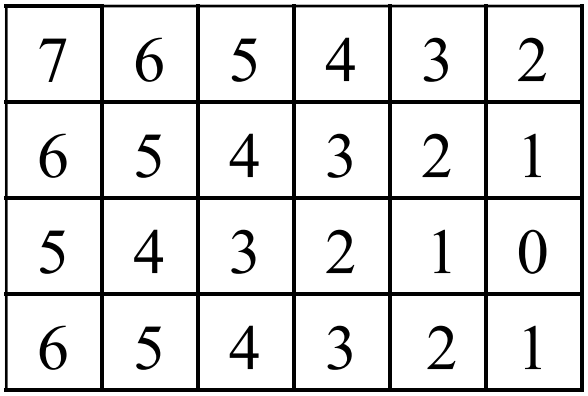
Alternative: FSA*

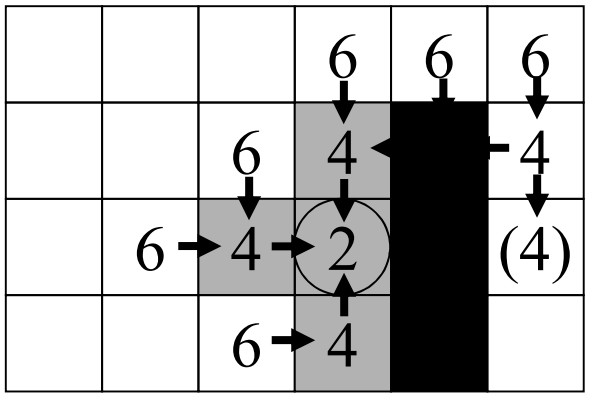
- One state becomes blocked

order of expansions









g-values + h-values = f-values

cost of the shortest path found so far from the start state to the given state

generated but not expanded state (OPEN list)

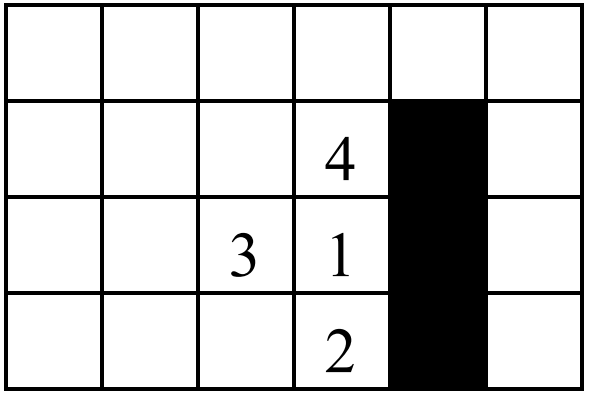
expanded state (CLOSED list)

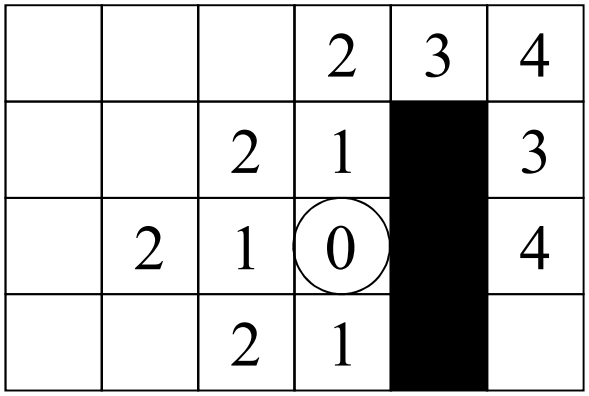
4-neighbor grid

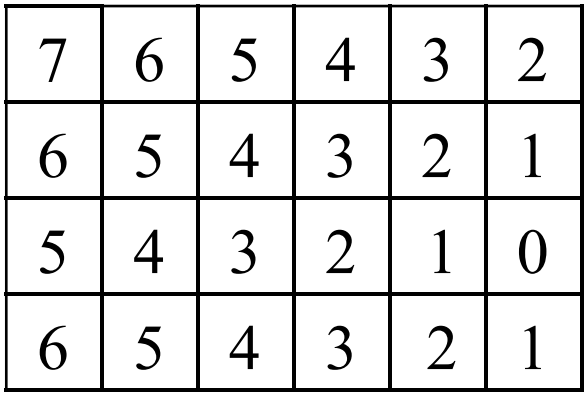
Alternative: FSA*

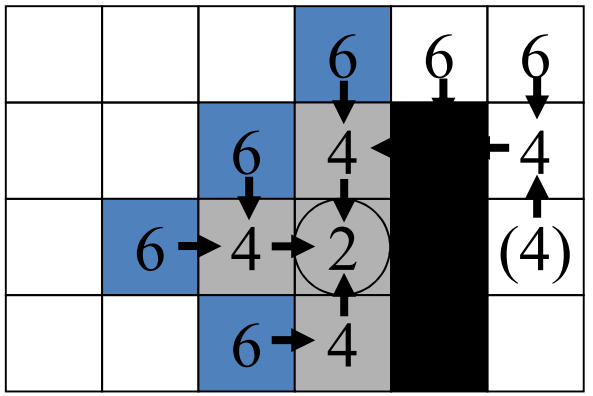
- One state becomes blocked

order of expansions









g-values + h-values = f-values

cost of the shortest path found so far from the start state to the given state

generated but not expanded state (OPEN list)

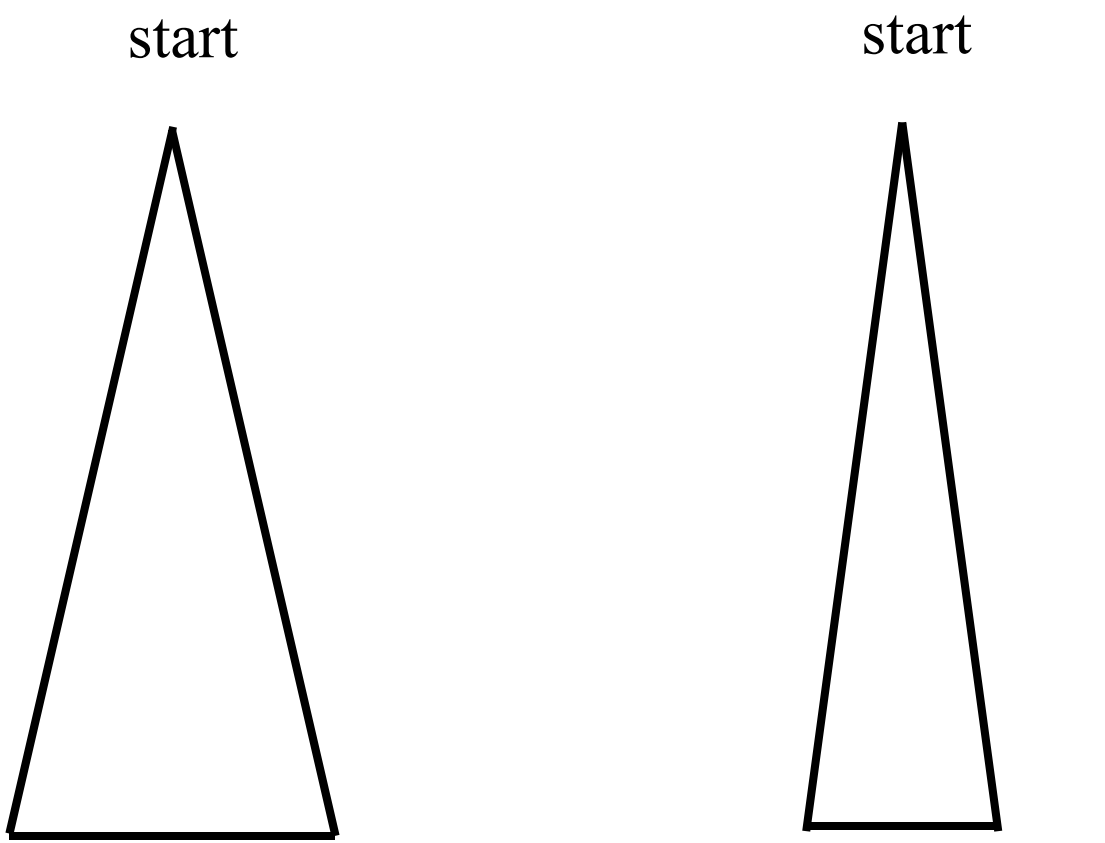
expanded state (CLOSED list)

4-neighbor grid

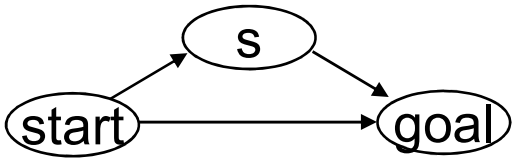
Alternative: Adaptive A*

- Adaptive A* speeds up A* searches for a sequence of similar search problems by making the h-values more informed after each search
- The principle behind Adaptive A* was earlier used in Hierarchical A* [Holte et al.]

Alternative: Adaptive A*



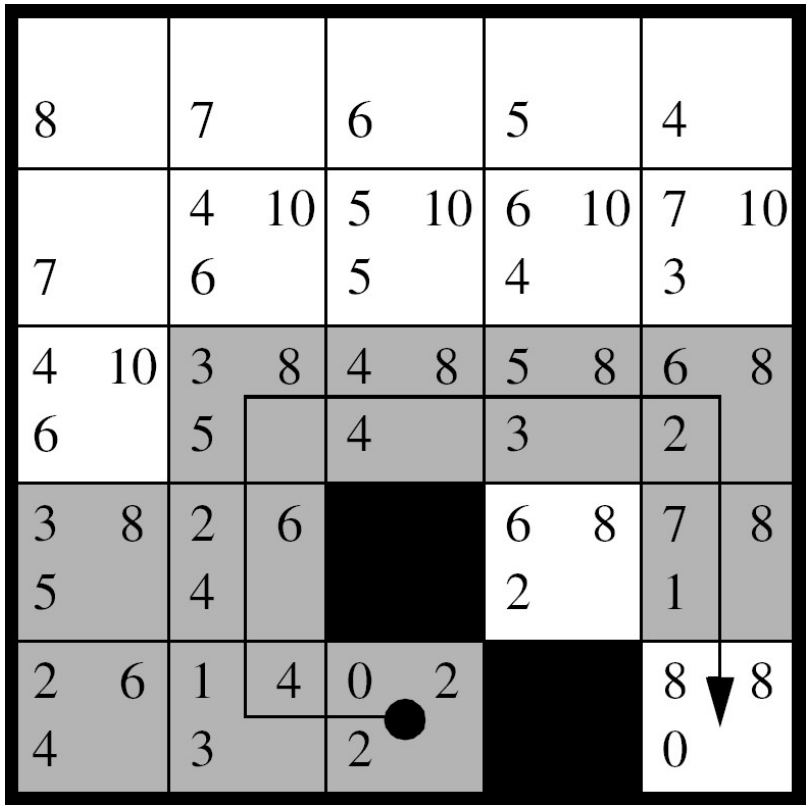
Alternative: Adaptive A*



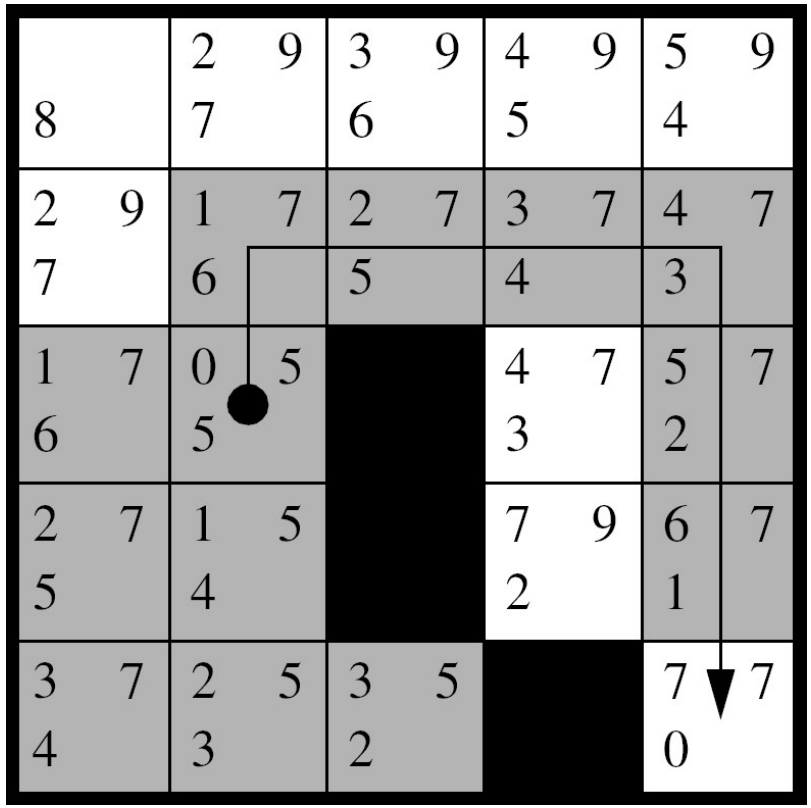
- Consider a state s that was expanded by A* with consistent h-values h_{old}:
  - distance(start,s) + distance(s,goal) ≥ distance(start,goal)
  - distance(s,goal) ≥ distance(start,goal) – distance(start,s)
  - distance(s,goal) ≥ $f(goal) - g(s) = h_{new}(s)$
- The h-values h_{new} are again consistent.
- The h-values h_{new} dominate the h-values h_{old}.
- These properties continue to hold even if the start state changes or the edge costs increase.
- The next A* search with h-values h_{new} expands no more states than an A* search with h-values h_{old} and likely many fewer states.

Alternative: Adaptive A*

g	f
h	



first A* search



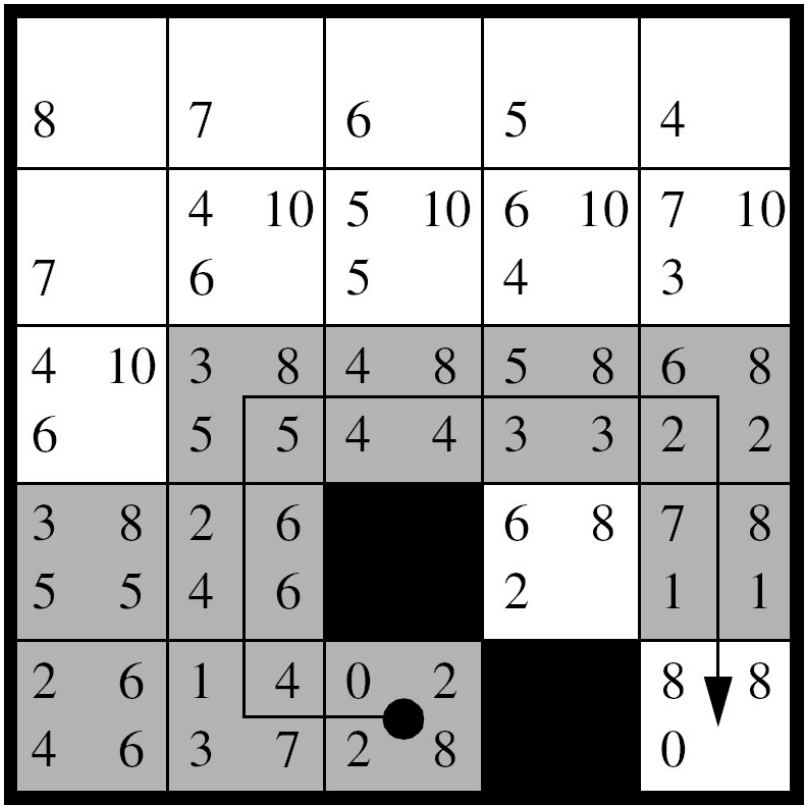
second A* search

13 expansions

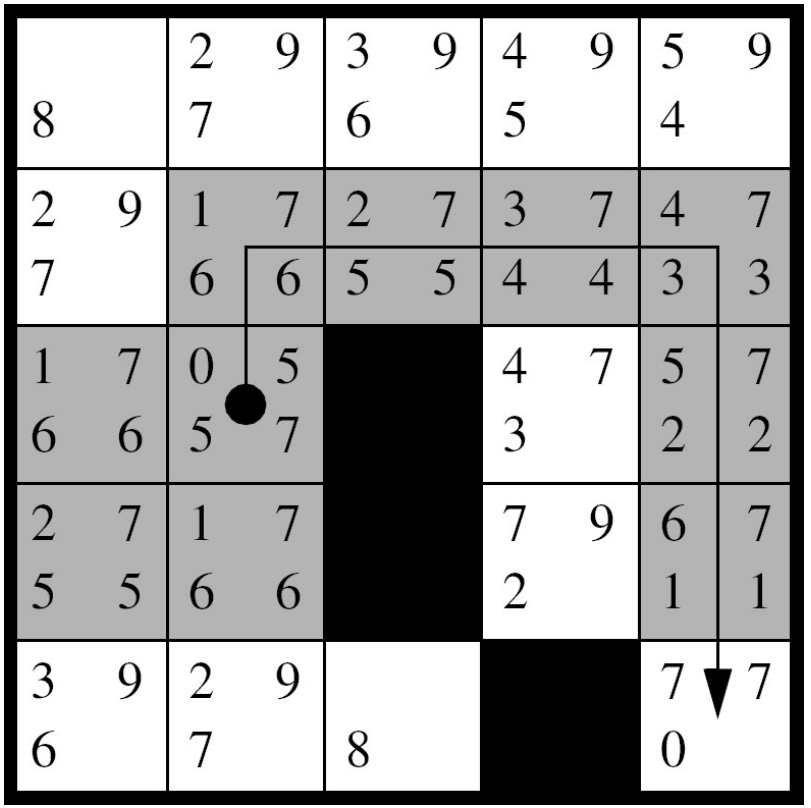
4-neighbor grid

Alternative: Adaptive A*

g	f
h_{old}	h_{new}



first Adaptive A* search



second Adaptive A* search

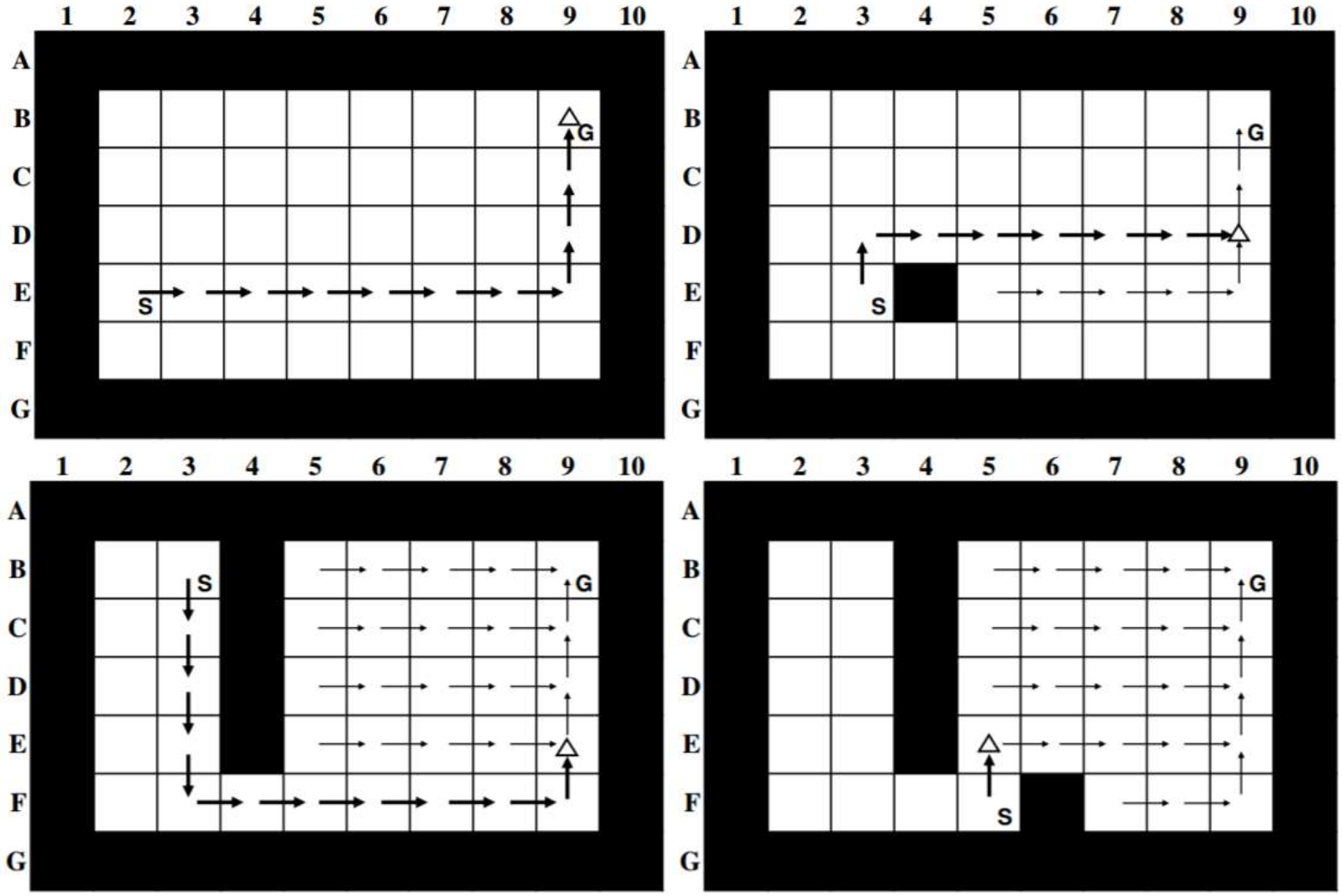
10 expansions

4-neighbor grid

Alternative: Adaptive A*

LPA* and D* Lite	Adaptive A*
• Adapt previous search tree	• Improve previous h-values
• Start node must remain unchanged • Edge cost in/decreases	• Goal node must remain unchanged • Edge cost increases only*
• Can result in more node expansions than A* • Fewer node expansions on average • Slow node expansions	• Guaranteed no more node expansions than A* • More node expansions on average • Fast node expansions

Alternative: Tree Adaptive A*



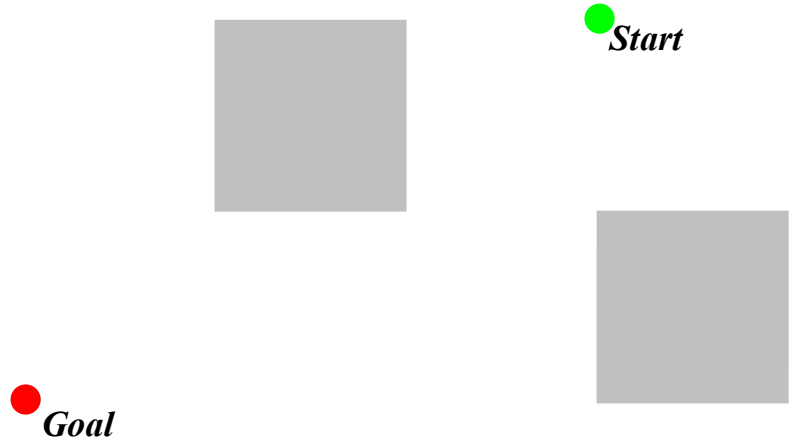
4-neighbor grid

TOC

- Single-robot path planning
  - Runtime matters
    - Hierarchical search (via preprocessing)
    - Incremental search (via using experience with previous similar searches)
  - Quality-runtime tradeoff matters
    - Any-angle search
- Multi-robot path planning and execution

Introduction

[from JPL]



A* on Visibility Graphs

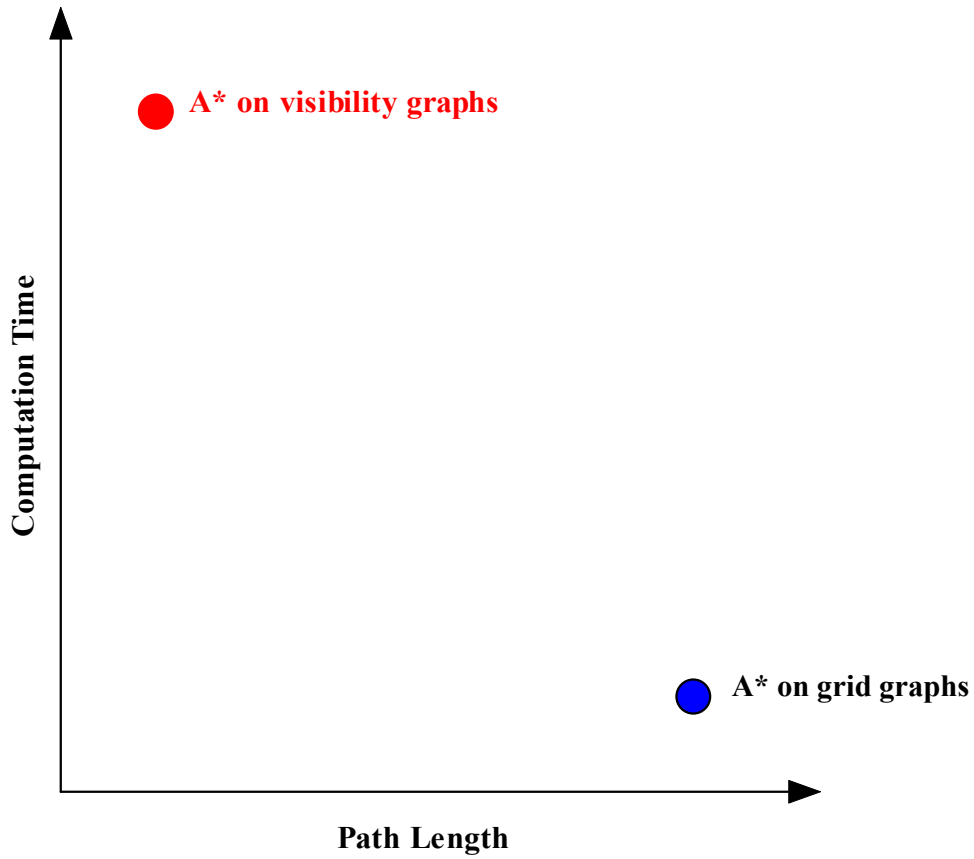
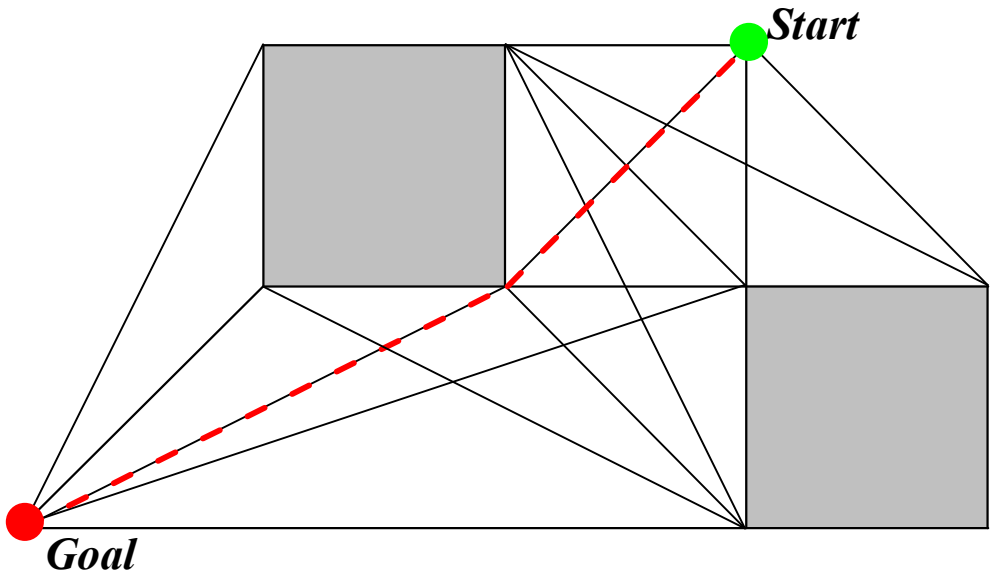


figure is notional

A* on Visibility Graphs

- A* on Visibility Graphs [Lozano-Perez et al.]
- Note: Sophisticated versions exist, e.g. [Shah and Gupta]



- Shortest path in 2D terrain
- Slow due to many edges and line-of-sight checks

A* on Grid Graphs

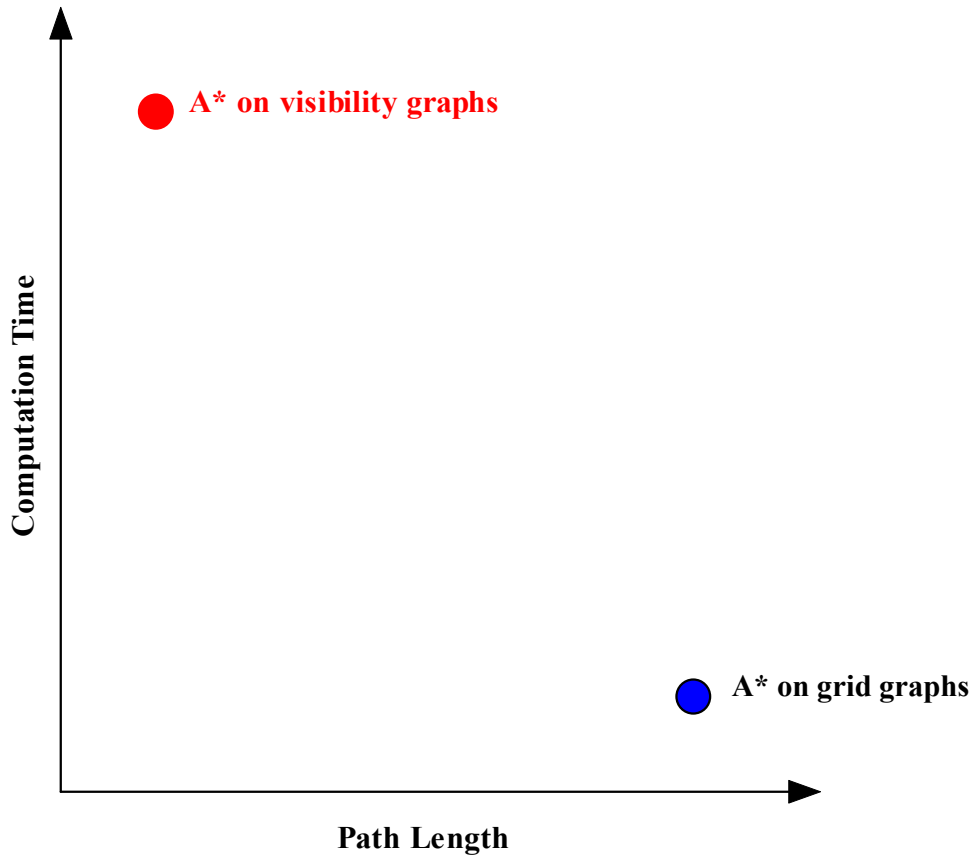
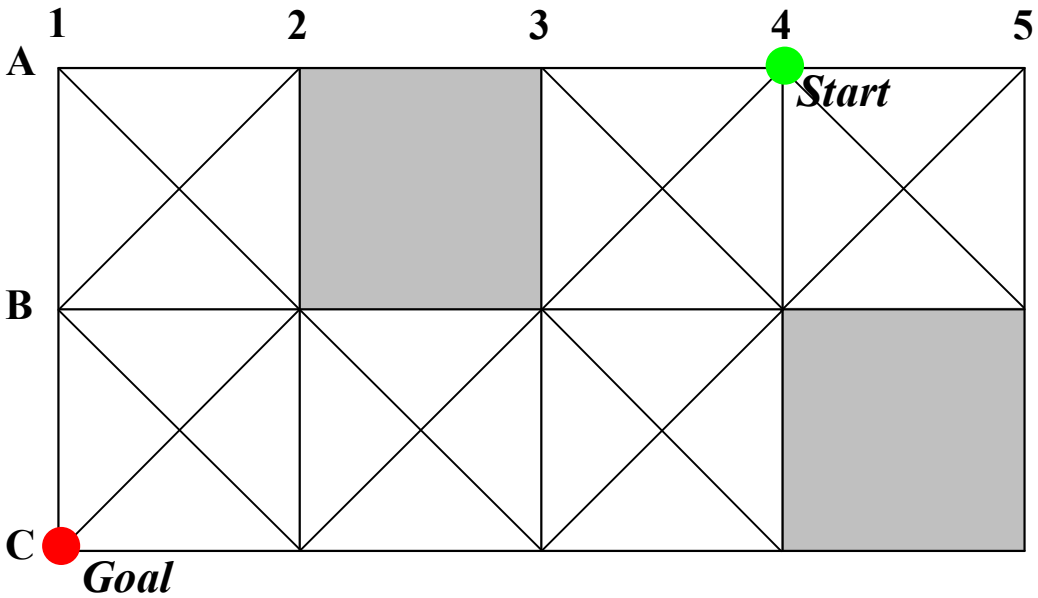


figure is notional

A* on Grid Graphs

- A* on grid graphs

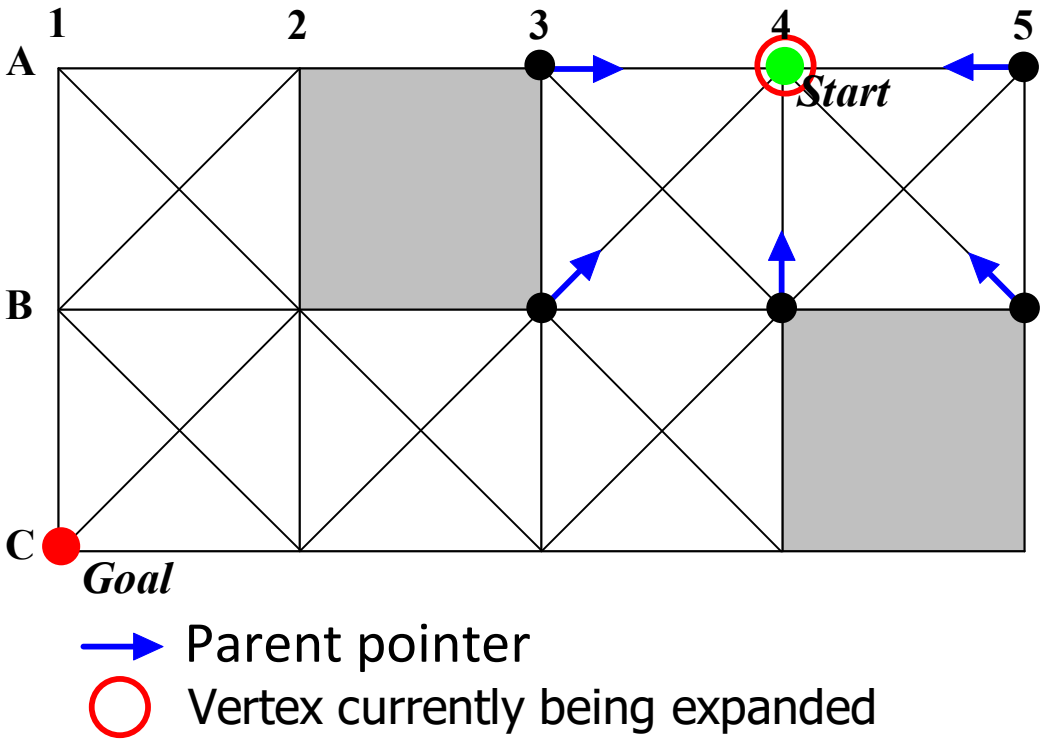


- A* assigns two values to every vertex s
  - g(s): the length of the shortest path from the start vertex to s found so far
  - parent(s): the parent pointer used to extract the path after termination
  - Following the parents from s to the start vertex results in a path of length g(s)

8-neighbor grid

A* on Grid Graphs

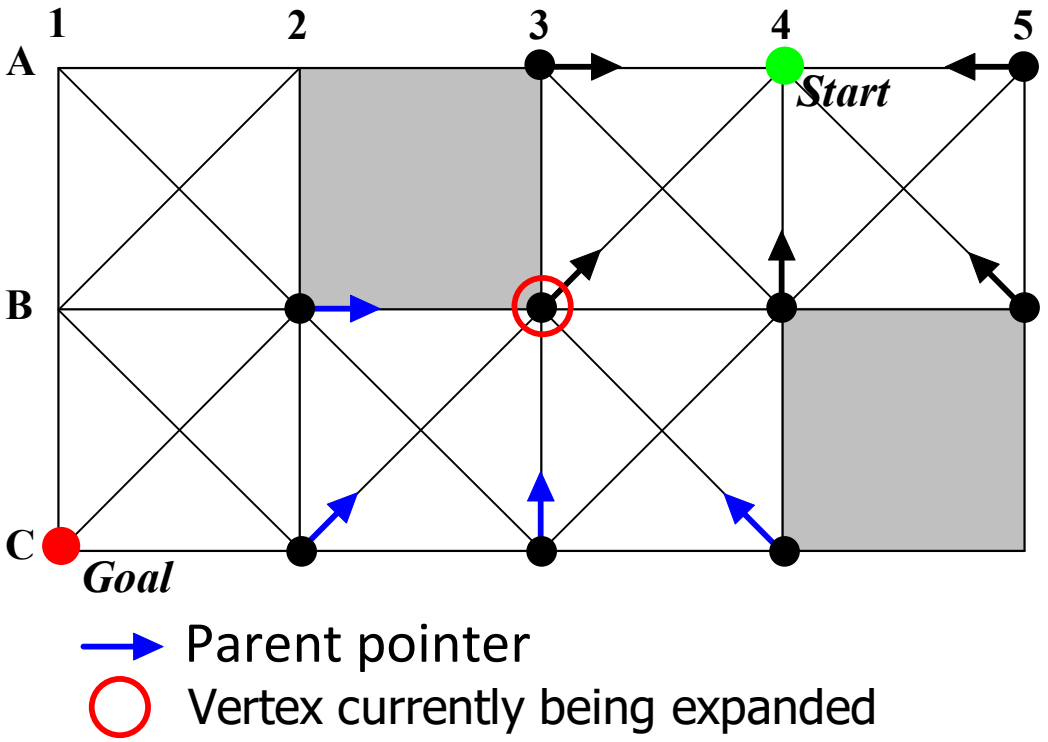
- A* on grid graphs



8-neighbor grid

A* on Grid Graphs

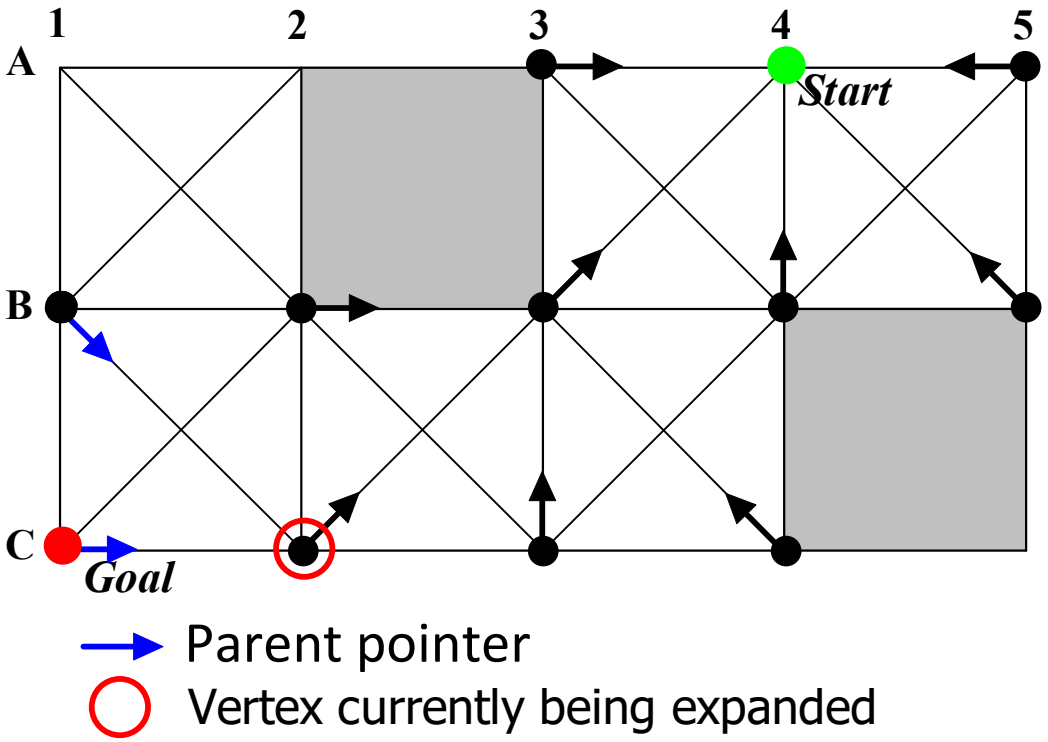
- A* on grid graphs



8-neighbor grid

A* on Grid Graphs

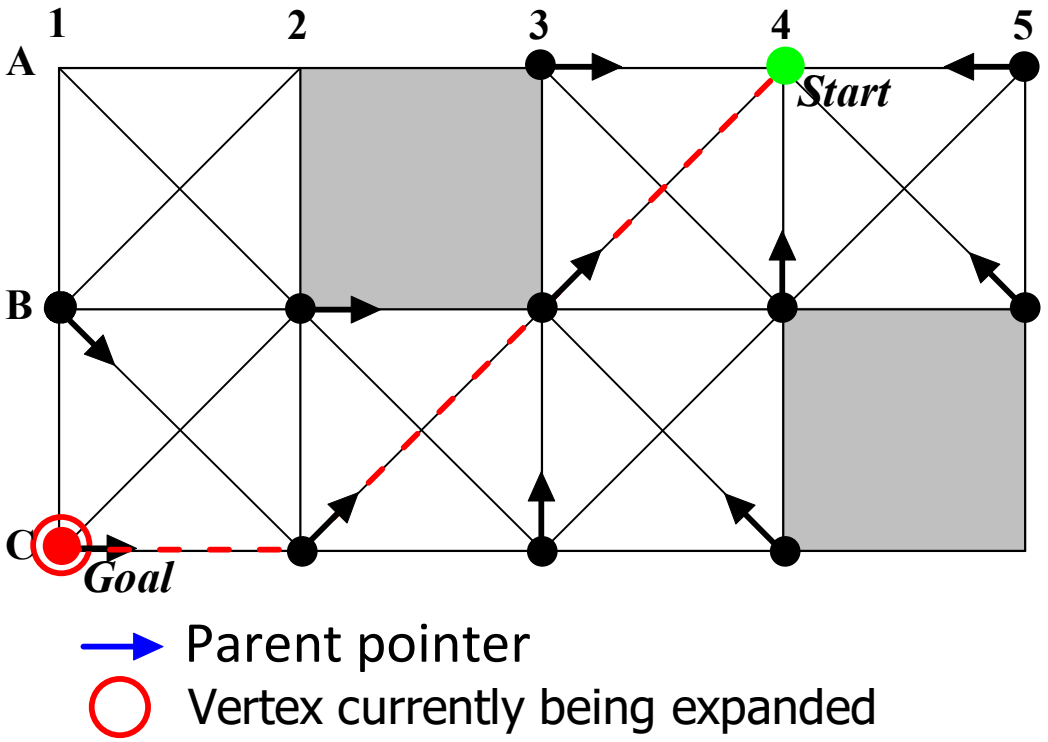
- A* on grid graphs



8-neighbor grid

A* on Grid Graphs

- A* on grid graphs



8-neighbor grid

from now on, we will show only the cells

A* on Grid Graphs

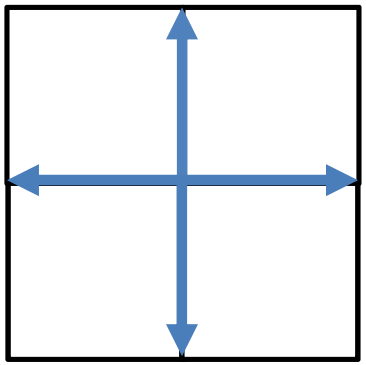
Dimension	Regular Grid	Neighbors	% Longer Than Shortest Path
2D	triangular grid corners	3-neighbor	≈ 100
		6-neighbor	≈ 15
	square grid corners	4-neighbor	≈ 41
		8-neighbor	≈ 8
	hexagonal grid centers	6-neighbor	at least ≈ 15
		12-neighbor	at least ≈ 4
3D	cubic grid corners	6-neighbor	at least ≈ 73
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

A* on Grid Graphs

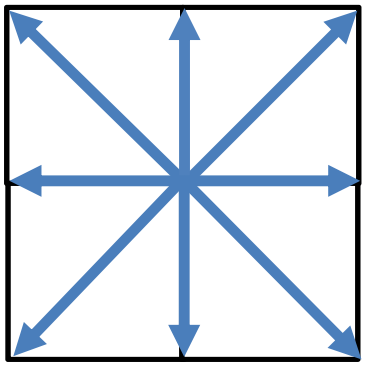
figure is notional

Grids with Higher Degree Vertices

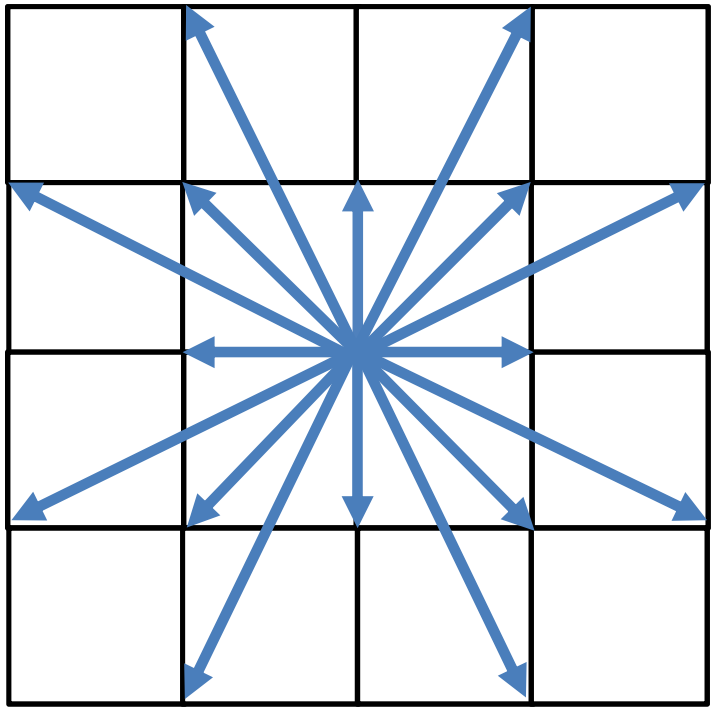
- Grid path finding on the 2^k neighborhoods [Rivera et al.]



4 neighborhood



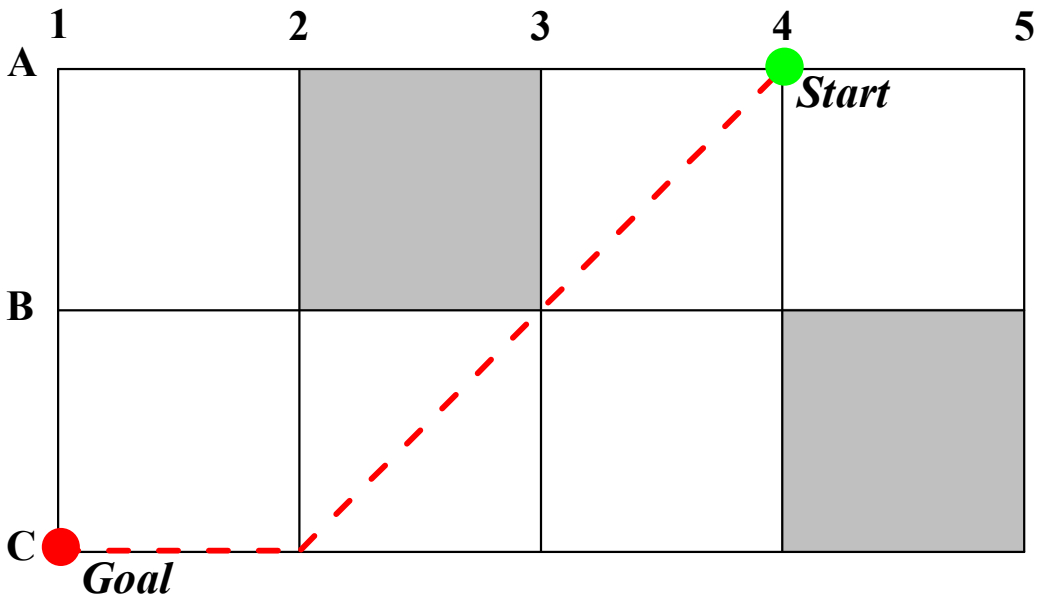
8 neighborhood



16 neighborhood

A* with Post Smoothing

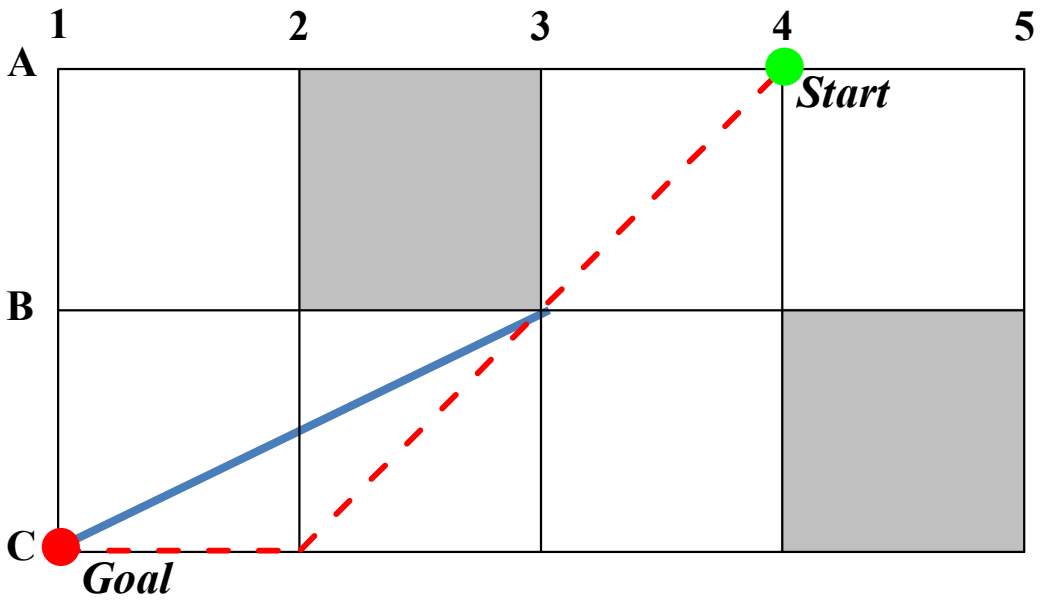
- A* with Post Smoothing [Thorpe; Botea et al.; Millington]



8-neighbor grid

A* with Post Smoothing

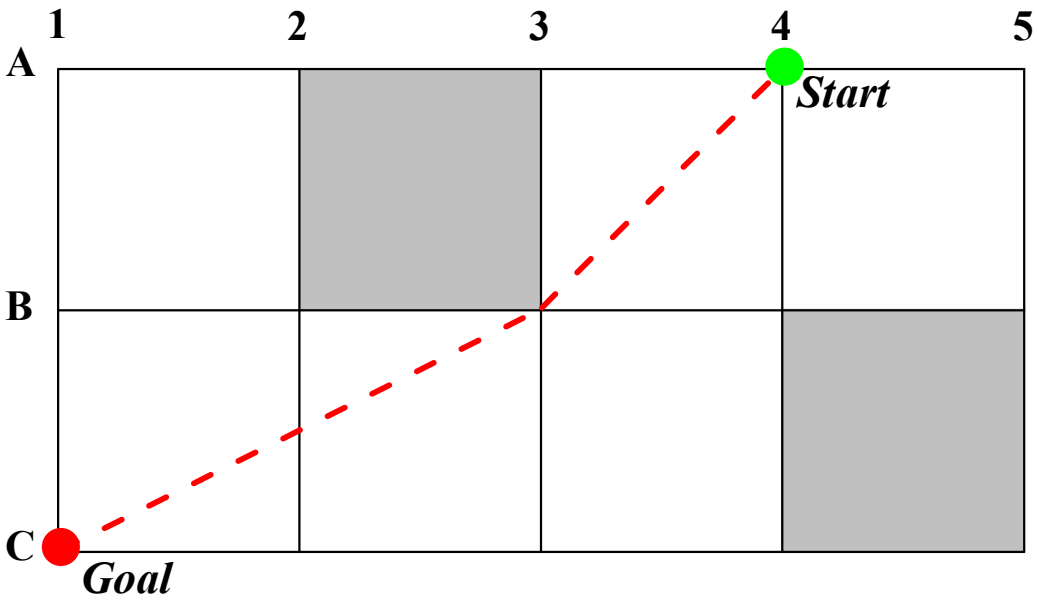
- A* with Post Smoothing



8-neighbor grid

A* with Post Smoothing

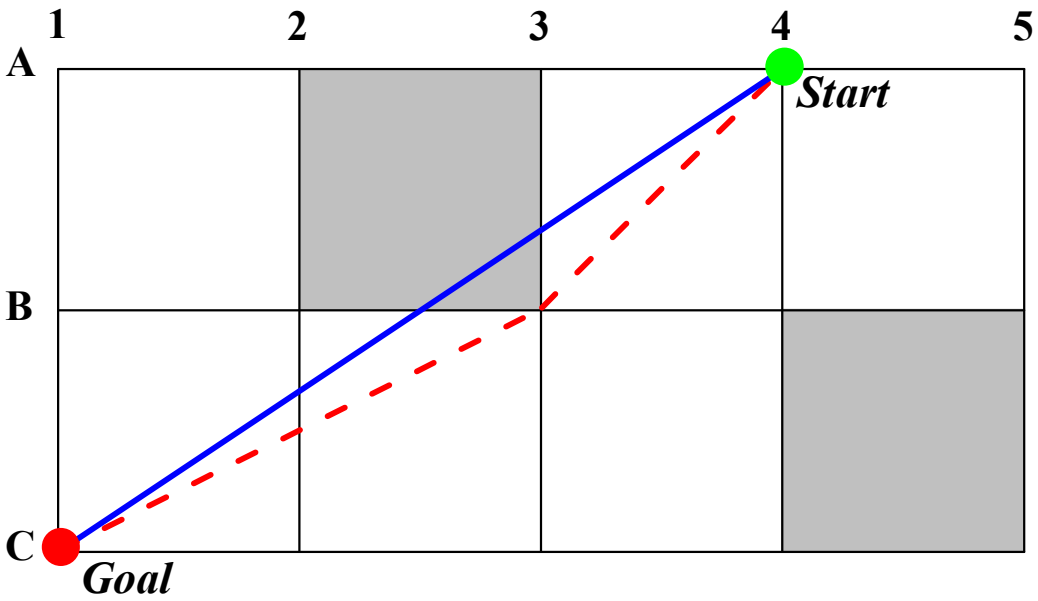
- A* with Post Smoothing



8-neighbor grid

A* with Post Smoothing

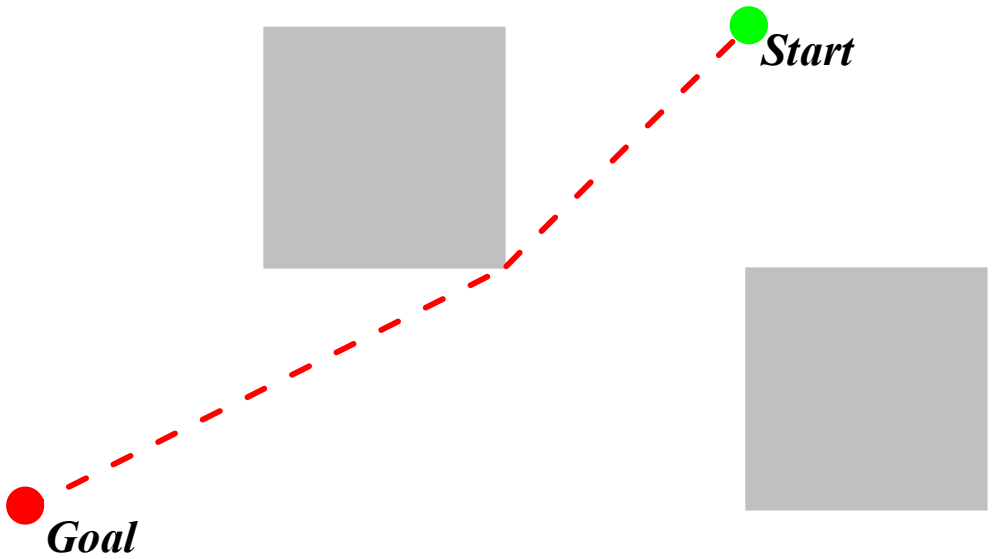
- A* with Post Smoothing



8-neighbor grid

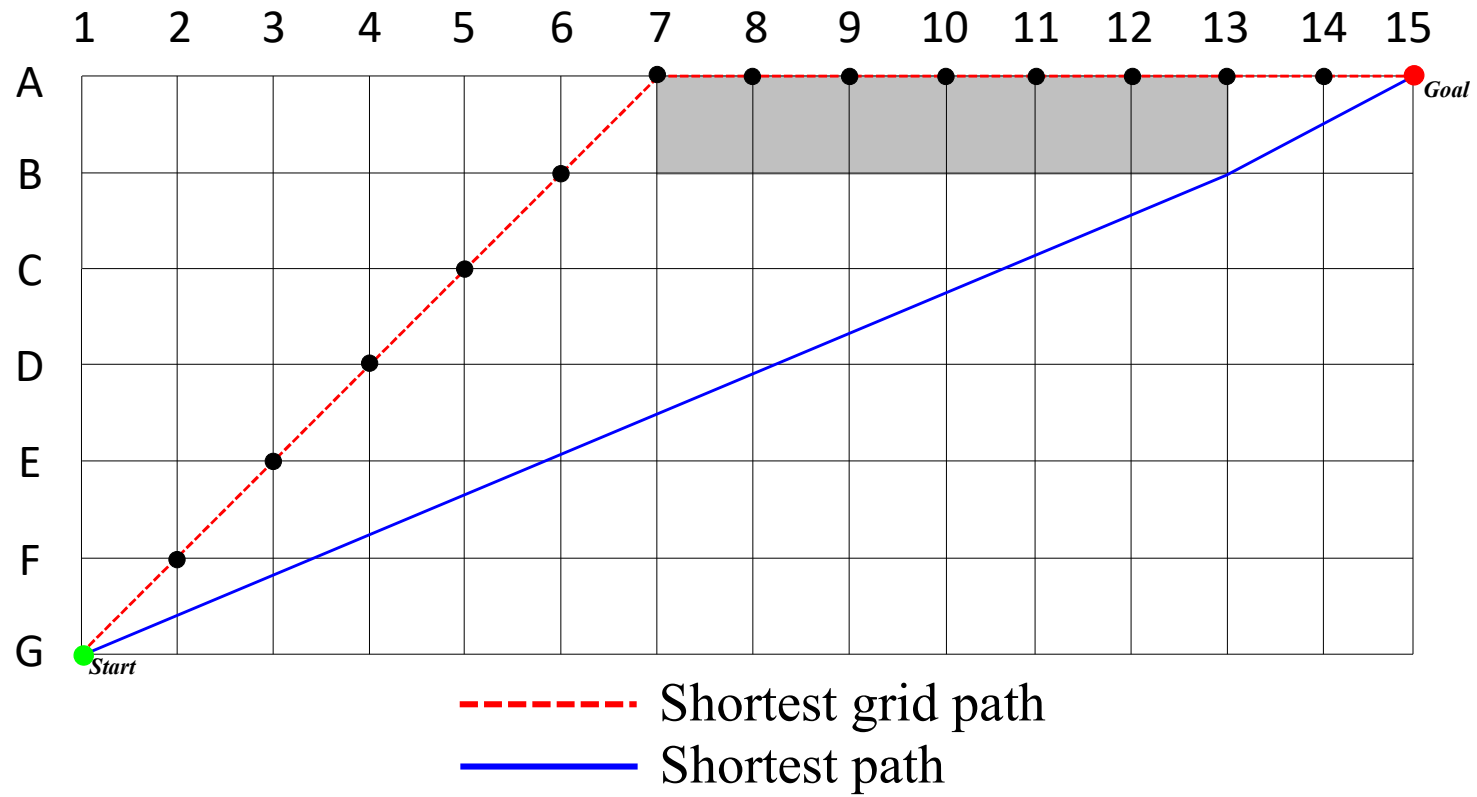
A* with Post Smoothing

- A* with Post Smoothing



A* with Post Smoothing

- A* with Post Smoothing



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

Any-Angle Search

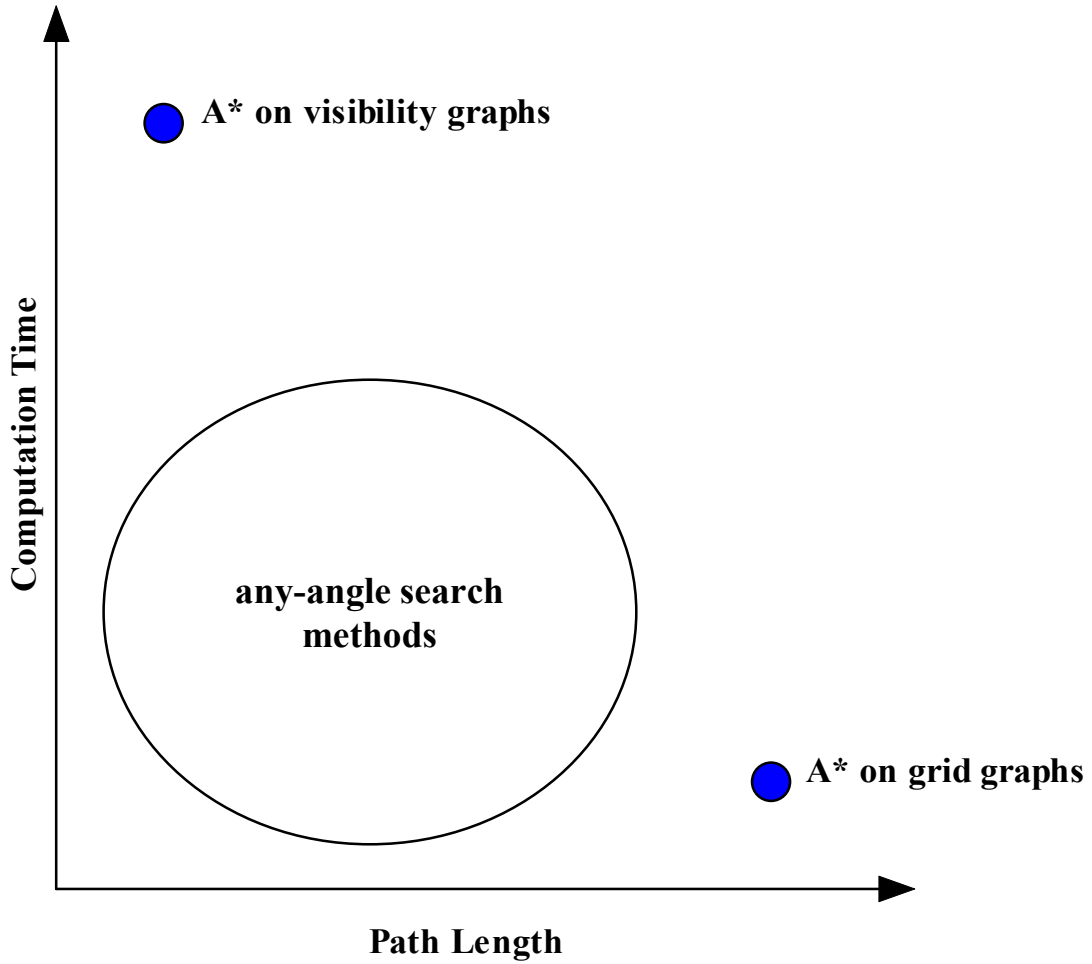


figure is notional

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)

Theta*

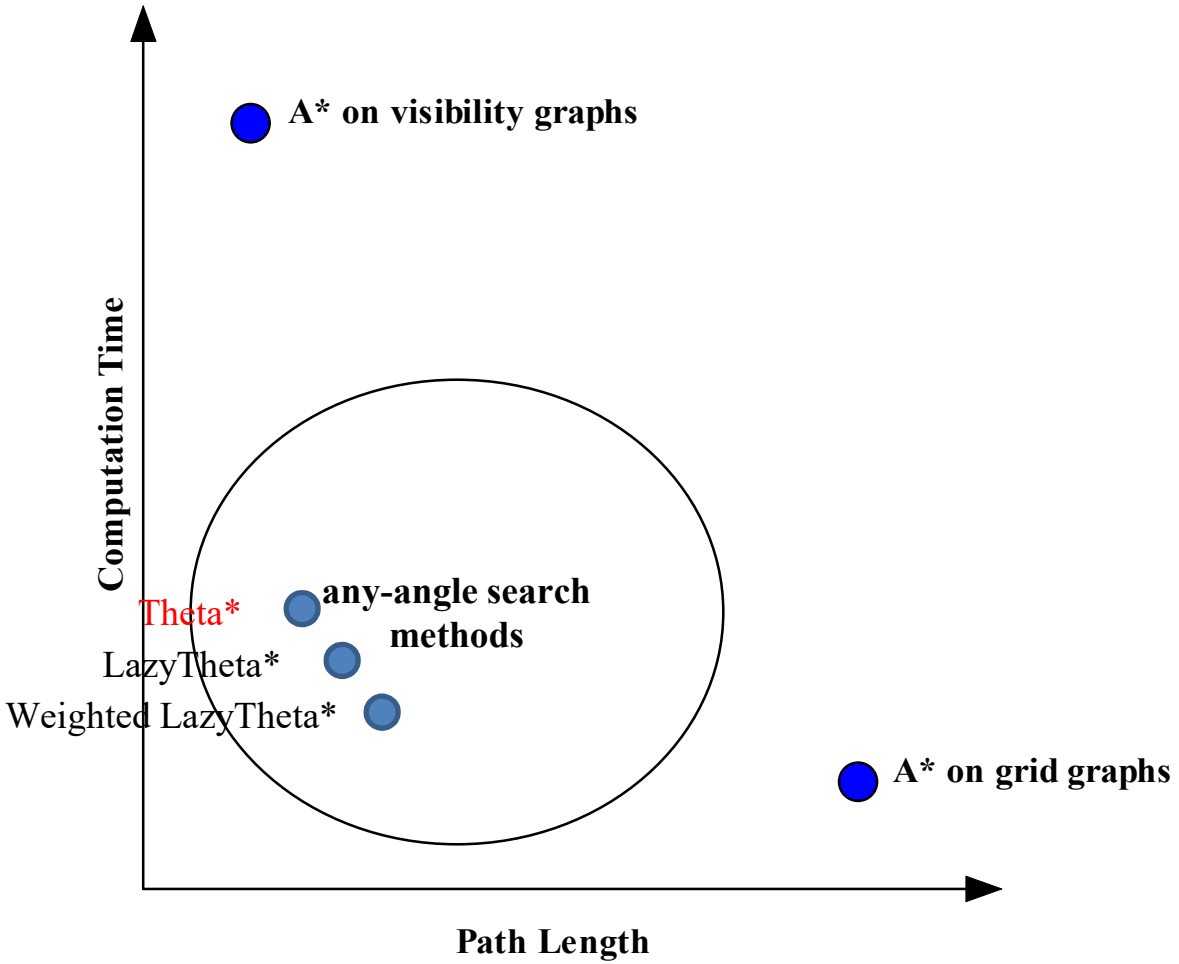
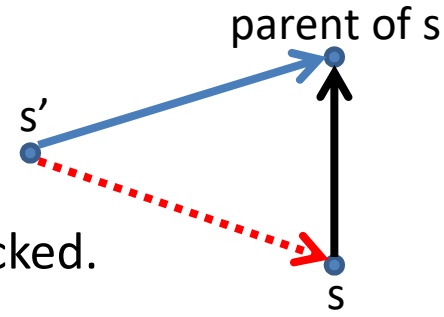


figure is notional

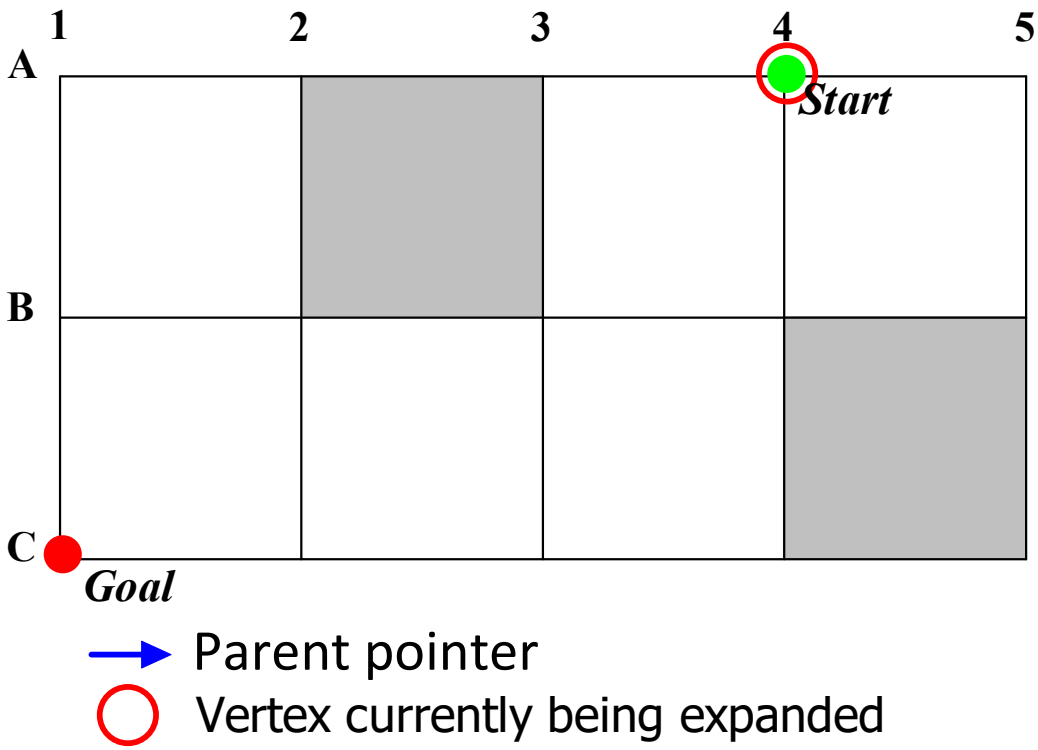
Theta*

- 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*

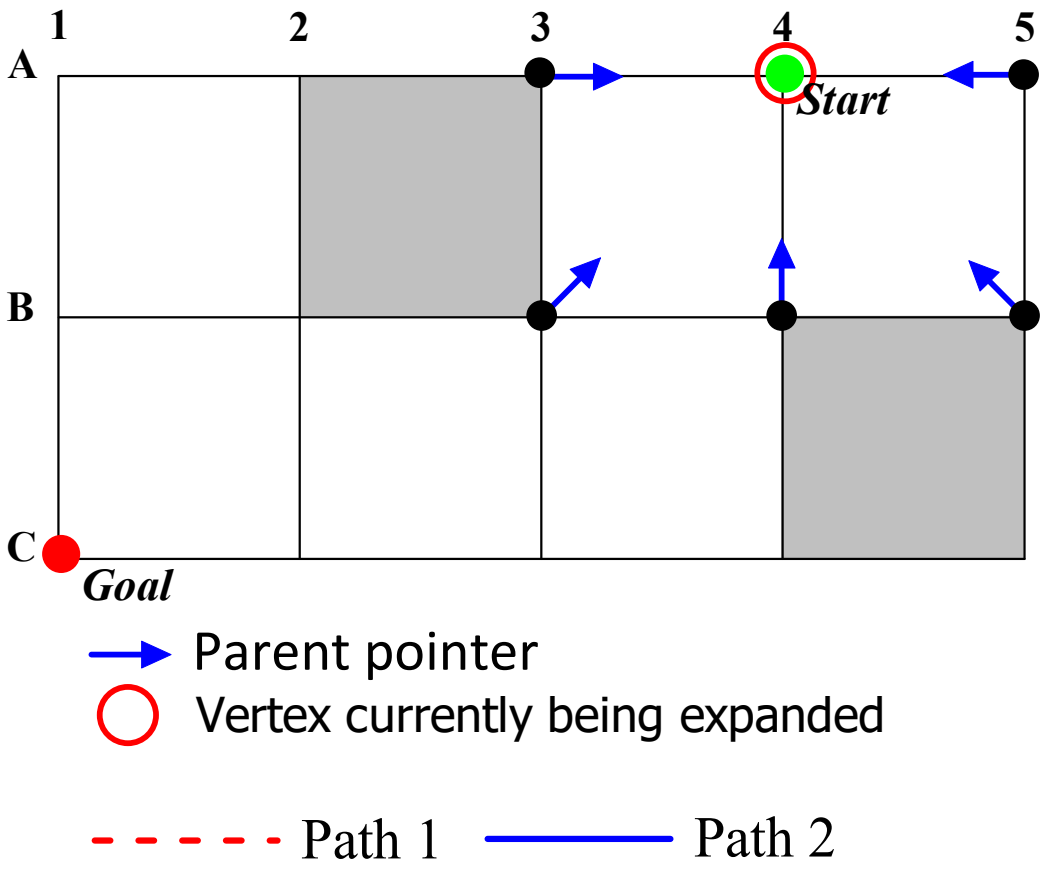
- Theta*



8-neighbor grid

Theta*

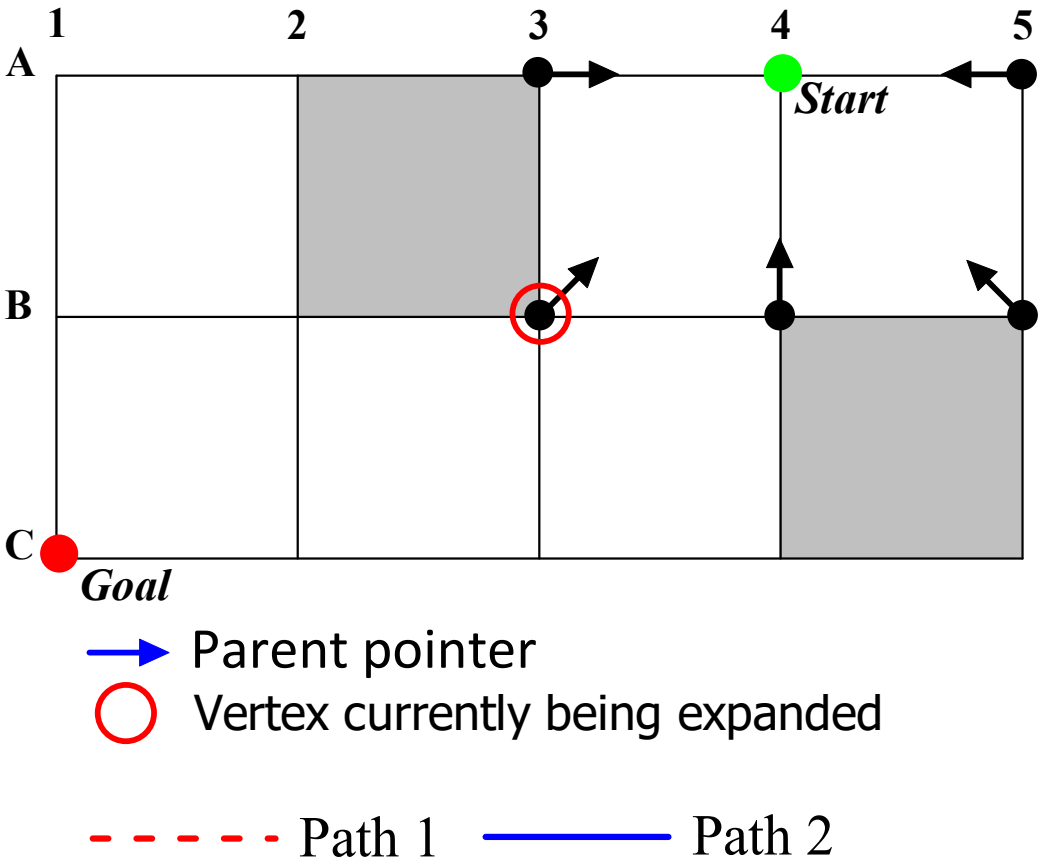
- Theta*



8-neighbor grid

Theta*

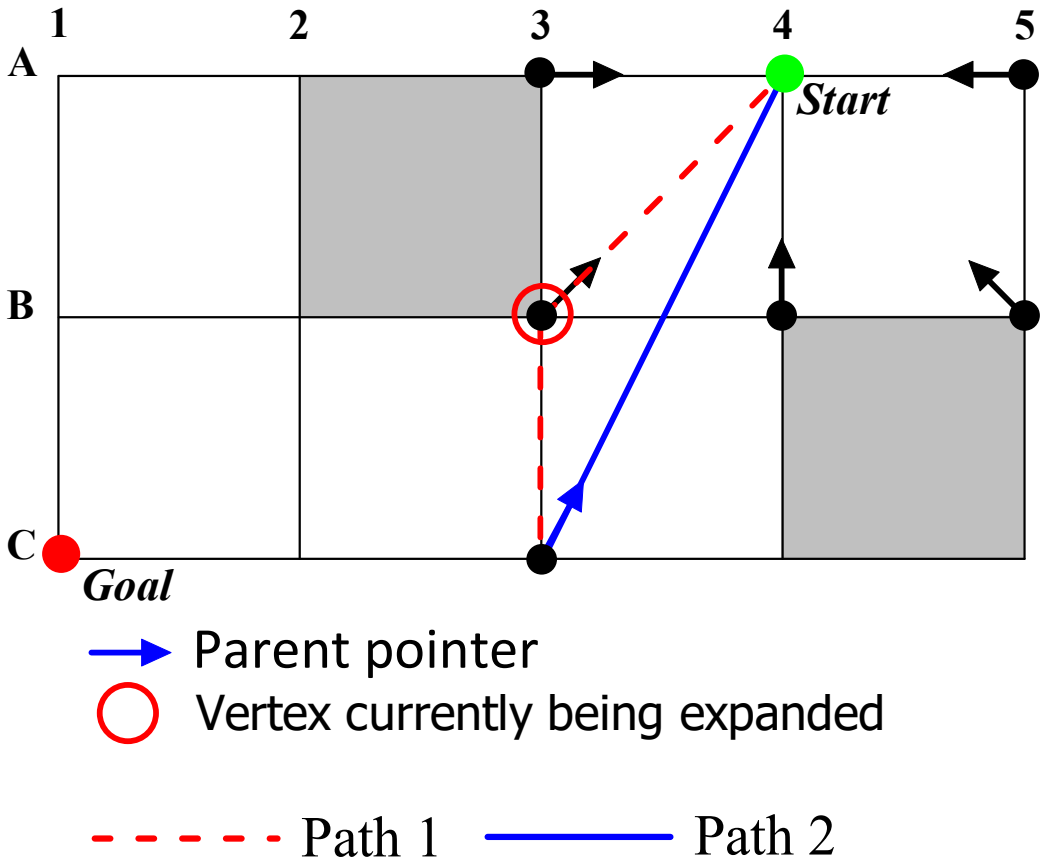
- Theta*



8-neighbor grid

Theta*

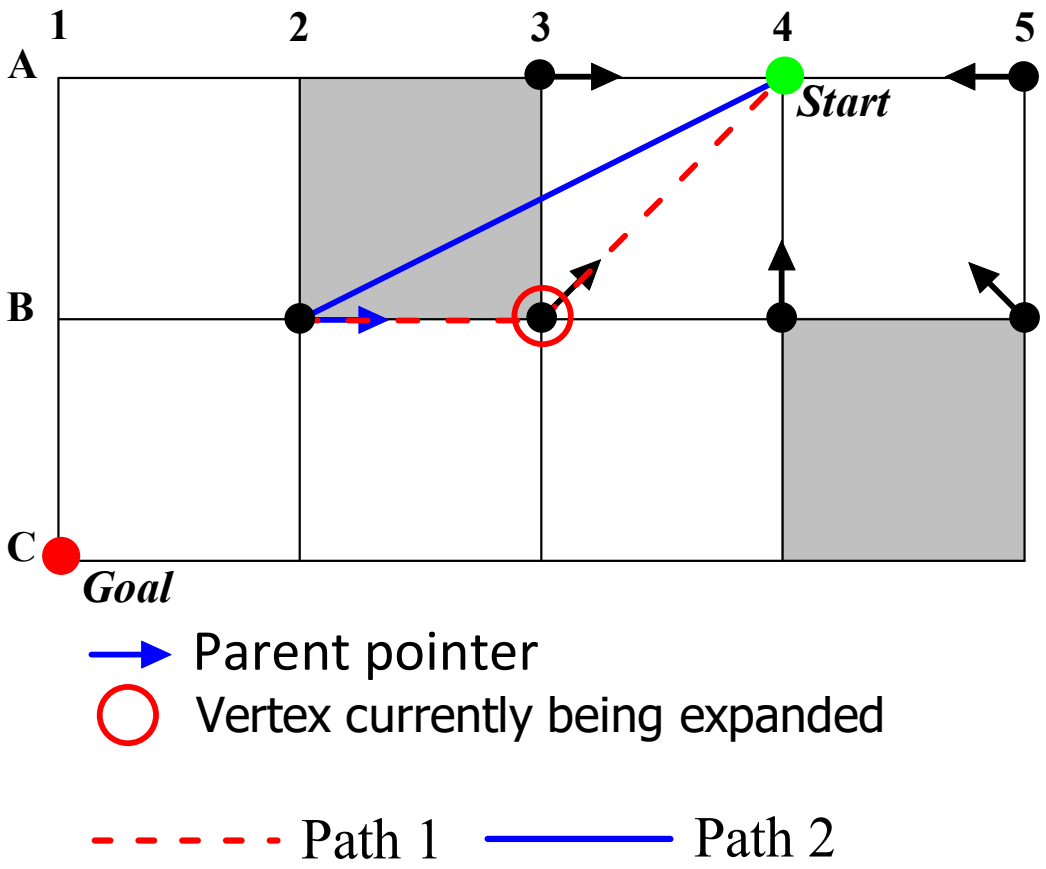
- Theta*



8-neighbor grid

Theta*

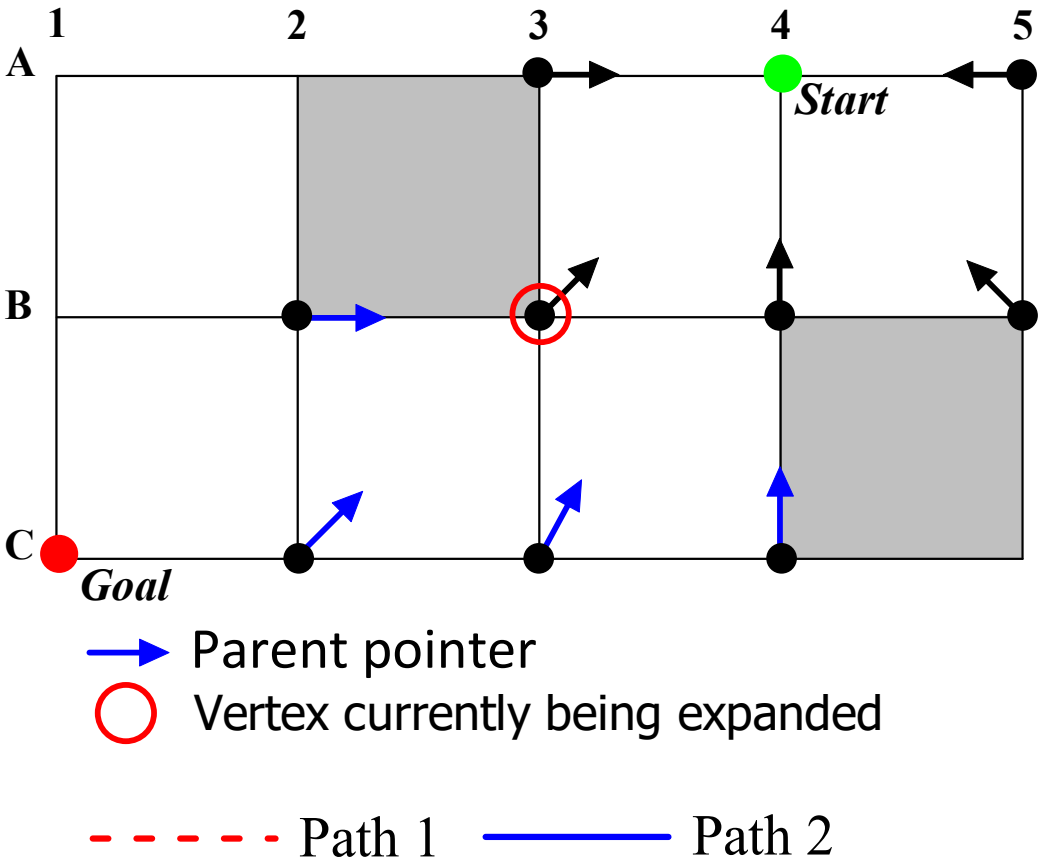
- Theta*



8-neighbor grid

Theta*

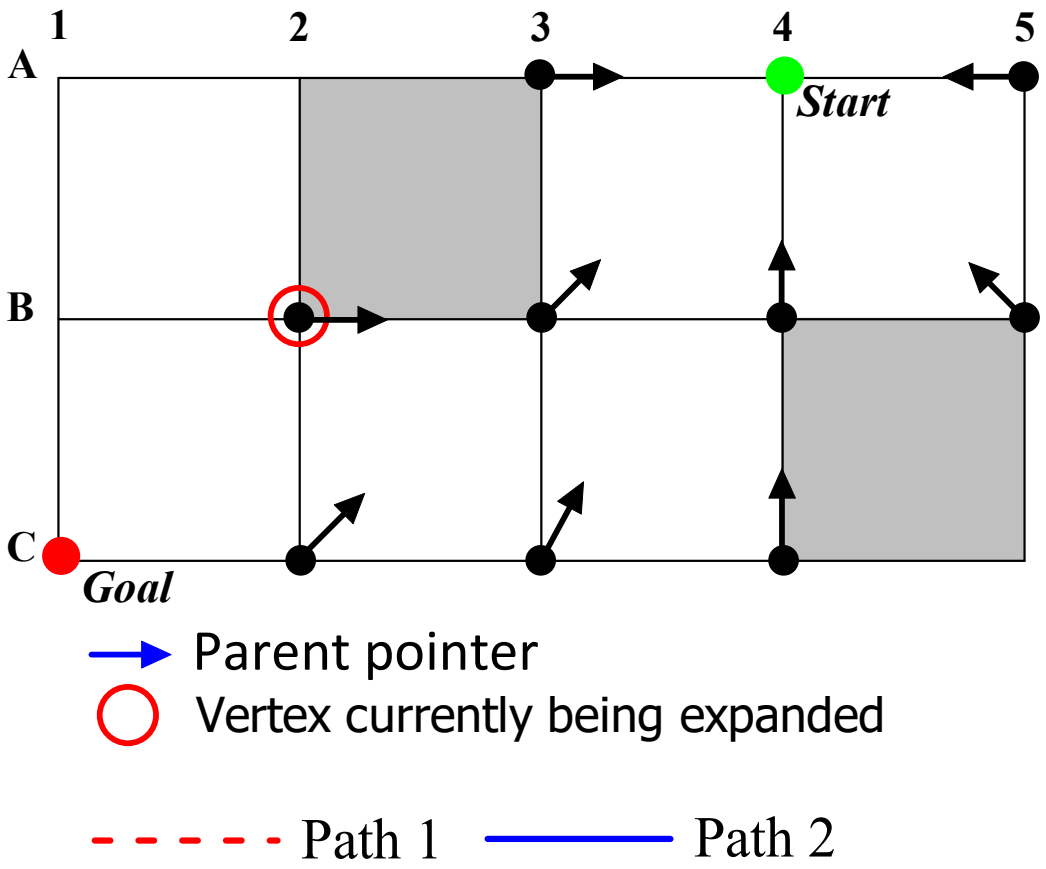
- Theta*



8-neighbor grid

Theta*

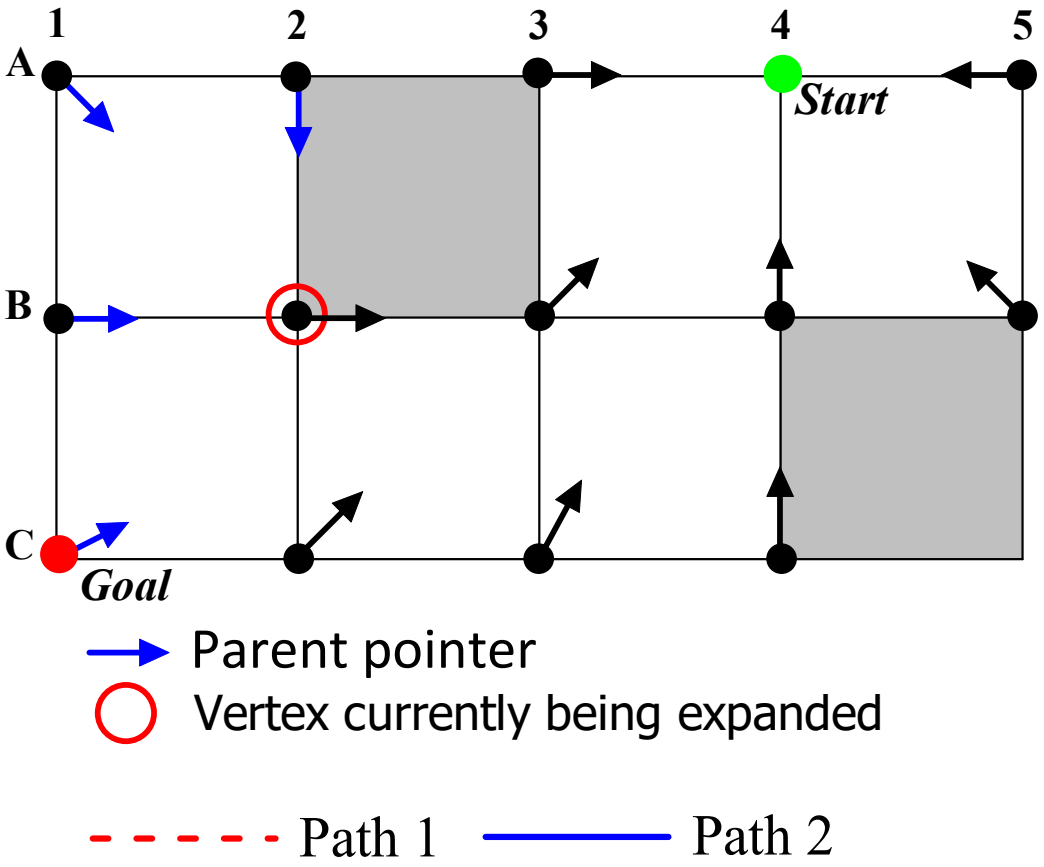
- Theta*



8-neighbor grid

Theta*

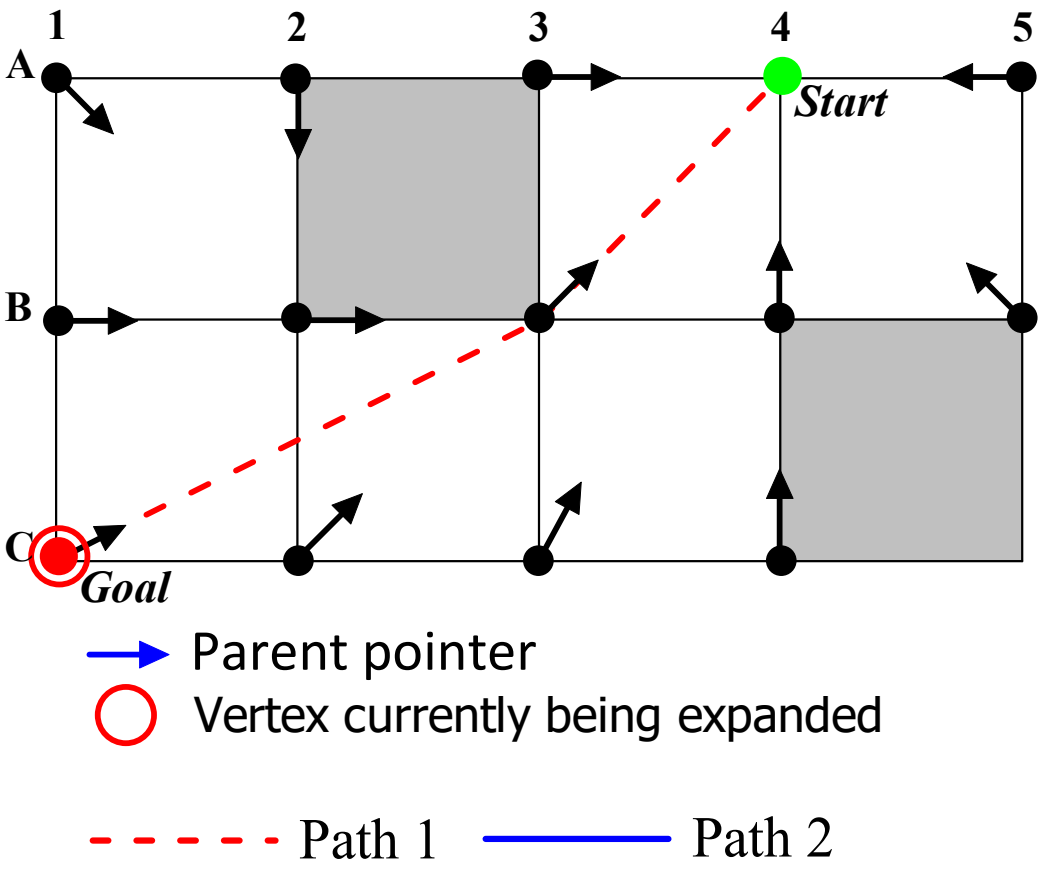
- Theta*



8-neighbor grid

Theta*

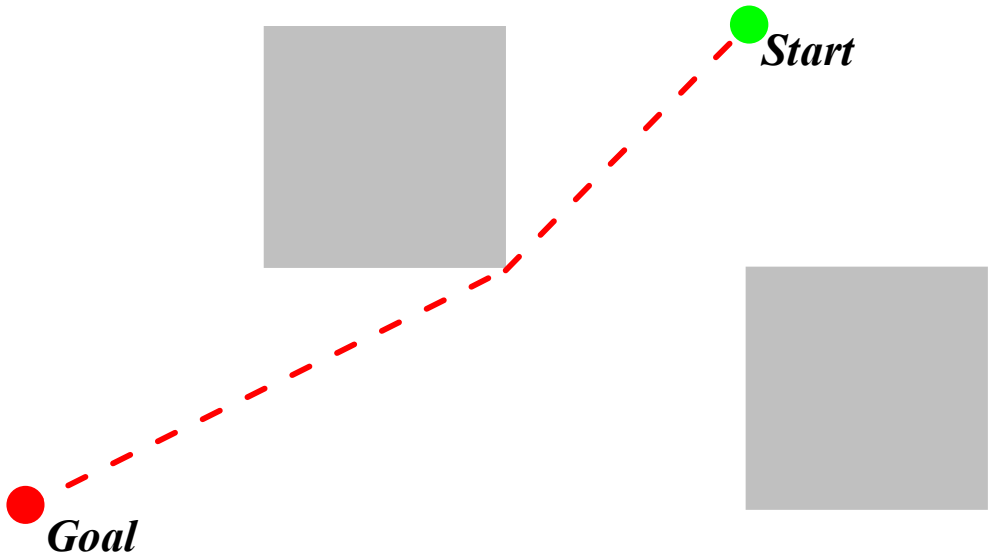
- Theta*



8-neighbor grid

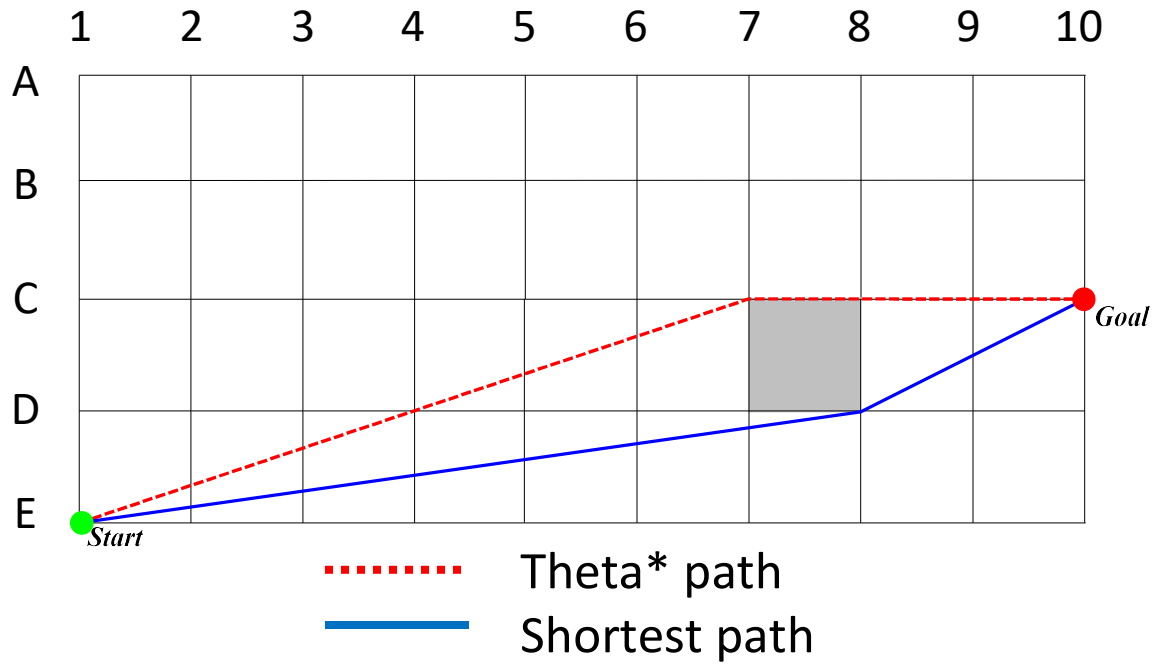
Theta*

- Theta*



Theta*

- 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

8-neighbor grid

Lazy Theta*

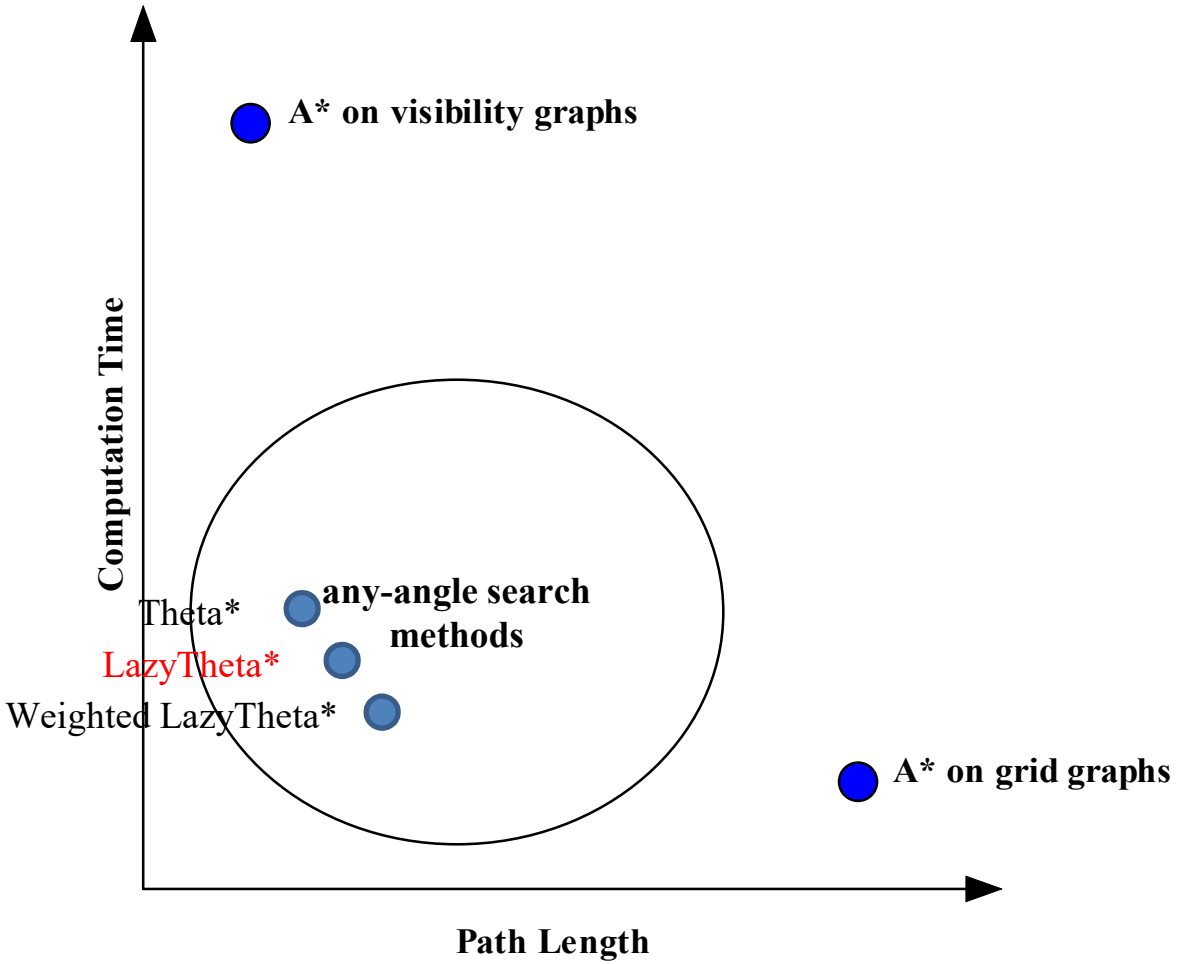


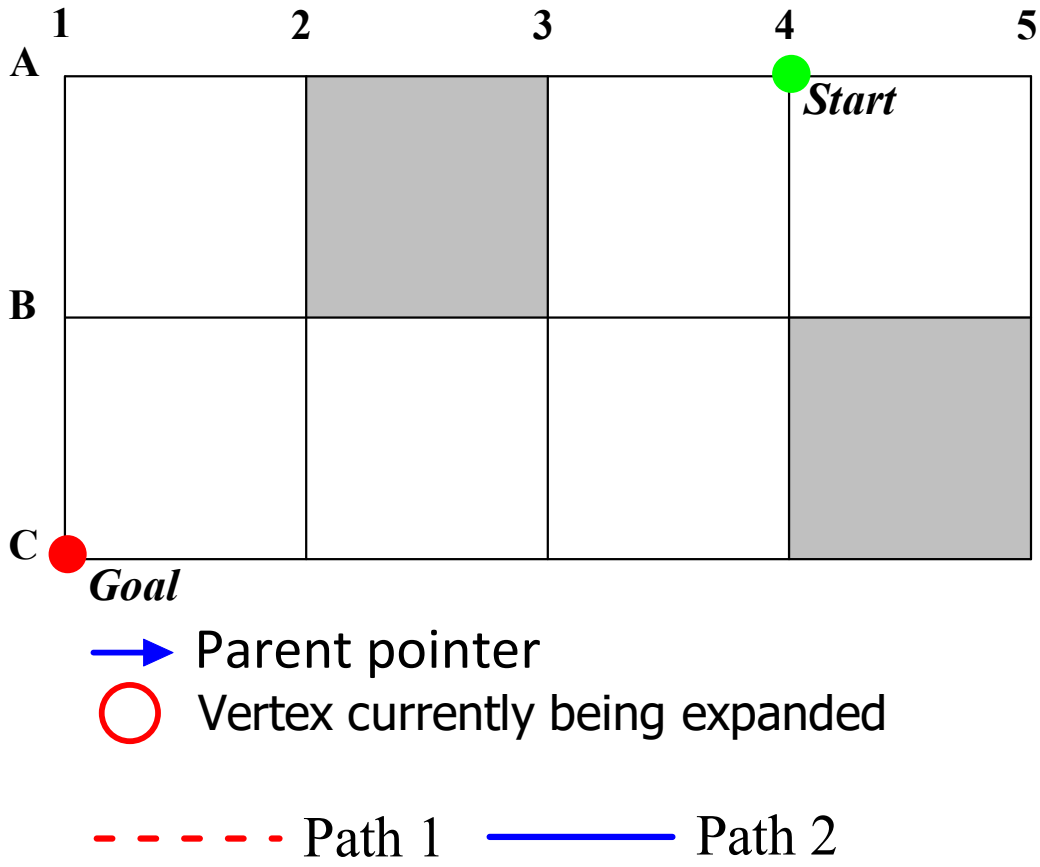
figure is notional

Lazy Theta*

- 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.]
  - Thus, Lazy Theta* performs one line-of-sight check only for each expanded vertex while Theta* performs one line-of-sight check for each generated vertex

Lazy Theta*

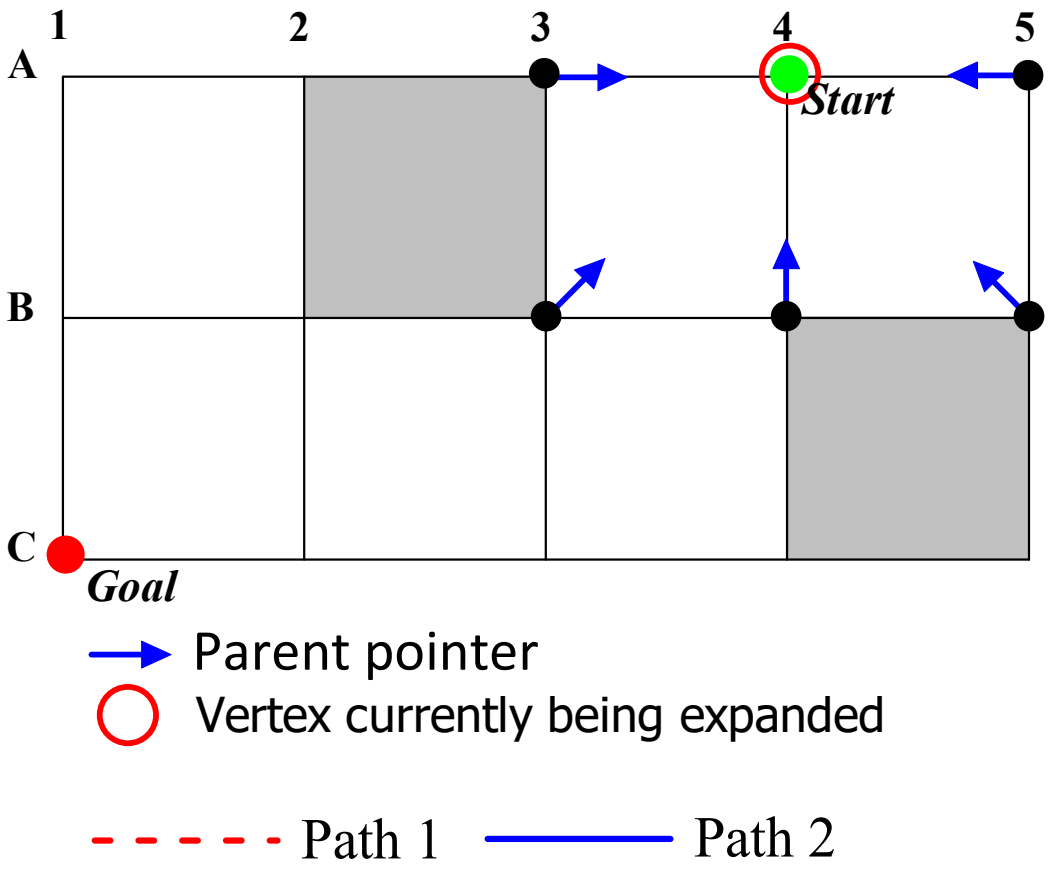
- Lazy Theta*



8-neighbor grid

Lazy Theta*

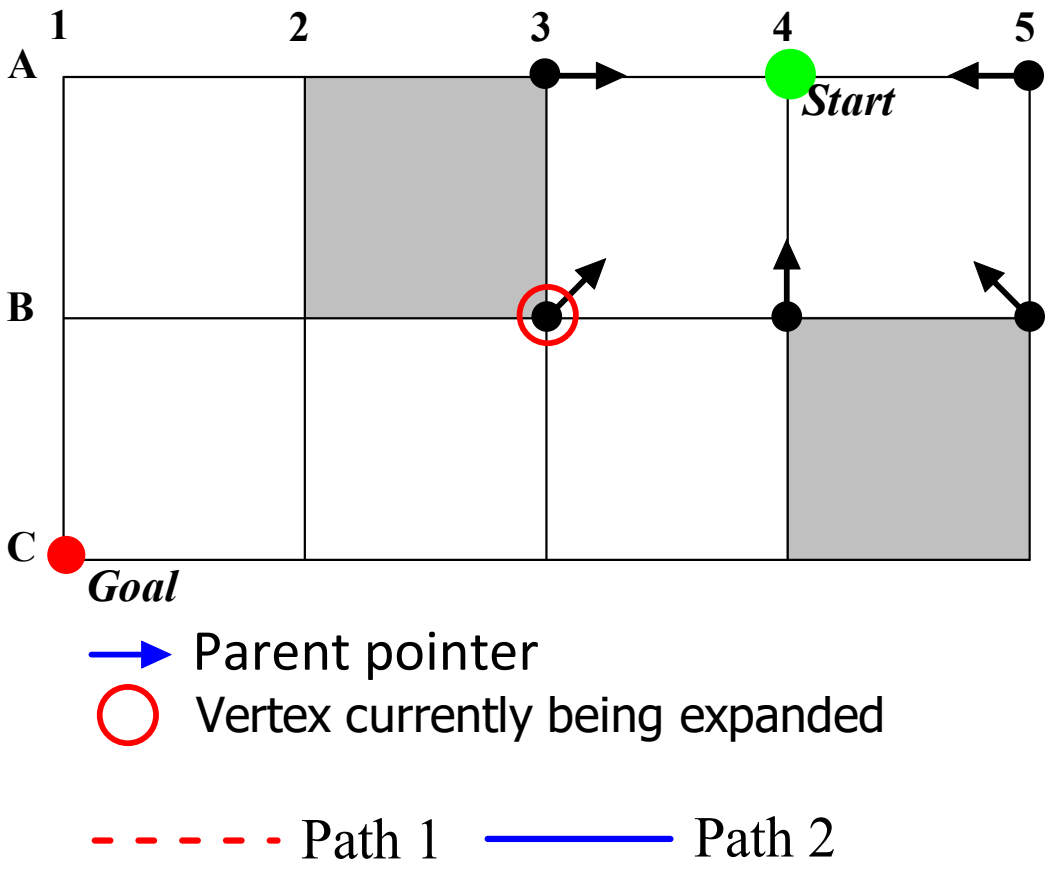
- Lazy Theta*



8-neighbor grid

Lazy Theta*

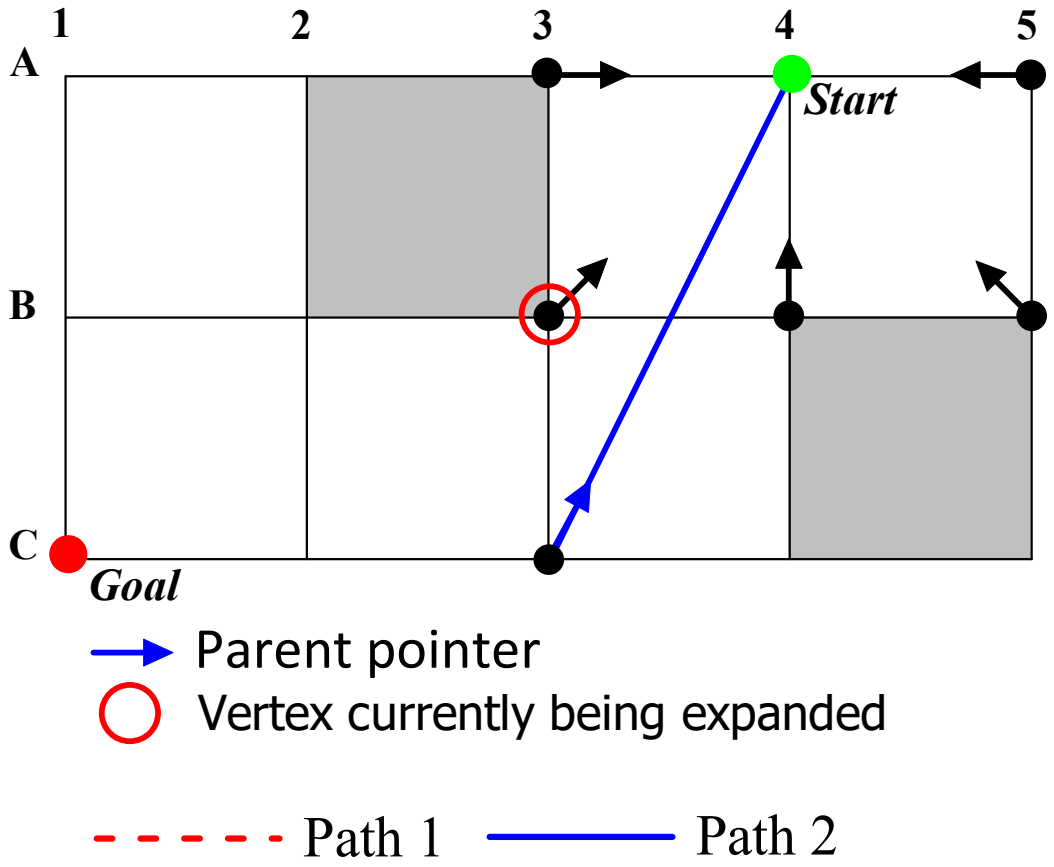
- Lazy Theta*



8-neighbor grid

Lazy Theta*

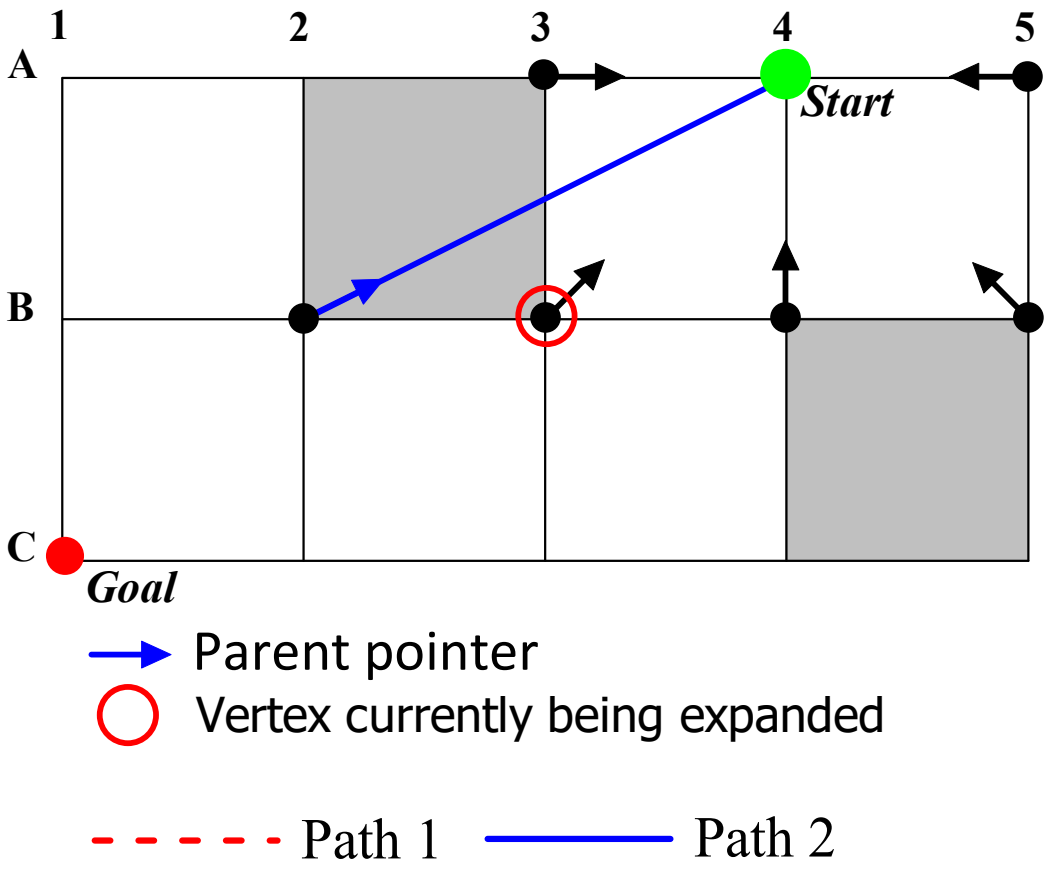
- Lazy Theta*



8-neighbor grid

Lazy Theta*

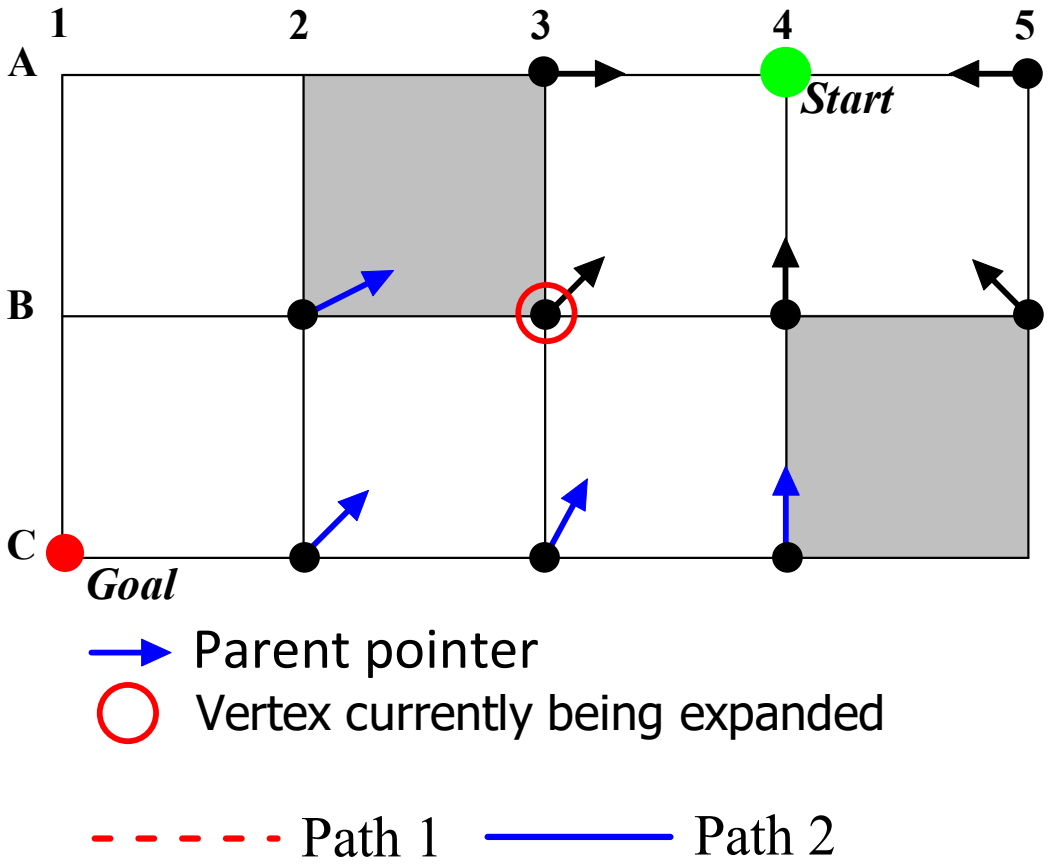
- Lazy Theta*



8-neighbor grid

Lazy Theta*

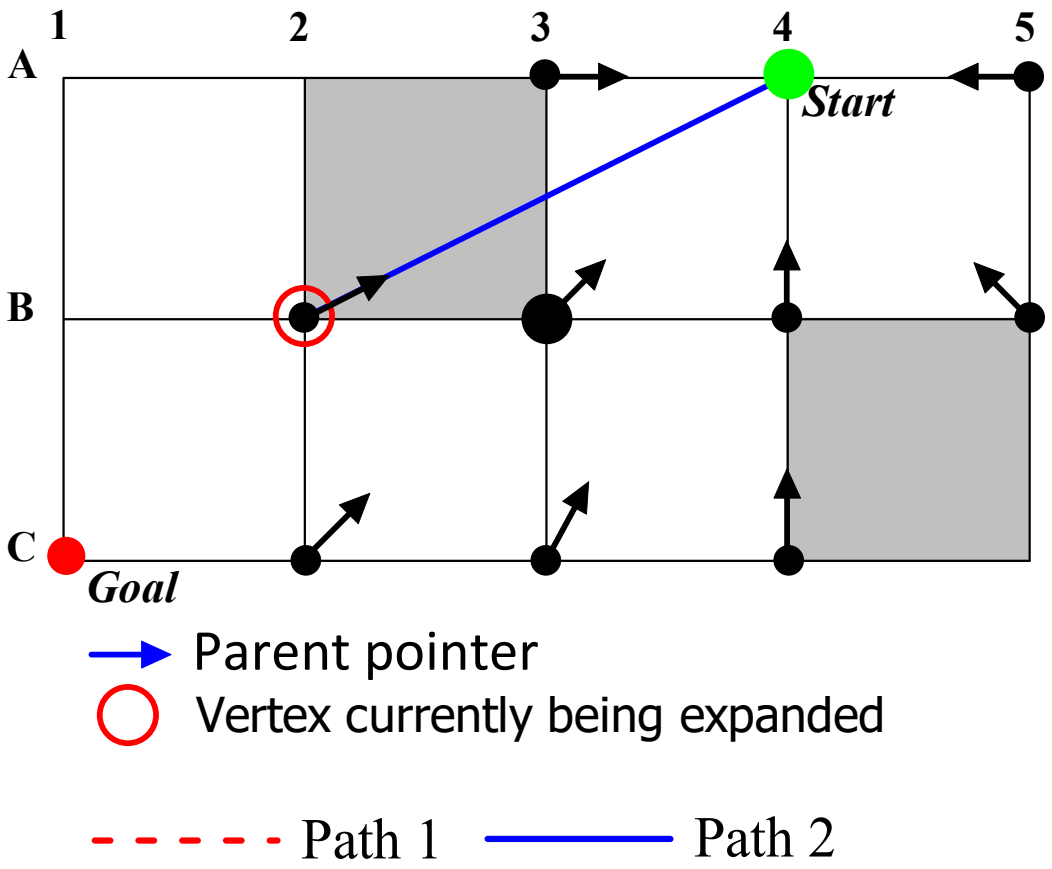
- Lazy Theta*



8-neighbor grid

Lazy Theta*

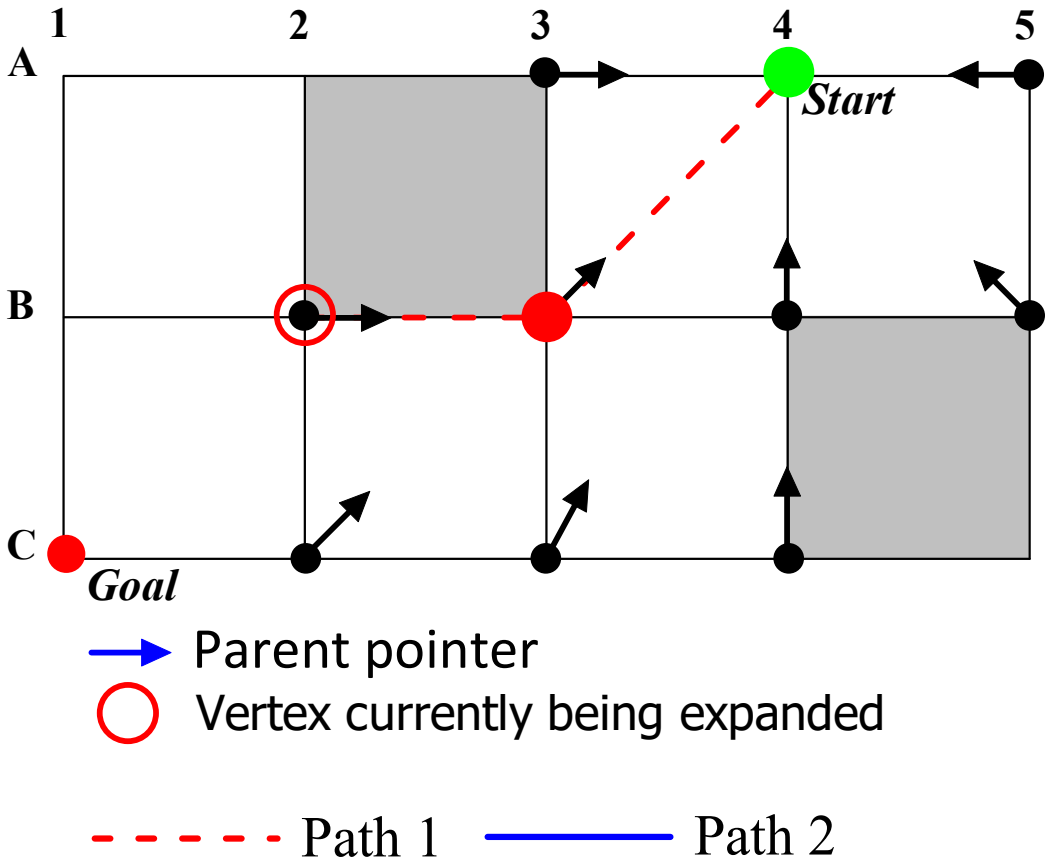
- Lazy Theta*



8-neighbor grid

Lazy Theta*

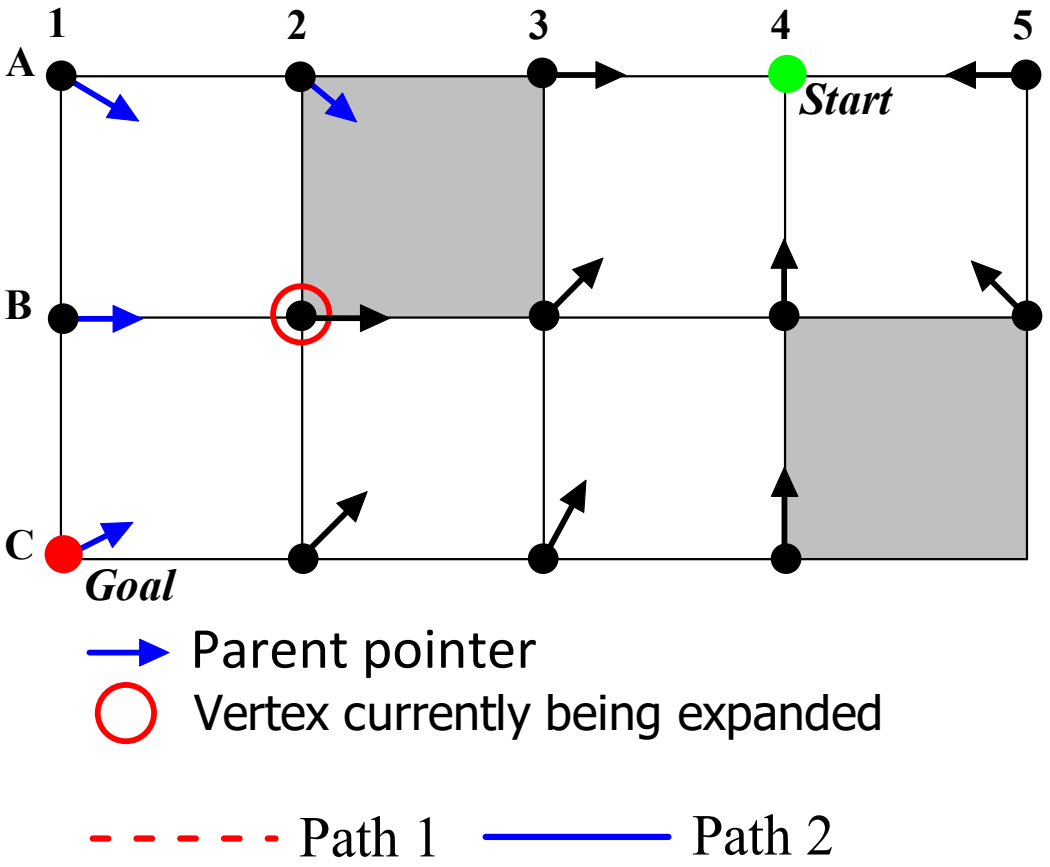
- Lazy Theta*



8-neighbor grid

Lazy Theta*

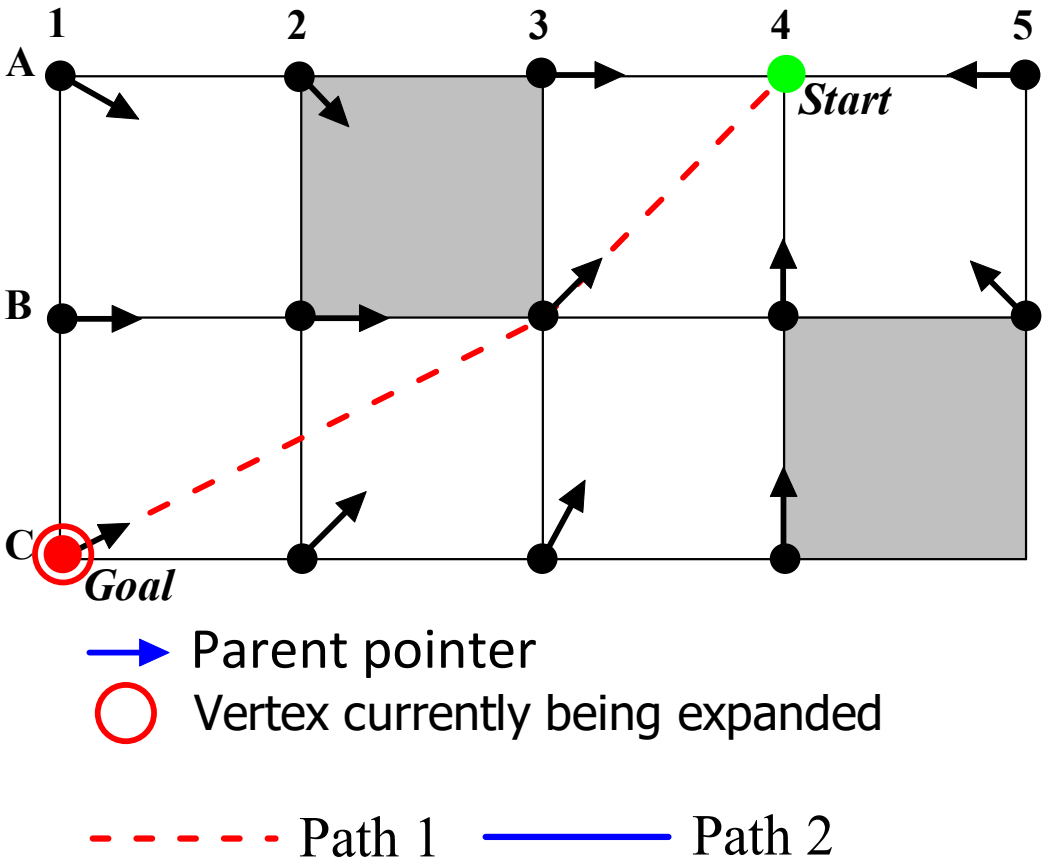
- Lazy Theta*



8-neighbor grid

Lazy Theta*

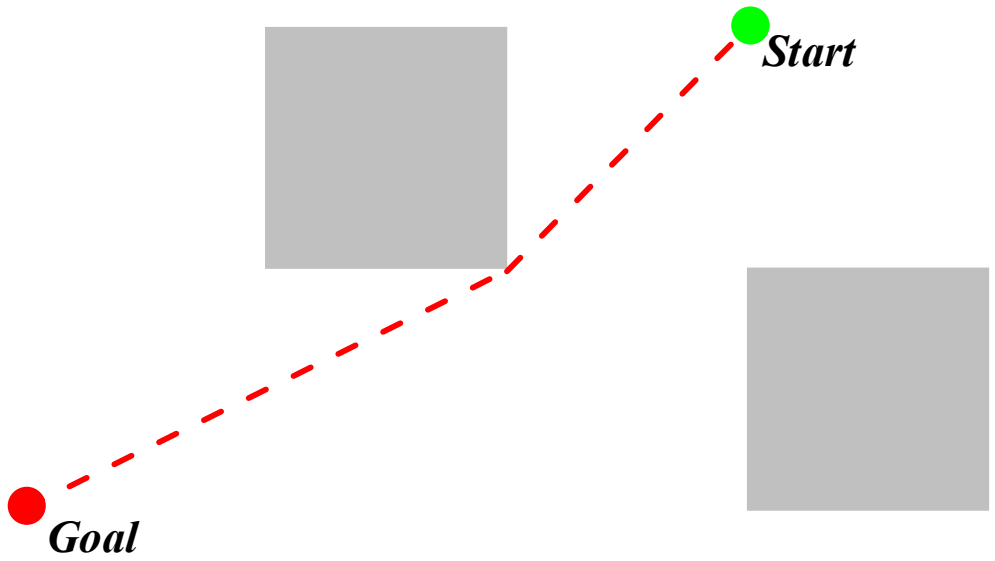
- Lazy Theta*



8-neighbor grid

Lazy Theta*

- Lazy Theta*



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

Weighted Lazy Theta*

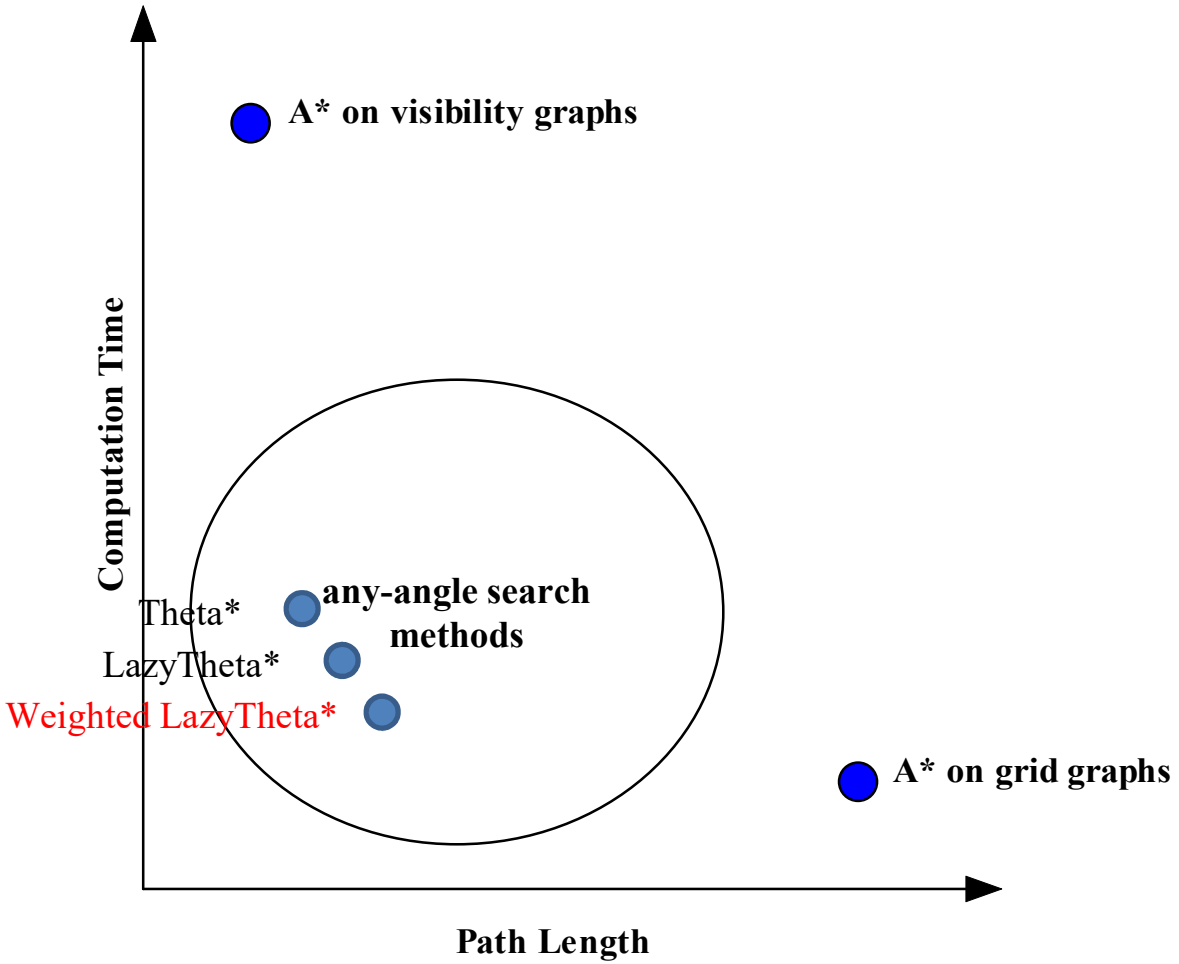
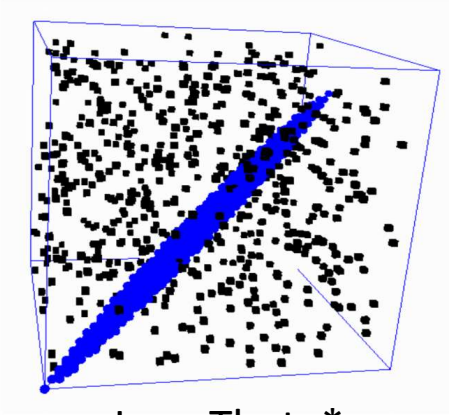


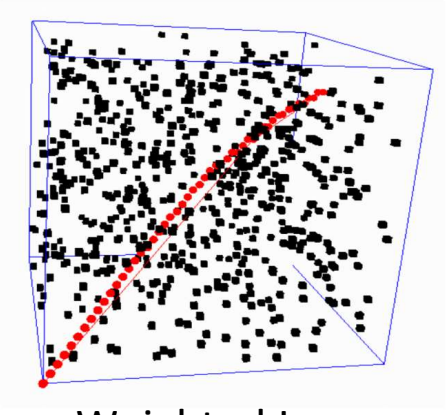
figure is notional

Weighted Lazy Theta*

- Lazy Theta* but based on weighted A* rather than A*
- Reduces vertex expansions and thus line-of-sight checks
- Both reductions reduce the computation time
- 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 w =1.3

26-neighbor grid

Alternatives to Theta*

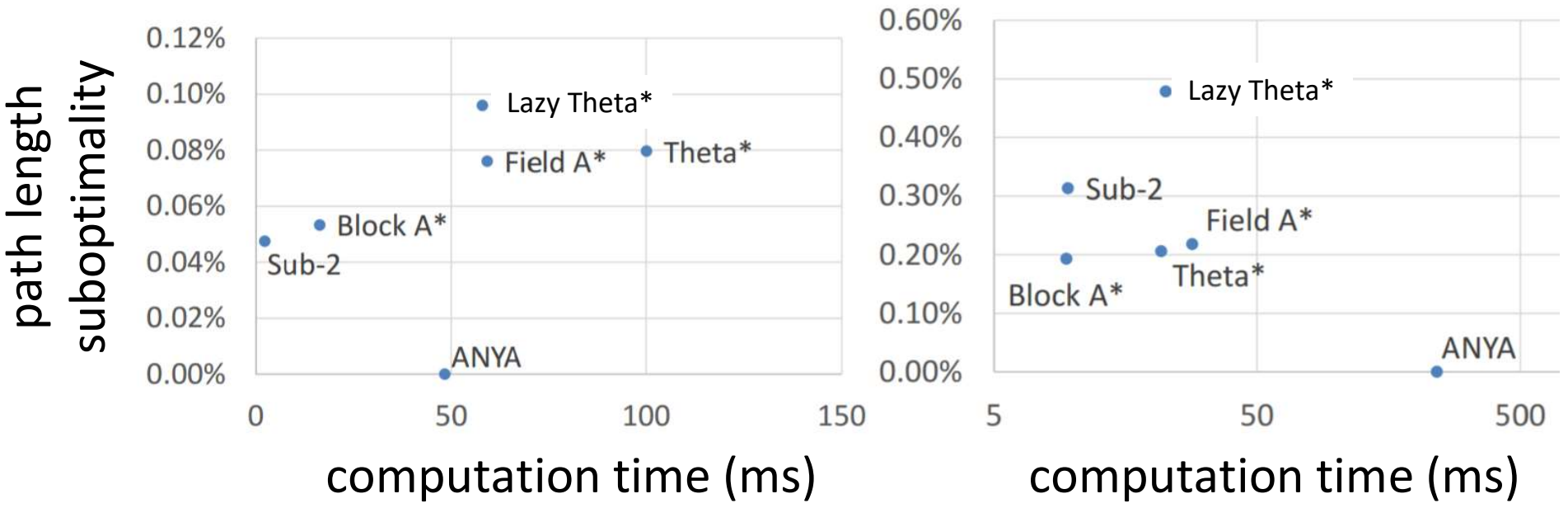
[from JPL]

- Other any-angle search algorithms
  - Accelerated A* [Sislak et al.]
    a sophisticated version of Theta*
  - Field D* [Ferguson and Stentz]
    an any-angle version of D* (Lite) with interpolation
  - Block A* [Yap et al.]
    an any-angle version of A* that operates on blocks of cells
  - Anya [Harabor et al.]
    an any-angle search method for 2D that finds shortests paths

[work done by different research groups, not me]

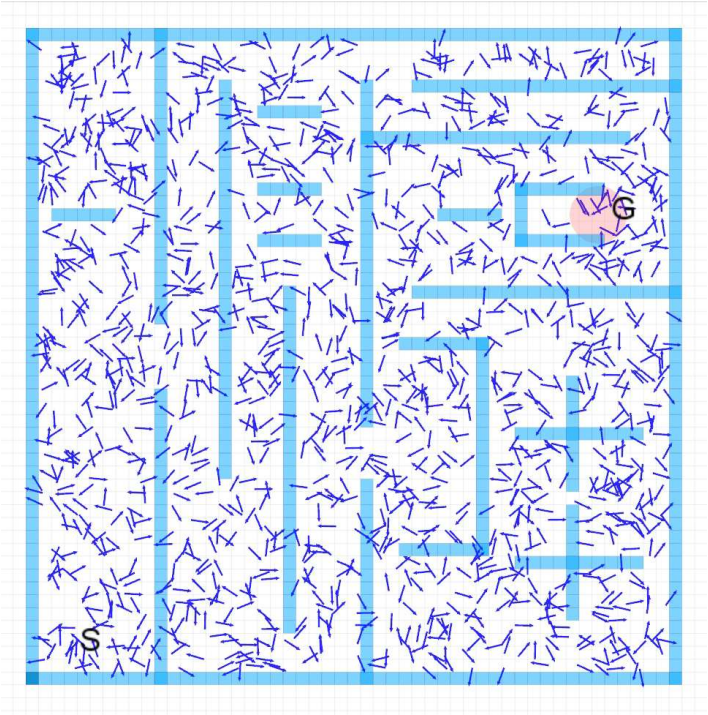
Experimental Results

- Sub-2: Theta* + subgoal graphs/“contraction hierarchies”
- Anya [Harabor et al.]: shortest any-angle search in 2D

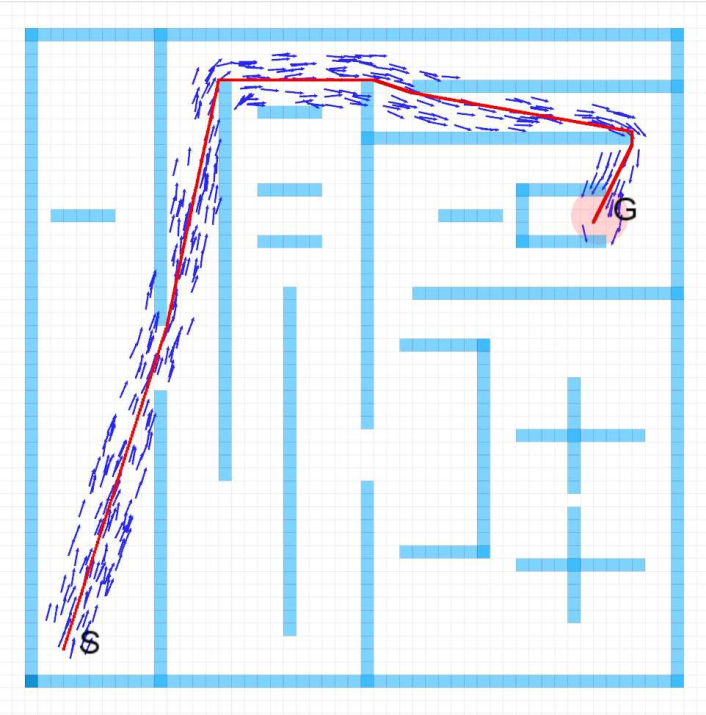


Application: Theta*-RRT

- Theta*-RRT is a variant of RRT that grows the trees in more focused ways



Uniform Sampling in RRT

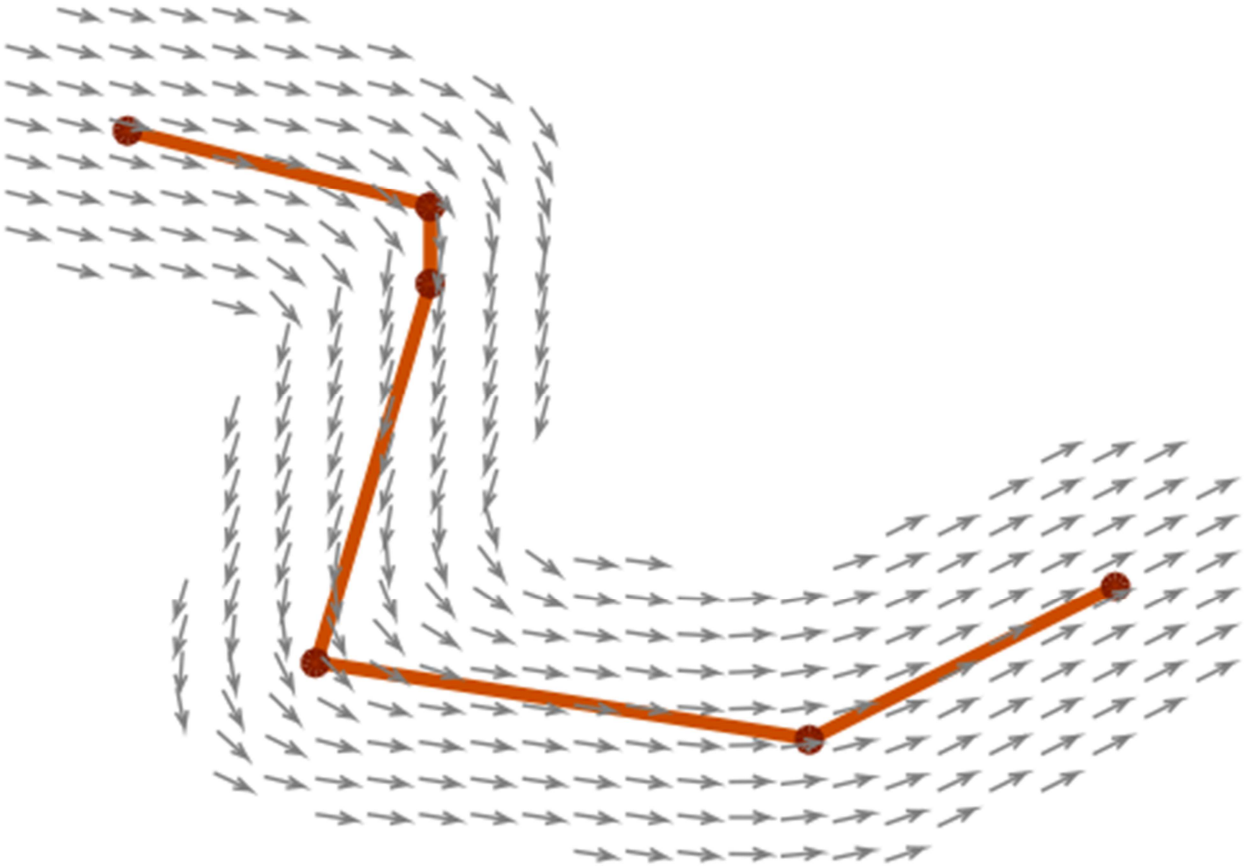


Sampling in Theta*-RRT

([Brunnen et al.] suggested earlier to use A*)

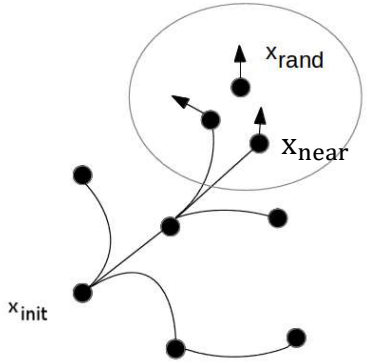
Application: Theta*-RRT

- Theta*-RRT generates samples around a feasible any-angle path



Application: Theta*-RRT

- Theta*-RRT connects x_{rand} to the minimum cost-to-go tree vertex

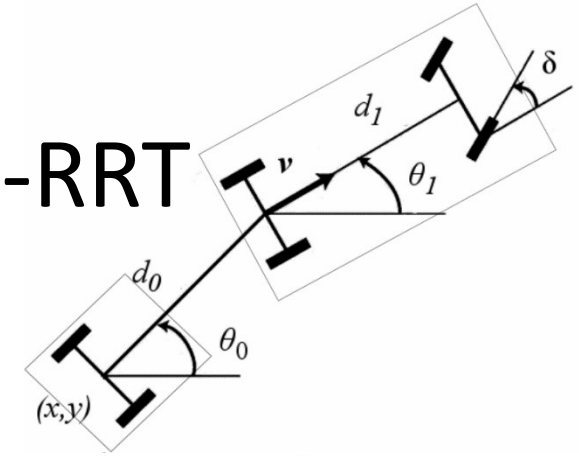


$$x_{near} = \underset{x \in X_{near}}{\operatorname{argmin}} C(x, x_{rand})$$

$$C(x, x_{rand}) = g(x) + C_\sigma(x, x_{rand}) + D_P(x, x_{rand})$$

  - $g(x)$ is the sum of the costs from the tree root x_{init} to the tree vertex x
  - $C_\sigma(x, x_{rand})$ measures the length and smoothness of the trajectory from x to x_{rand}
  - $D_P(x, x_{rand})$ is the geodesic distance of x and x_{rand} to path P

Application: Theta*-RRT



- 8D truck and trailer robot

Random environment									
Planner	Tree size $\mathbf{N_v}$		Planning time $\mathbf{T_s}$ $[s]$		Trajectory length $\mathbf{l_p}$ $[m]$		Roughness $\mathbf{R}$		Problems solved
Theta*-RRT	**52.2**	± 48.3	**0.0547**	± 0.0790	**44.331**	± 2.8418	**0.0057**	± 0.0060	100%
A*-RRT	75.7	± 52.4	0.1019	± 0.0984	51.74	± 7.89	3.4993	± 8.8502	58%
RRT	836	± 378	1.32	± 0.84	66.96	± 14.7	2.17	± 2.00	100%
RRT*	3957	± 2756	816.16	± 656.58	52.39	± 13.12	0.54	± 1.01	76%
A*-RRT* [7]	3582	± 3138	949.6	± 823.7	49.30	± 12.79	0.1013	± 0.2647	100%
Maze environment									
Planner	Tree size $\mathbf{N_v}$		Planning time $\mathbf{T_s}$ $[s]$		Trajectory length $\mathbf{l_p}$ $[m]$		Roughness $\mathbf{R}$		Problems solved
Theta*-RRT	**522**	± 167	**2.57**	± 1.50	**98.59**	± 4.95	**1.0073**	± 0.7226	100%
A*-RRT	661	± 181	4.56	± 2.0858	101.79	± 8.26	1.1317	± 1.0372	100%
RRT	4858	± 1276	38.88	± 15.83	126.34	± 16.52	2.0788	± 1.2985	100%
RRT*	0	± 0	0	± 0	0	± 0	0	± 0	0% – **failed**
A*-RRT* [7]	0	± 0	0	± 0	0	± 0	0	± 0	0% – **failed**
Narrow corridor environment									
Planner	Tree size $\mathbf{N_v}$		Planning time $\mathbf{T_s}$ $[s]$		Trajectory length $\mathbf{l_p}$ $[m]$		Roughness $\mathbf{R}$		Problems solved
Theta*-RRT	**1513**	± 492	**20.87**	± 13.77	**77.10**	± 6.75	2.15	± 1.12	100%
A*-RRT	2139	± 573	33.46	± 16.74	79.66	± 5.94	**1.9352**	± 0.8722	100%
RRT	1794	± 5473	733.98	± 438.28	83.77	± 7.04	2.31	± 1.47	100%
RRT*	0	± 0	0	± 0	0	± 0	0	± 0	0% – **failed**
A*-RRT* [7]	0	± 0	0	± 0	0	± 0	0	± 0	0% – **failed**

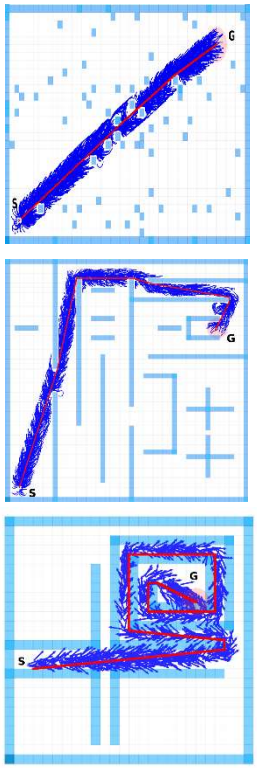
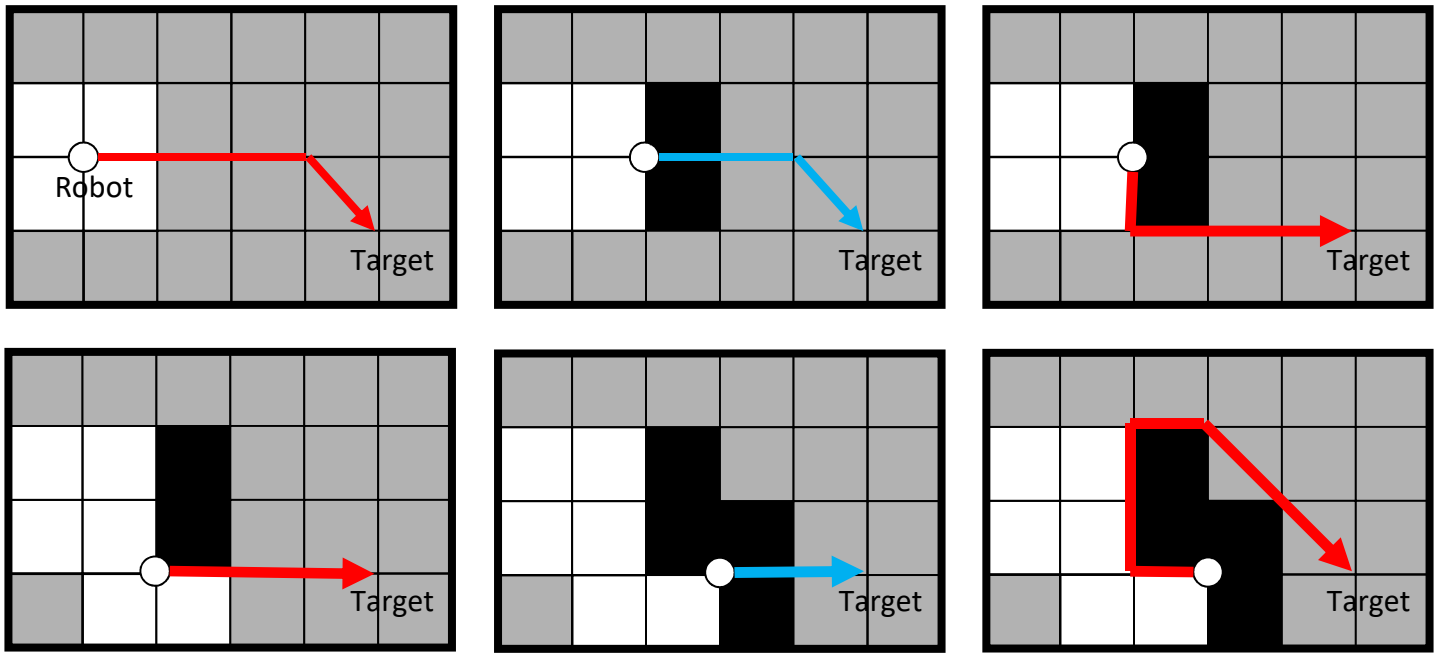


TABLE II

EXPERIMENTAL RESULTS: TRAJECTORY QUALITY AND PLANNING EFFICIENCY FOR THE TRUCK-AND-TRAILER SYSTEM.

Incremental Any-Angle Search

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



8-neighbor grid

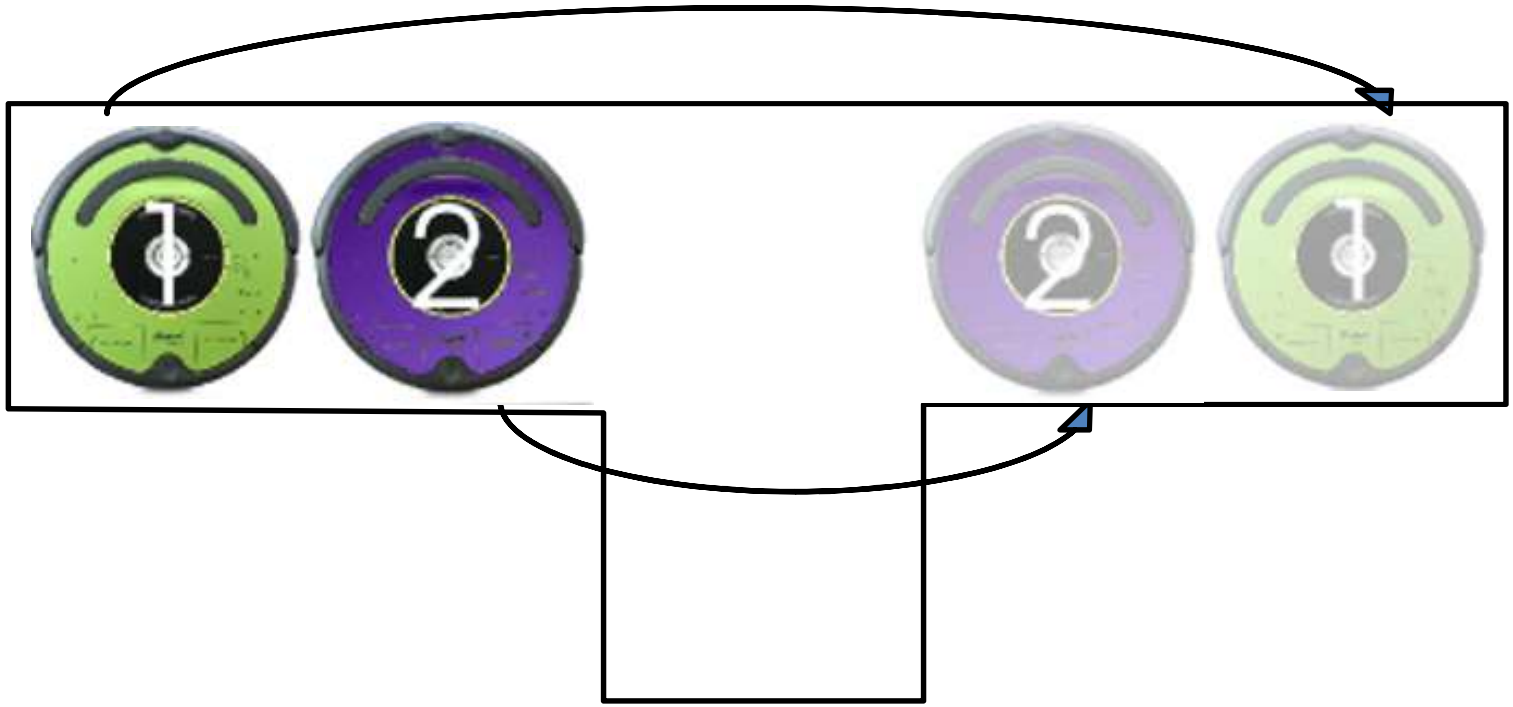
TOC

- Single-robot path planning
  - Runtime matters
    - Hierarchical search (via preprocessing)
    - Incremental search (via using experience with previous similar searches)
  - Quality-runtime tradeoff matters
    - Any-angle search
- Multi-robot path planning and execution

Multi-Robot Path Finding

- Multi-robot path finding
  - Given: a number of robots (each with a start and goal location) and a known environment
  - Task: find collision-free paths for the robots from their start to their goal locations that minimize some objective

Multi-Robot Path Finding

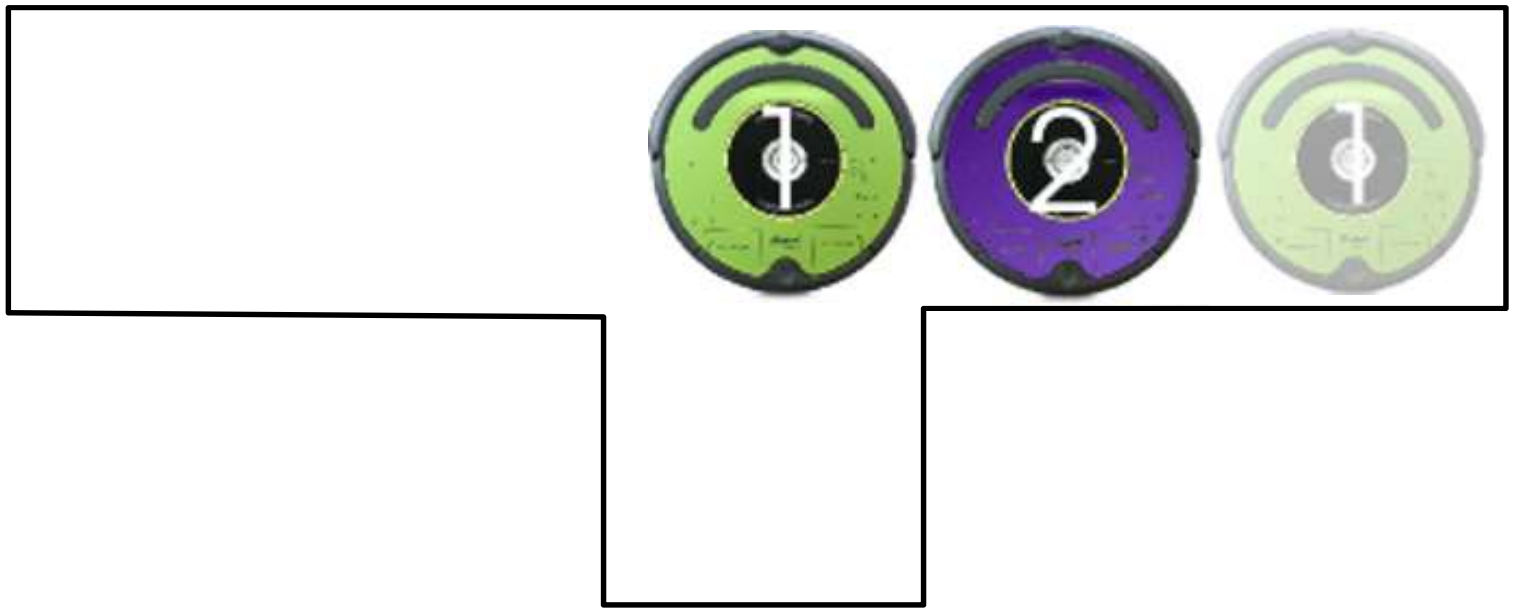


4-neighbor grid

Multi-Robot Path Finding

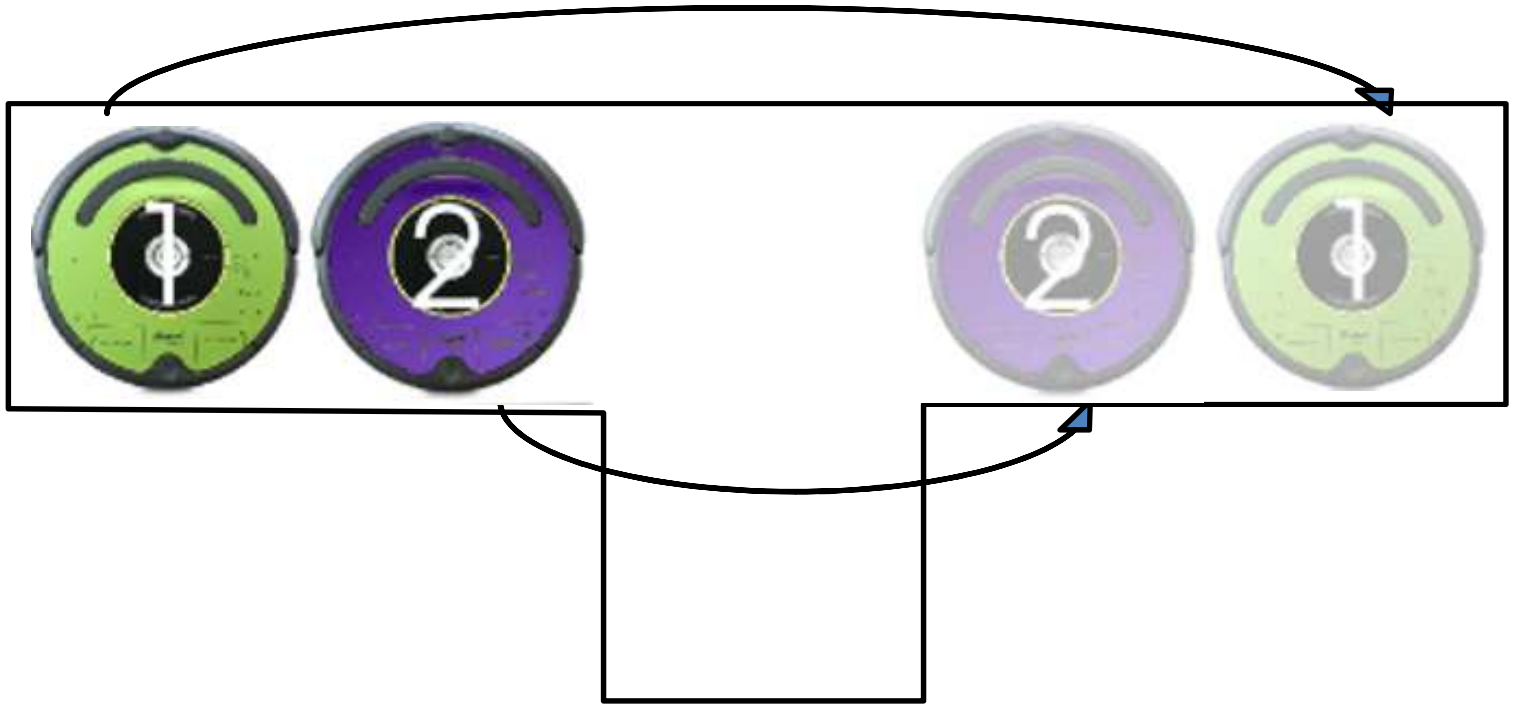
4-neighbor grid

Multi-Robot Path Finding



4-neighbor grid

Multi-Robot Path Finding



4-neighbor grid

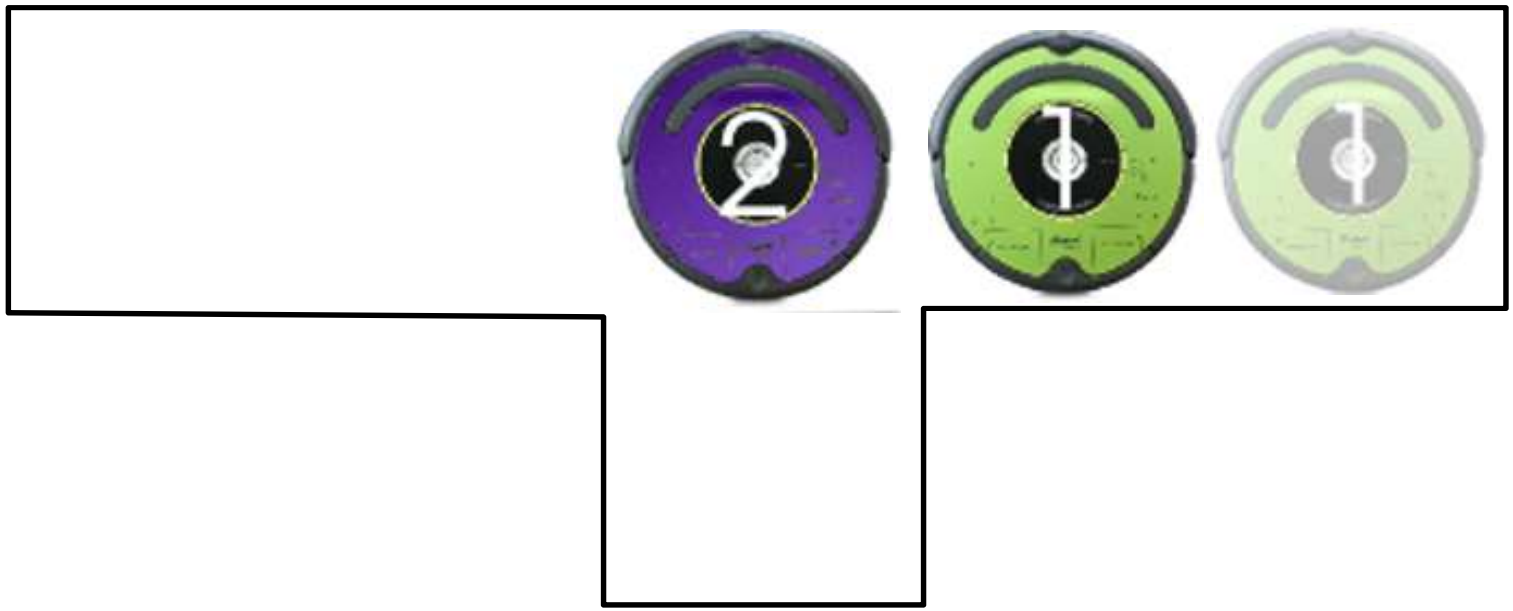
Multi-Robot Path Finding

4-neighbor grid

Multi-Robot Path Finding

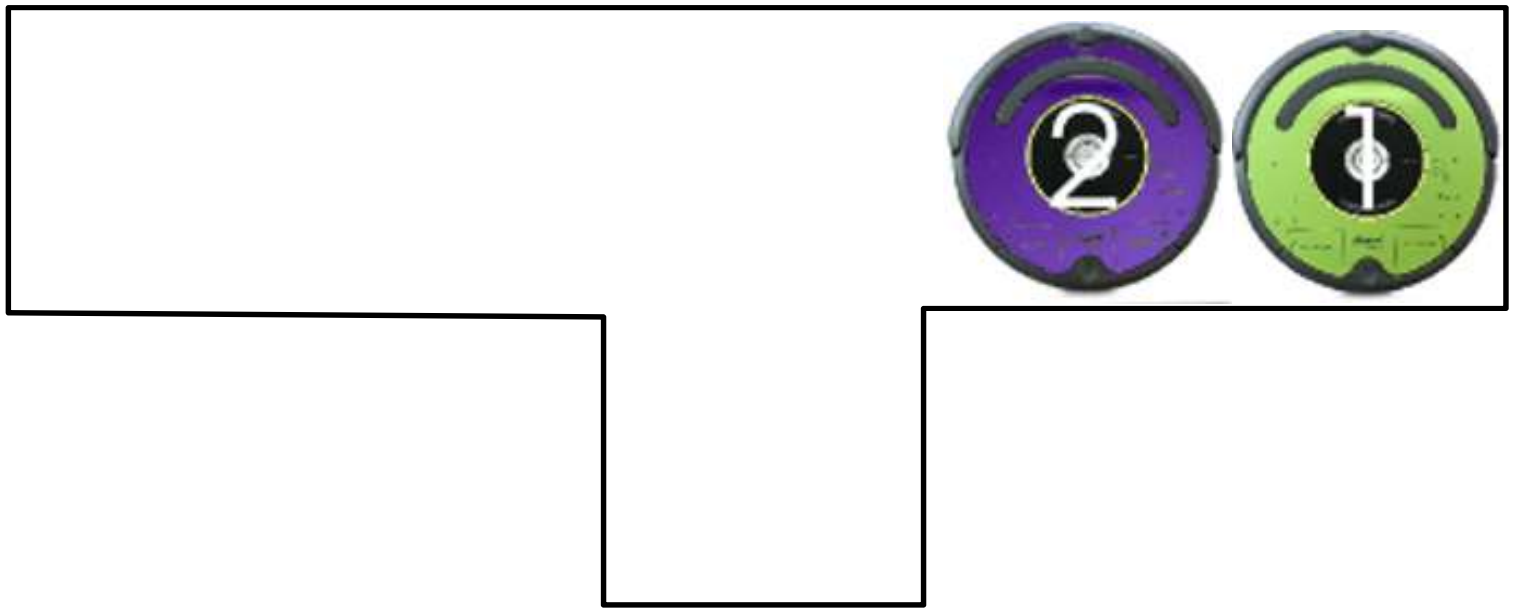
4-neighbor grid

Multi-Robot Path Finding



4-neighbor grid

Multi-Robot Path Finding

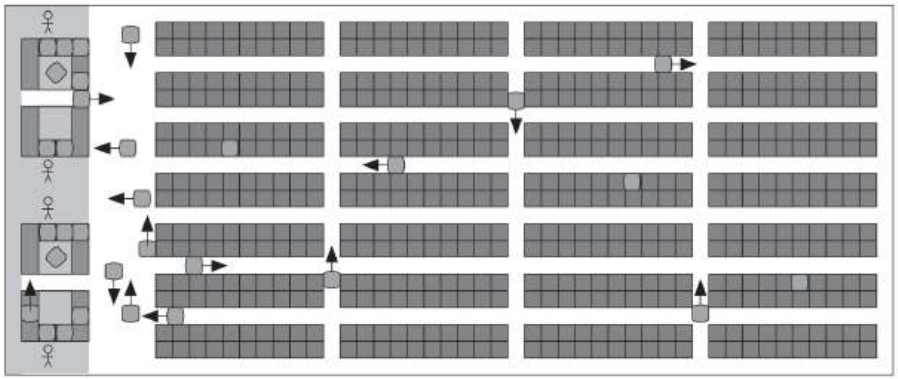


- Minimize makespan:
  latest arrival time of a robot at its goal location
- Minimize flowtime:
  sum of the arrival times of all robots at their goal locat

4-neighbor grid

Multi-Robot Path Finding

- Application: Amazon fulfillment centers



[work by Kiva Systems/Amazon Robotics, not me]

Multi-Robot Path Finding (MAPF)

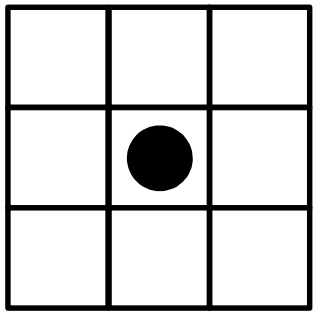
- Application: Amazon fulfillment centers

[work by Kiva Systems/Amazon Robotics, not me]

Multi-Agent Path Finding (MAPF)

Robot

Agent

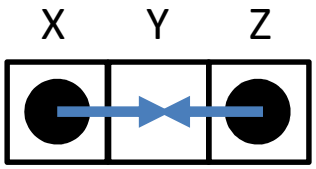


- Simplifying assumptions
  - Point robots
  - No kinematic constraints
  - Discretized environment
    - we use grids here but most techniques work on planar graphs in general

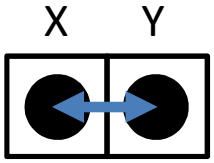
4-neighbor grid

Multi-Agent Path Finding (MAPF)

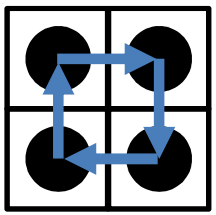
- Each agent moves N, E, S or W into an adjacent unblocked cell
- Not allowed ("vertex collision")
  - Agent 1 moves from X to Y
  - Agent 2 moves from Z to Y



- Not allowed ("edge collision")
  - Agent 1 moves from X to Y
  - Agent 2 moves from Y to X



- Allowed



4-neighbor grid

Multi-Agent Path Finding (MAPF)

- Optimal MAPF algorithms
  - Theorem [Yu and LaValle]: MAPF is NP-hard to solve optimally for makespan or flowtime minimization

[www.random-ideas.net]

- Bounded-suboptimal MAPF algorithms
  - Theorem: MAPF is NP-hard to approximate within any factor less than 4/3 for makespan minimization on graphs in general

Multi-Agent Path Finding (MAPF)

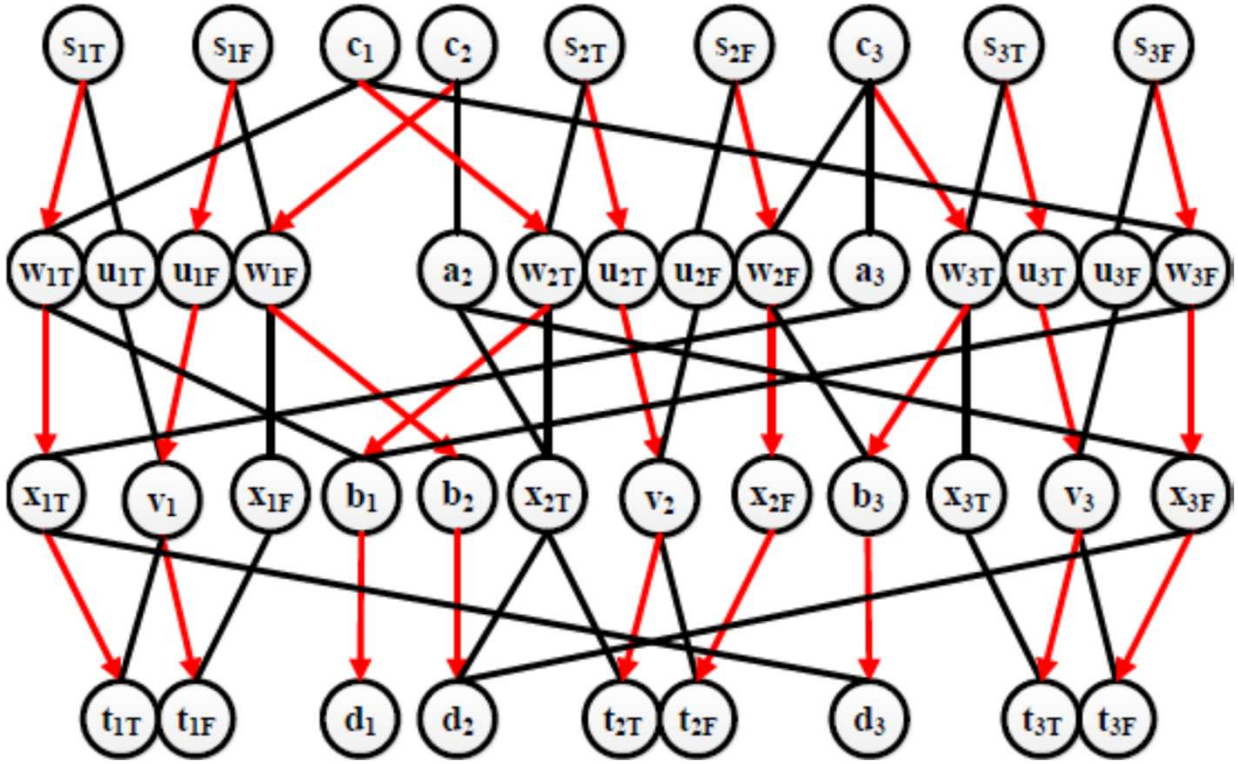
- Reduction from (≤3, =3)-SAT: It is NP-complete to determine whether a given (≤3, =3)-SAT instance is satisfiable

- Each clause contains at most 3 literals
- Each variable appears in exactly 3 clauses
- Each variable appears uncomplemented at least once
- Each variable appears complemented at least once

- Example: $(X_1 \vee X_2 \vee \overline{X_3}) \wedge (\overline{X_1} \vee X_2 \vee \overline{X_3}) \wedge (X_1 \vee \overline{X_2} \vee X_3)$

Multi-Agent Path Finding (MAPF)

- Example: $(X_1 \vee \overline{X_2} \vee X_3) \wedge (\overline{X_1} \vee X_2 \vee \overline{X_3}) \wedge (X_1 \vee \overline{X_2} \vee X_3)$



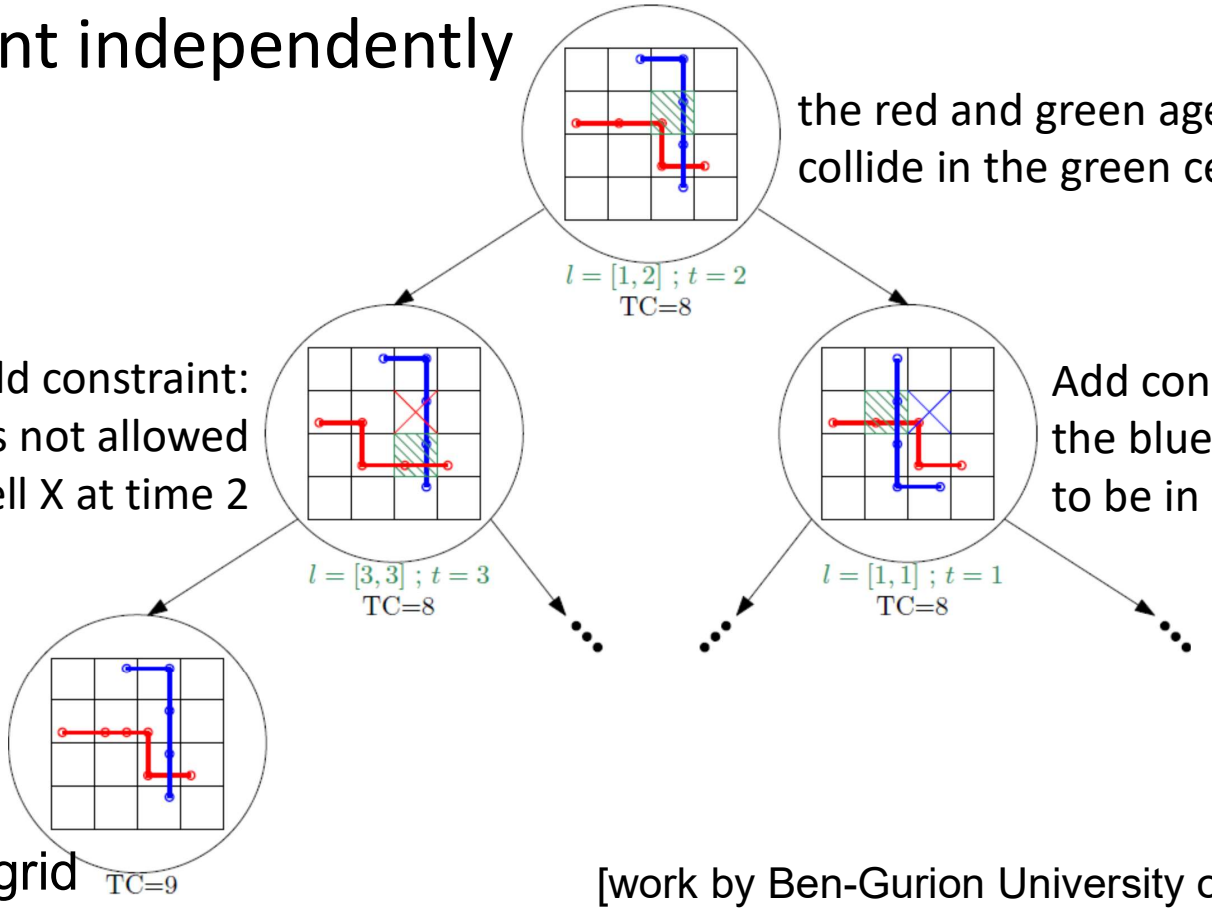
X1 ≡ false X2 ≡ true X3 ≡ true

Multi-Agent Path Finding (MAPF)

- Makespan is 3 if and only if (≤3, =3)-SAT instance is satisfiable
- Makespan is 4 if and only if (≤3, =3)-SAT instance is unsatisfiable
- Any MAPF approximation algorithm with ratio 4/3 – ϵ thus computes a MAPF plan with makespan 3 [and thus solves the SAT instance] if and only if a (≤3, =3)-SAT instance is satisfiable

Conflict-Based Search with Highways

- Conflict-based search [Sharon et al.]: Bounded-suboptimal MAPF solver that plans for each agent independently



4-neighbor grid

[work by Ben-Gurion University of the Negev, not me]

Conflict-Based Search with Highways

- Experience graphs [Phillips et al.]:
  Bounded-suboptimal single-agent path planner so that the resulting path uses edges in a given subgraph (the experience graph) as much as possible

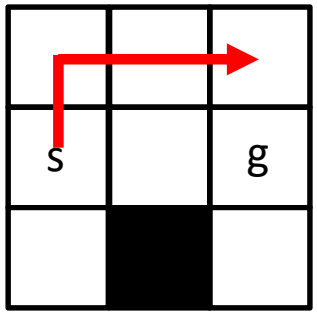
[work by CMU, not me]

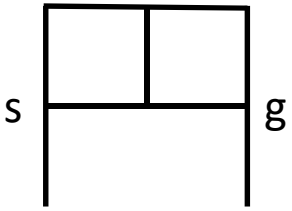
Conflict-Based Search with Highways

optimal | suboptimality bound 4

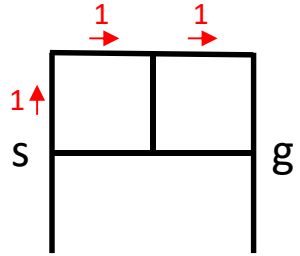
	regular (no highways)	highways #1	highways #2 (experience graphs)

- Graph for A* search

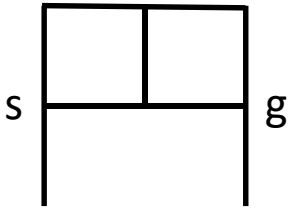




all costs are 1

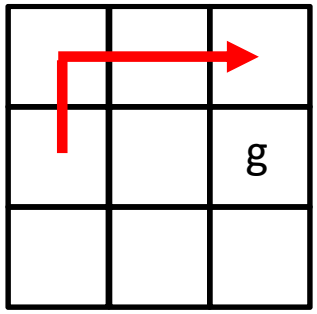


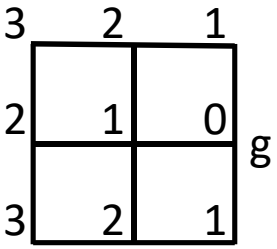
all costs are 4 except for the ones shown



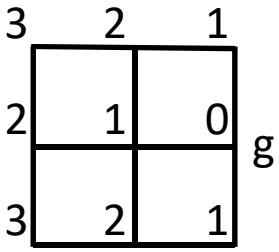
all costs are 1

- Graph relaxation for calculating h-values of A* search

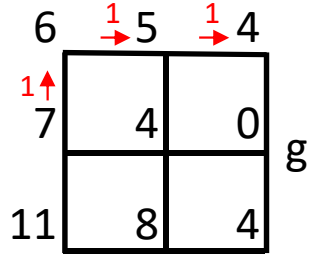




all costs are 1



all costs are 1



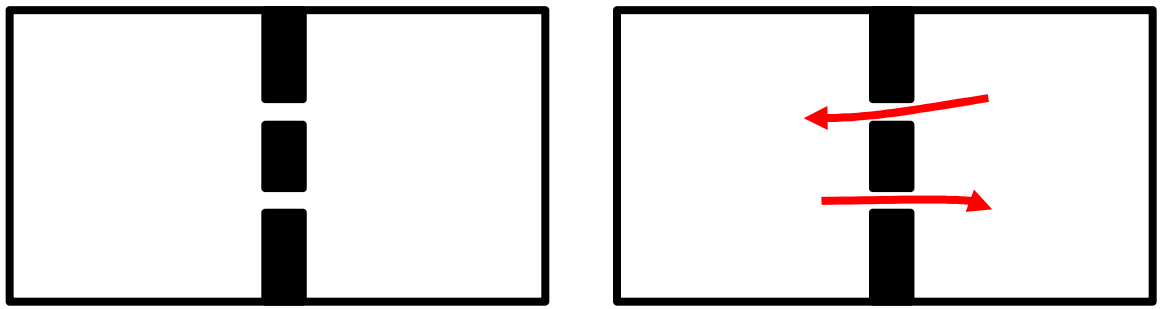
all costs are 4 except for the ones shown

4-neighbor grid

[work by CMU, not me]

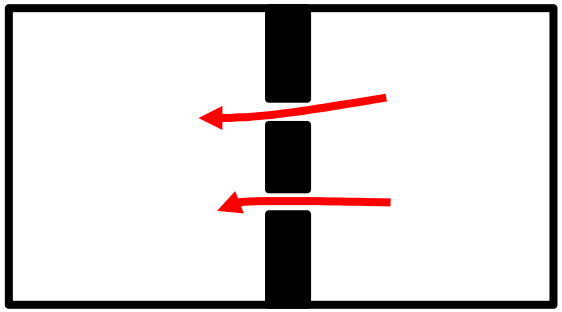
Conflict-Based Search with Highways

- Conflict-based search with highways (ECBS+HWY): Bounded suboptimal MAPF solver
  - Conflict-based search
  - Experience graphs create lanes (called highways) for the agents to avoid head-to-head collisions, which decreases the computation time of conflict-based search



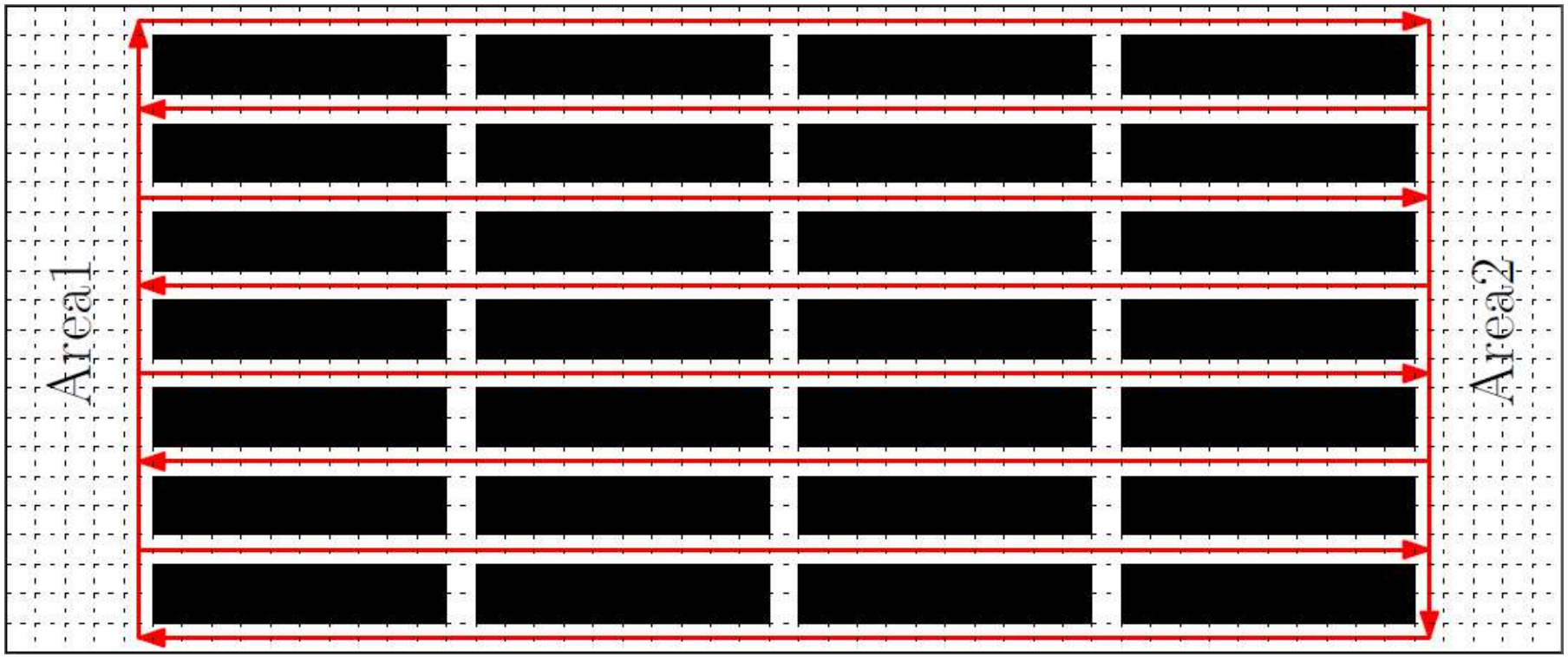
Conflict-Based Search with Highways

- Conflict-based search with highways (ECBS+HWY)
  - Highways provide consistency and thus predictability of agent movement, which might be important for human co-workers
  - Highways do not make MAPF instances unsolvable because they are only used as advice rather than hard constraints



Conflict-Based Search with Highways

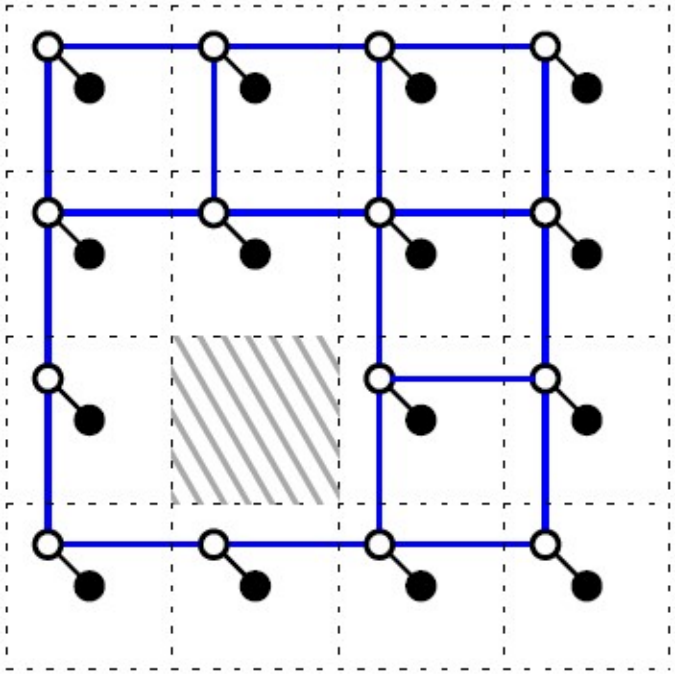
- Conflict-based search with highways (ECBS+HWY)



4-neighbor grid

Conflict-Based Search with Highways

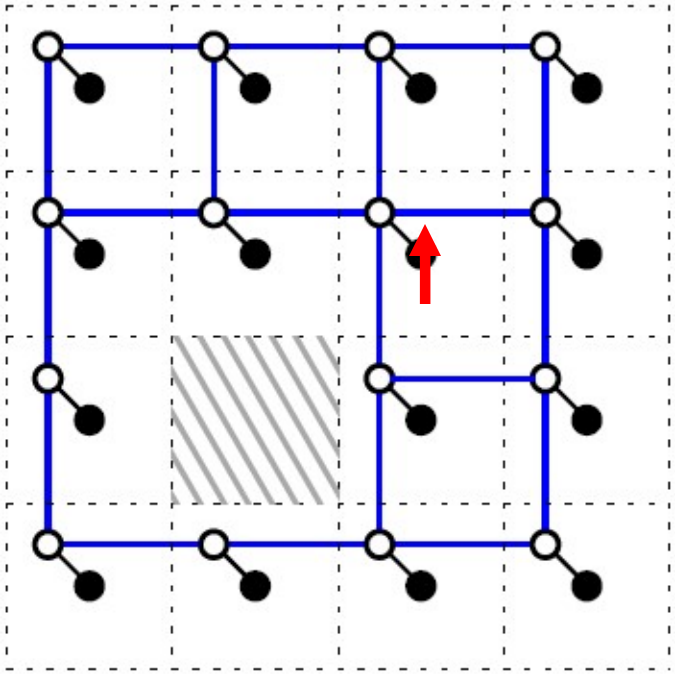
- Learning highways with graphical models
- Plan a shortest path for each agent independently
- Direction vector of a cell: Average of entry and exit directions of each path for the given cell
- Features
  - Collision?
  - Direction of direction vector (N, E, S, W)
  - Magnitude of direction vector > 0.5?



4-neighbor grid

Conflict-Based Search with Highways

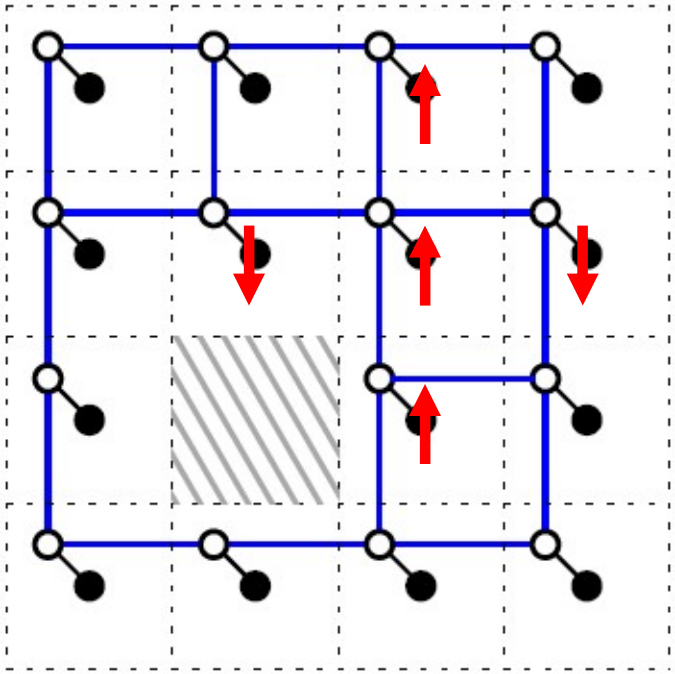
- Learning highways with graphical models
- Plan a shortest path for each agent independently
- Direction vector of a cell: Average of entry and exit directions of each path for the given cell
- Features
  - Collision?
  - Direction of direction vector (N, E, S, W)
  - Magnitude of direction vector > 0.5?



4-neighbor grid

Conflict-Based Search with Highways

- Learning highways with graphical models
- Plan a shortest path for each agent independently
- Direction vector of a cell: Average of entry and exit directions of each path for the given cell
- Features
  - Collision?
  - Direction of direction vector (N, E, S, W)
  - Magnitude of direction vector > 0.5?



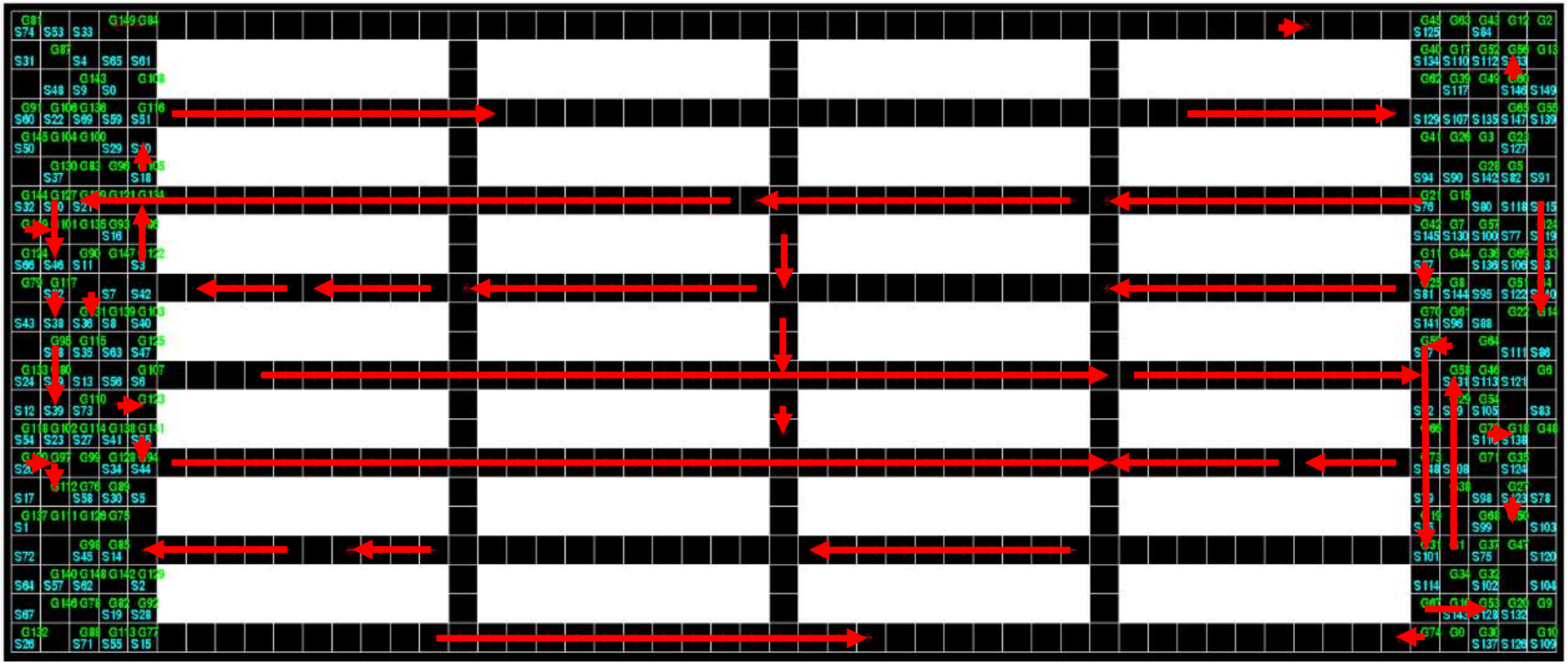
4-neighbor grid

Conflict-Based Search with Highways

- Graphical models encode probabilistic knowledge
  - If agents collide in a cell, make it more likely that there is a highway in that cell
  - If most agents move northward in a cell, make it more likely that a highway in that cell, if any, is a northward one
  - If a northward highway is in a cell, make it more likely that highways in its northern and southern neighbors, if any, are also northward ones (to form a longer lane)
  - If a northward highway is in a cell, make it more likely that highways in its western and eastern neighbors, if any, are southward ones (to form adjacent lanes in opposite directions)

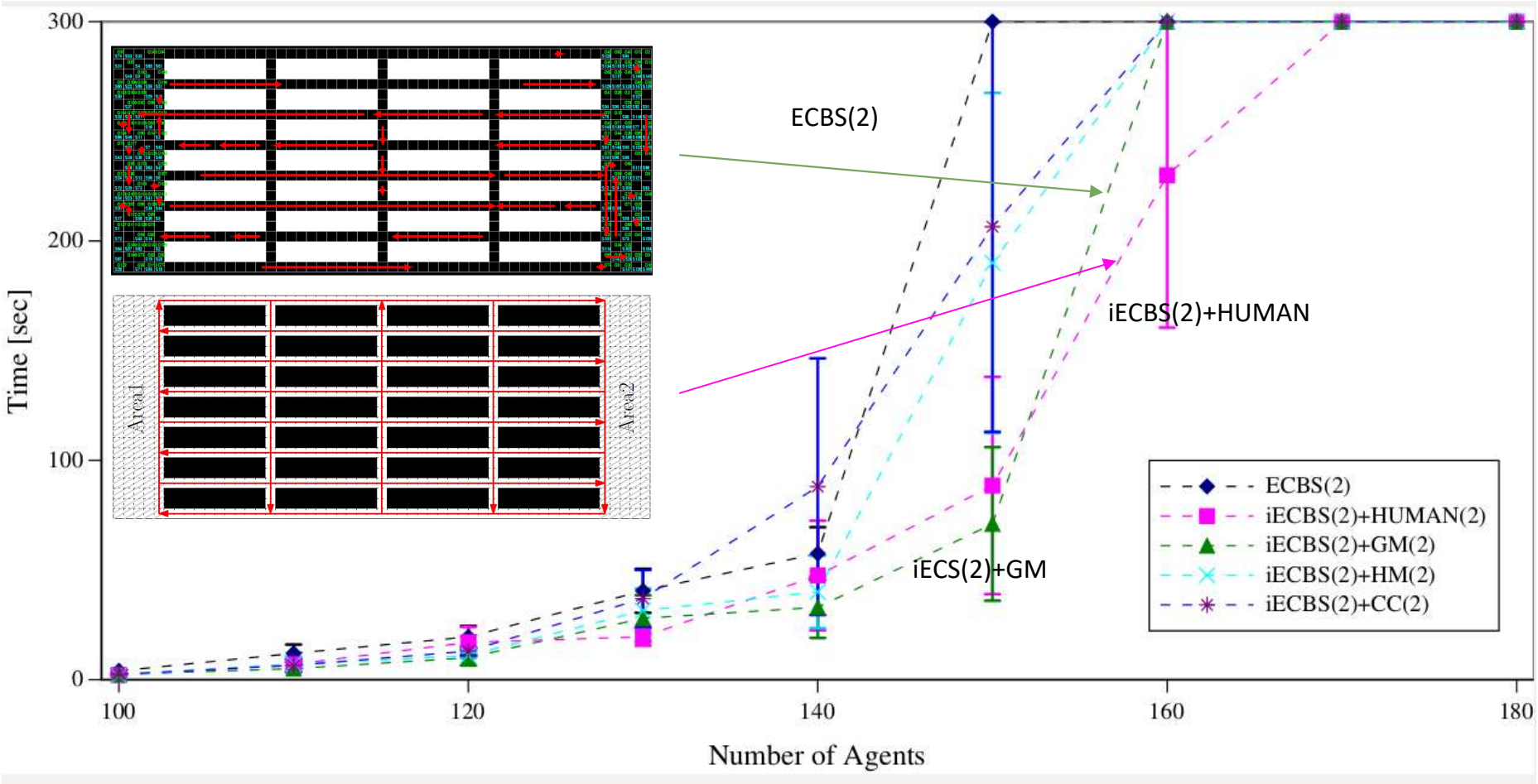
4-neighbor grid

Conflict-Based Search with Highways



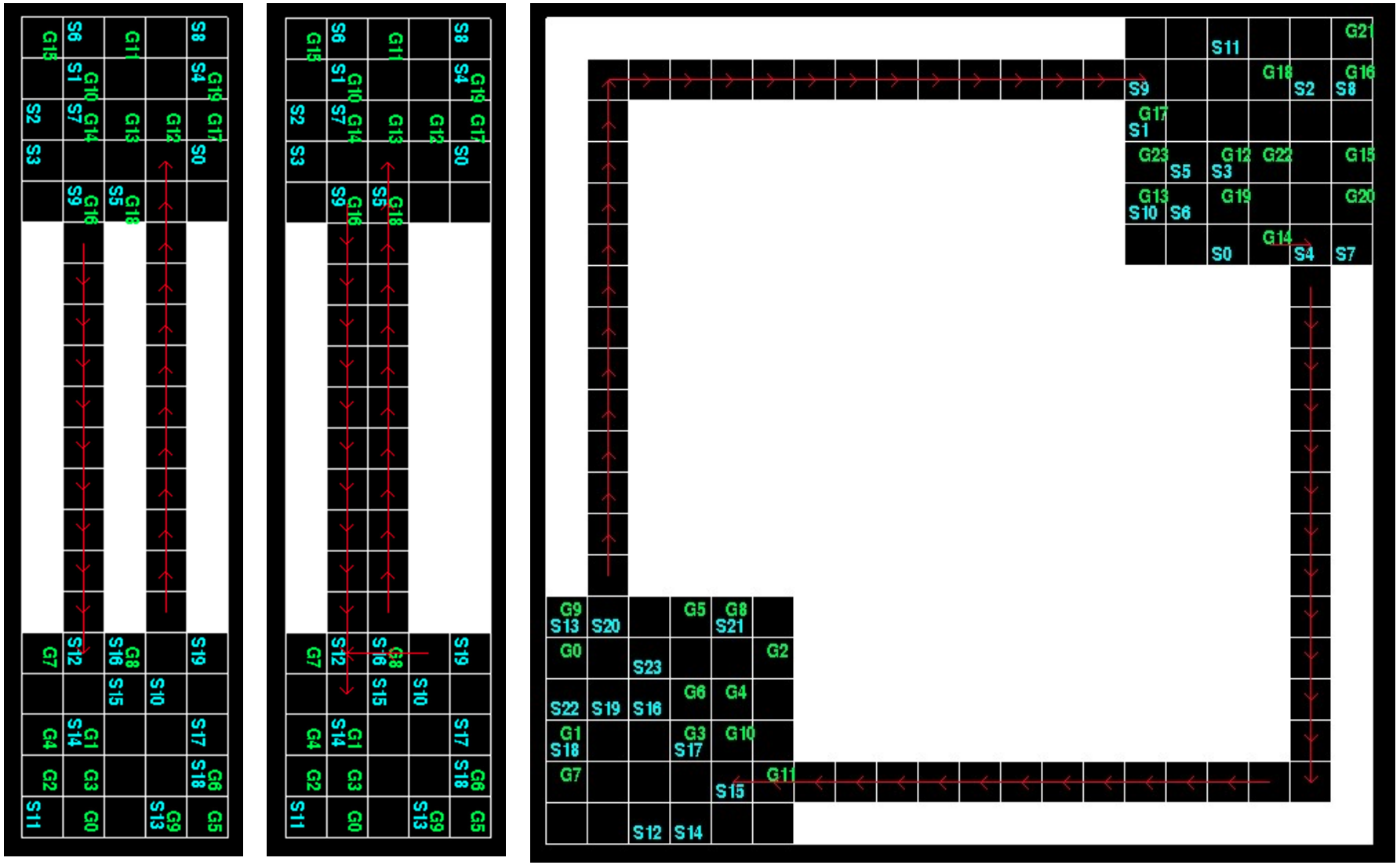
4-neighbor grid

Conflict-Based Search with Highways



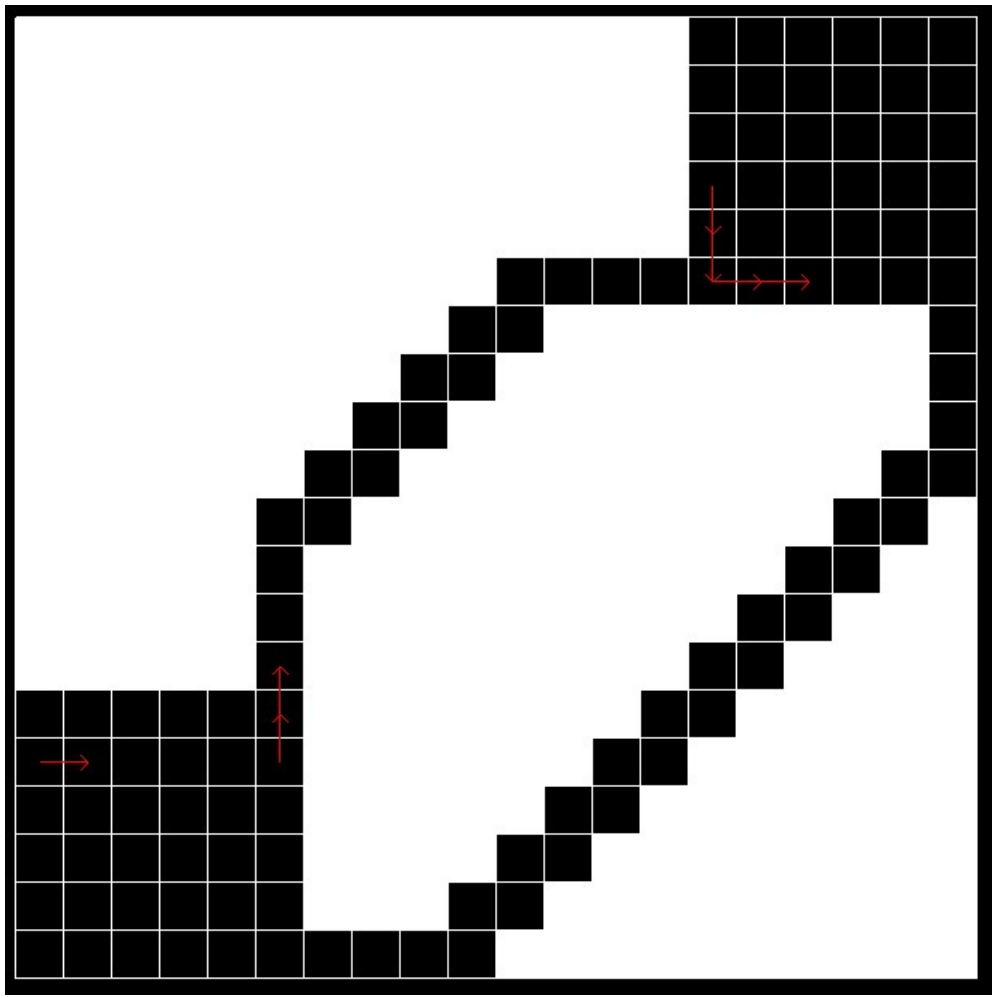
4-neighbor grid

Conflict-Based Search with Highways



4-neighbor grid

Conflict-Based Search with Highways



4-neighbor grid

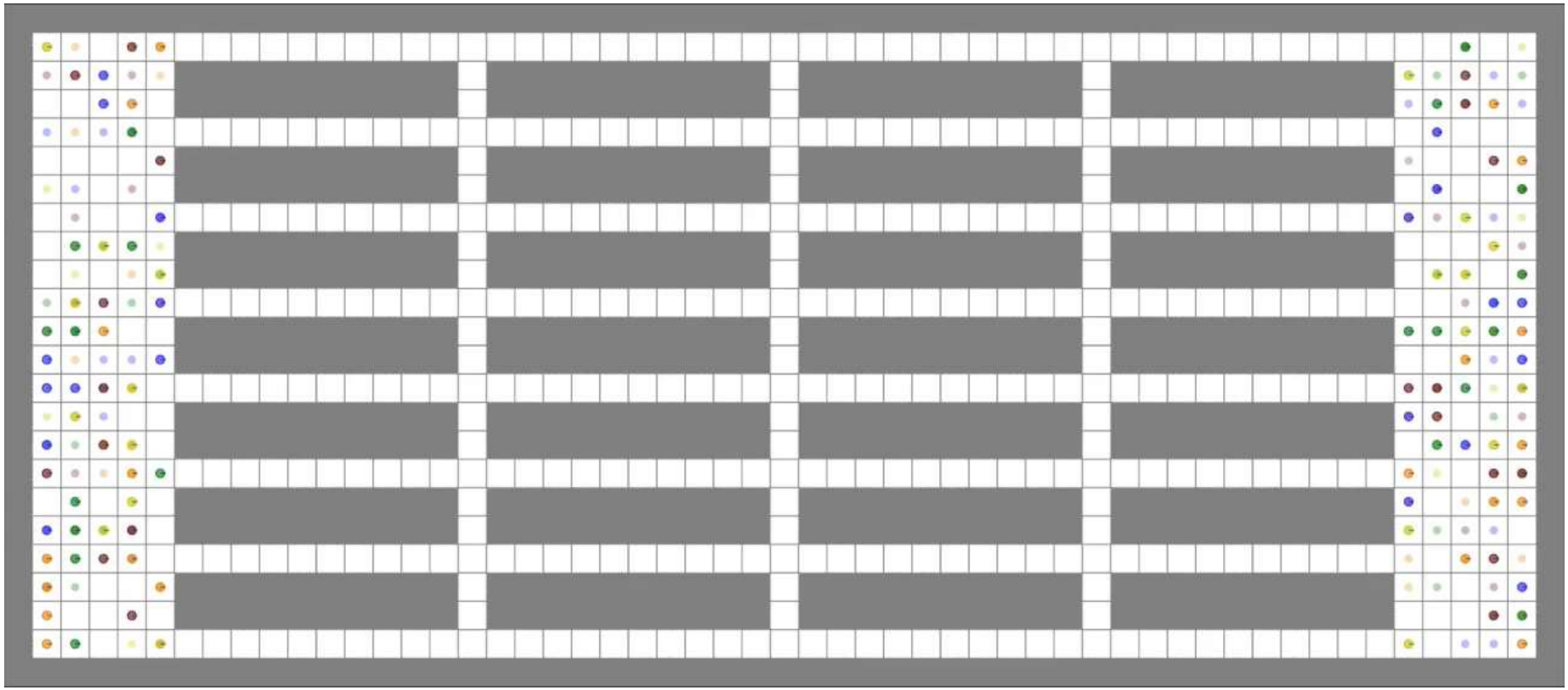
Conflict-Based Search with Highways

- Rapid random restarts help to solve more multi-agent path finding problems within a given runtime limit.
- Here: We randomize the ordering in which the agents plan their paths in the high-level root node.

runs	time limit	38 “easy”	12 “hard”	50 total
1	300 sec	100.00%	0.00%	76.00%
3	100 sec	97.65%	96.87%	97.60%
5	60 sec	98.57%	98.81%	98.70%

Conflict-Based Search with Highways

- Conflict-based search with highways (ECBS+HWY)

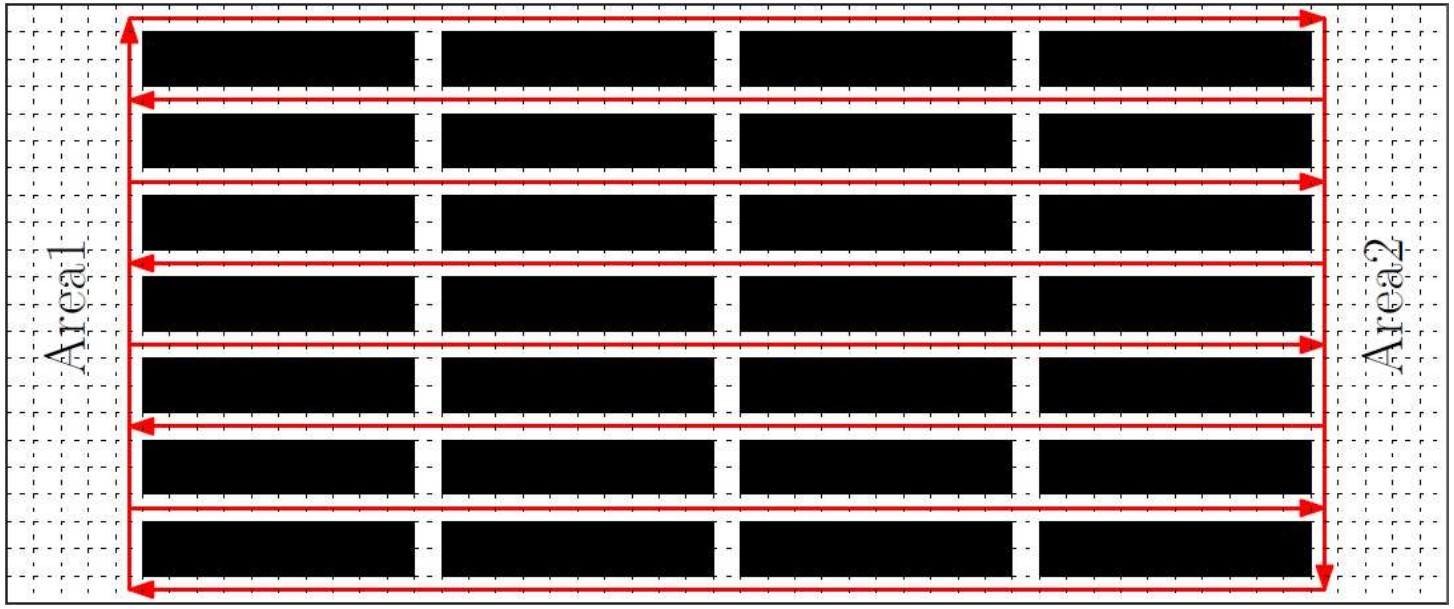


4-neighbor grid

8x

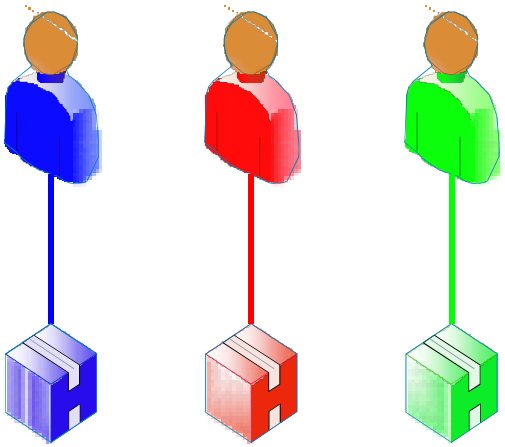
Conflict-Based Search with Highways

- 130 agents (half moving right, half moving left)
- Minimize flowtime with suboptimality bound 2



- Conflict-based search: 48.5 seconds
- Conflict-based search with highways: 29.1 seconds

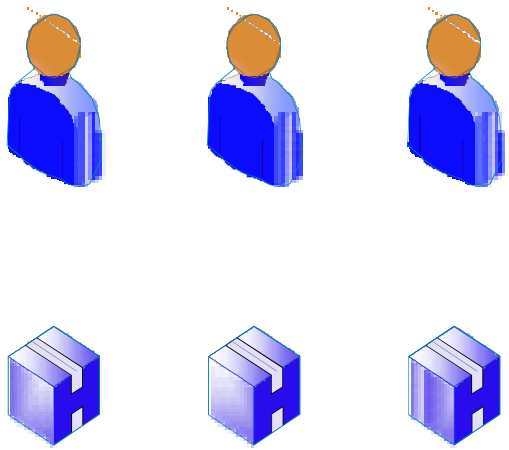
Extensions



non-anonymous MAPF

NP-hard

solved with A* approaches
e.g. conflict-based search or M*



anonymous MAPF

polynomial-time solvable for
makespan minimization
solved with flow approaches
e.g. max-flow algorithm

Anonymous MAPF

- (Non-anonymous) MAPF
  - Given: a number of agents (each with a start and goal location) and a known environment
  - Task: find collision-free paths for the agents from their start to their goal locations that minimize makespan or flowtime
- Anonymous MAPF
  - Given: a number of agents (each with a start location), an equal number of goal locations, and a known environment
  - Task: assign a different goal location to each agent and then find collision-free paths for the agents from their start to their goal locations that minimize makespan or flowtime

Anonymous MAPF

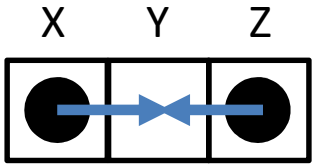
- Theorem [Yu and Lavalle]: An anonymous MAPF instance admits a MAPF plan with makespan at most T if and only if the time-expanded network with T periods admits a max flow of the number of agents.

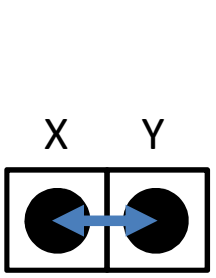
[work by the University of Illinois at Urbana-Champaign, not me]

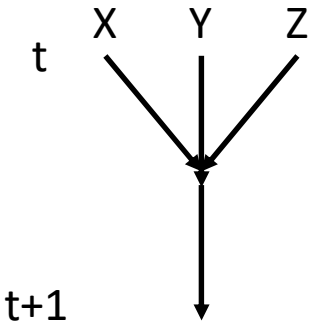
Anonymous MAPF

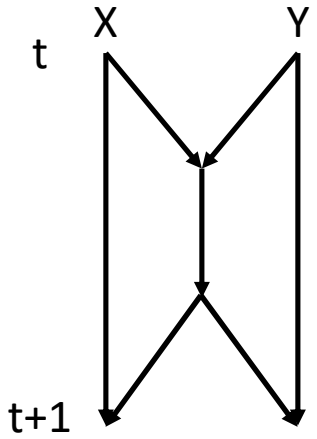
all edges have capacity one

- Each agent moves N, E, S or W into an adjacent unblocked cell
- Not allowed ("vertex collision")
  - Agent 1 moves from X to Y
  - Agent 2 moves from Z to Y
- Not allowed ("edge collision")
  - Agent 1 moves from X to Y
  - Agent 2 moves from Y to X



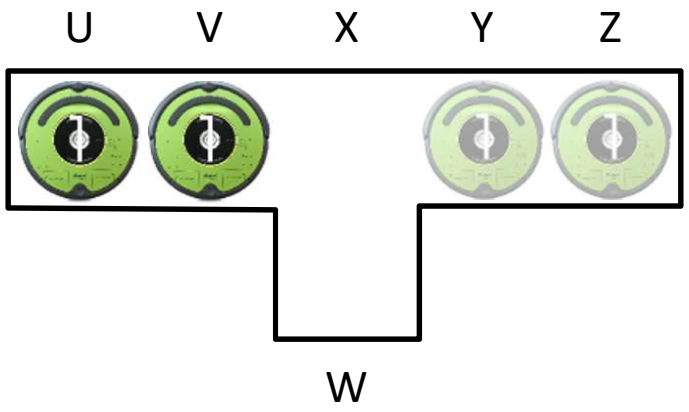


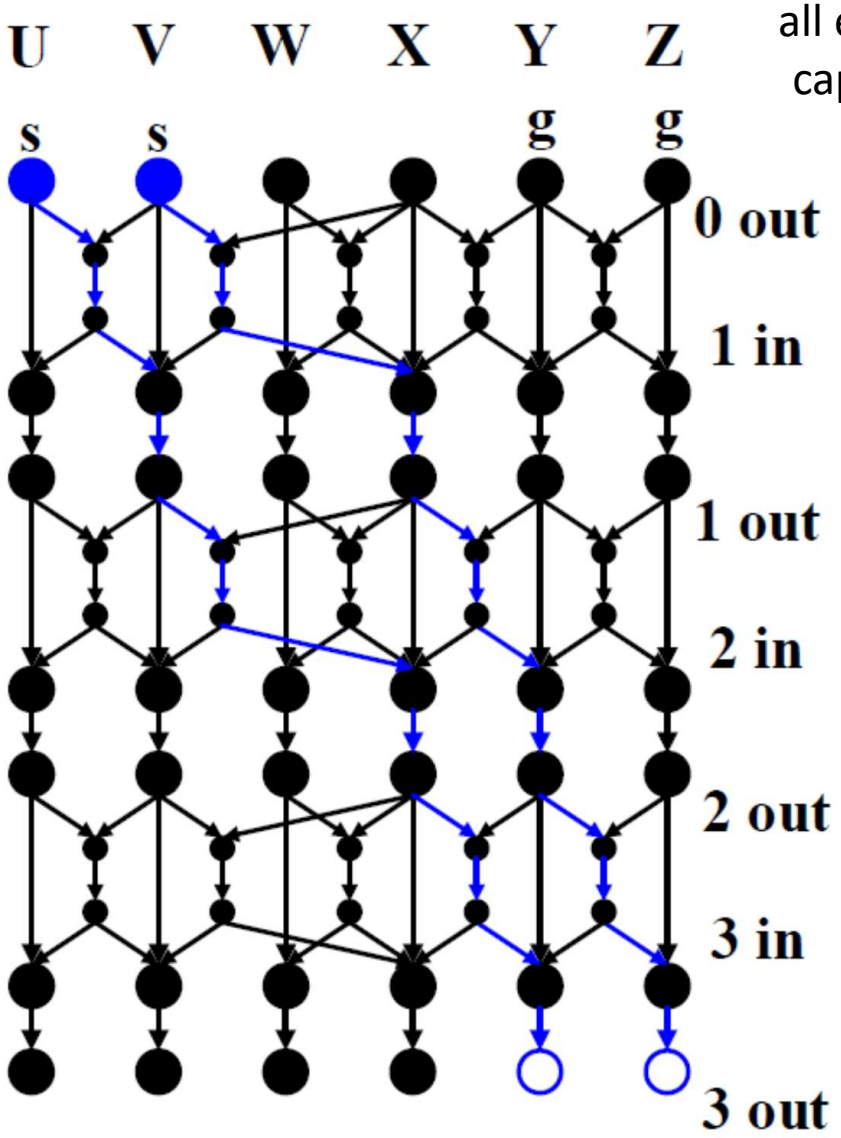




[work by the University of Illinois at Urbana-Champaign, not me]

Anonymous MAPF

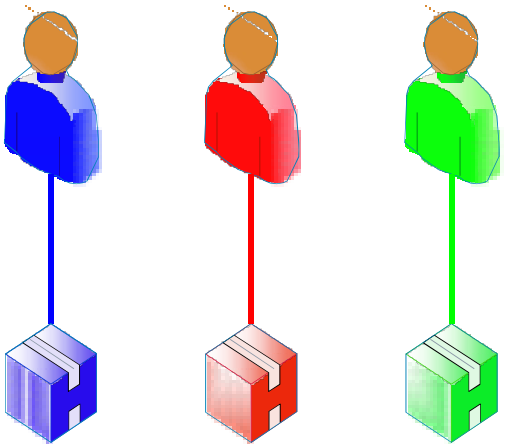




all edges have capacity one

[work by the University of Illinois at Urbana-Champaign, not me]

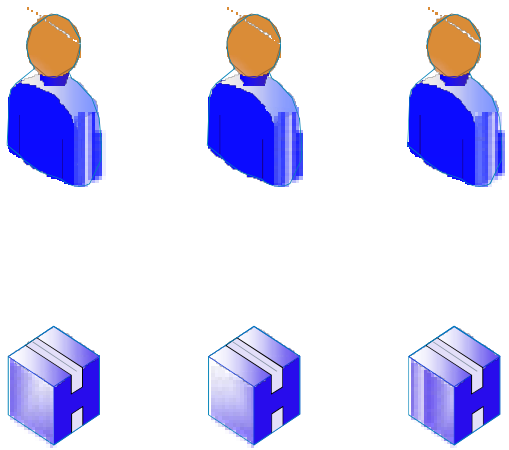
Target Assignment and Path Finding (TAPF)



non-anonymous MAPF

NP-hard

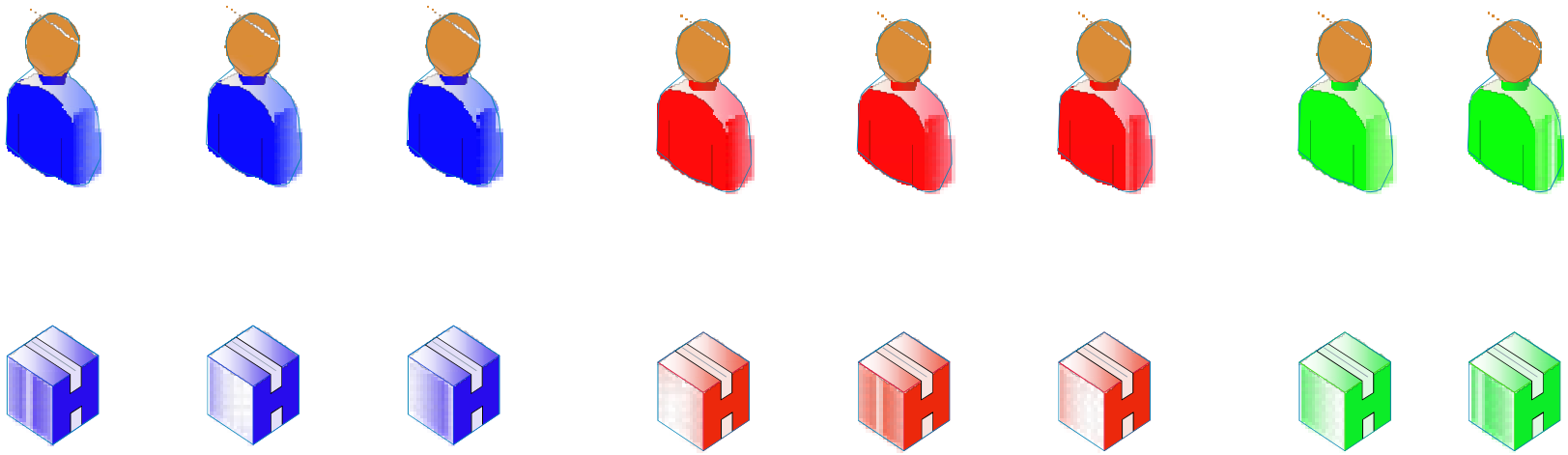
solved with A* approaches
e.g. conflict-based search or M*



anonymous MAPF

polynomial-time solvable for
makespan minimization
solved with flow approaches
e.g. max-flow algorithm

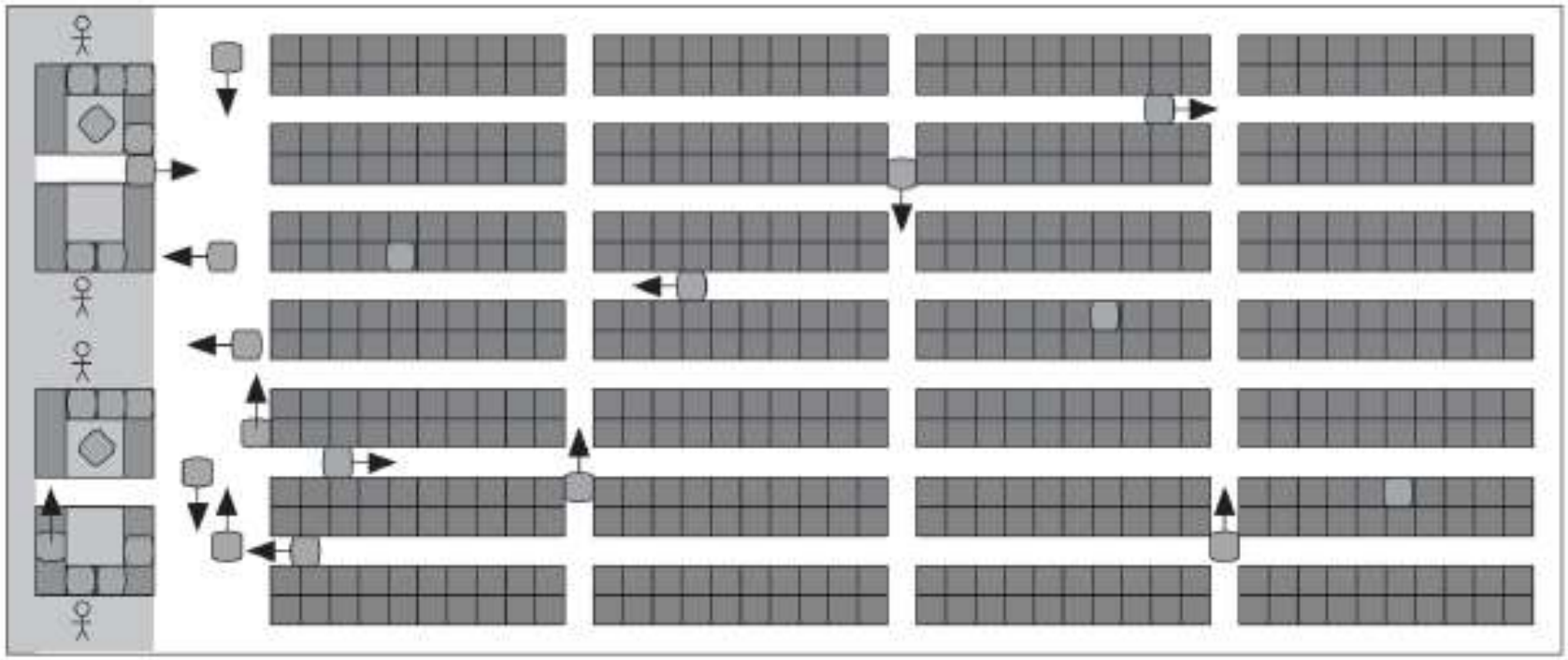
Target Assignment and Path Finding (TAPF)



mix of non-anonymous and anonymous MAPF

Target Assignment and Path Finding (TAPF)
with k groups (here: 3), also called types

Target Assignment and Path Finding (TAPF)



[Wurman, D'Andrea and Mountz]

Group 0: Agents that move from the packing stations to the storage locations
Group 1: Agents that move from the storage locations to Packing Station 1
Group 2: Agents that move from the storage locations to Packing Station 2
Group 3: Agents that move from the storage locations to Packing Station 3

Target Assignment and Path Finding (TAPF)

- Theorem: TAPF (with k>1 groups) is NP-hard to solve optimally for makespan or flowtime minimization

- Theorem: TAPF (with k>1 groups) is NP-hard to approximate within any factor less than 4/3 for makespan minimization on graphs in general

Target Assignment and Path Finding (TAPF)

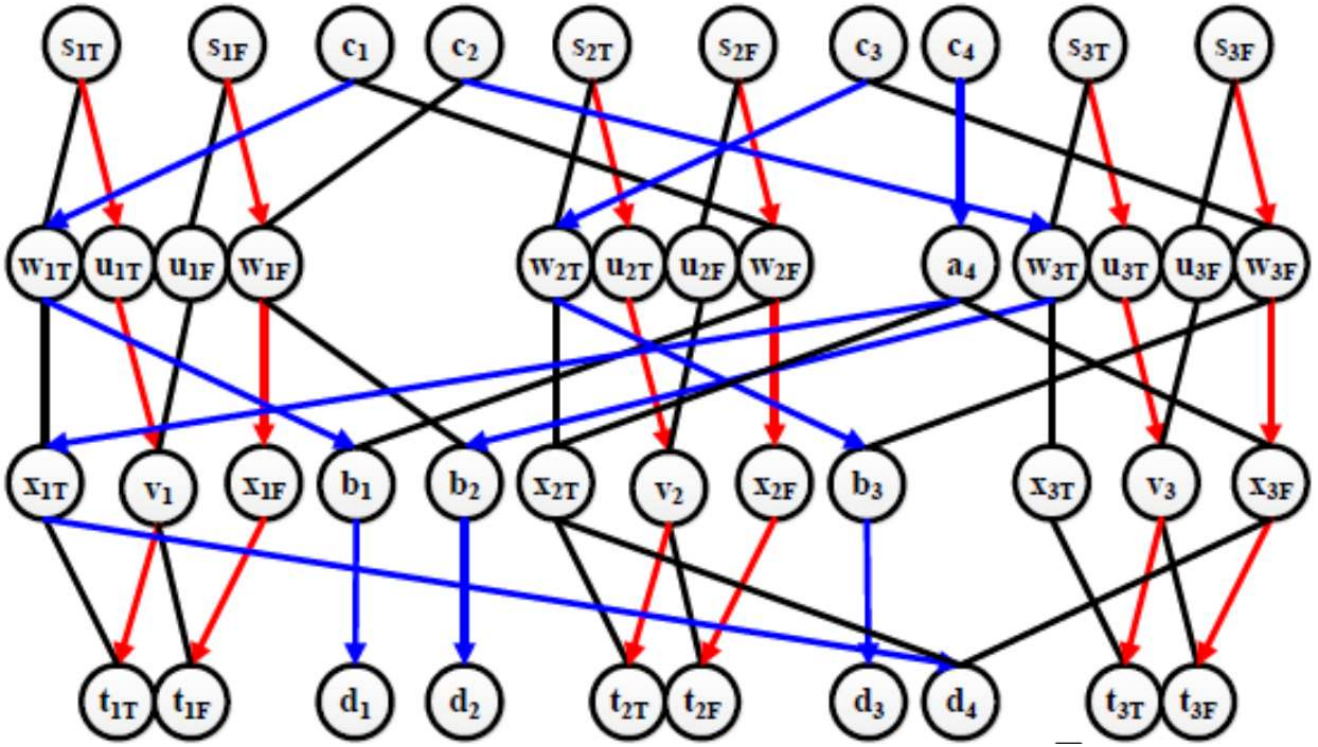
- Reduction from $2/\overline{2}/3$-SAT: It is NP-complete to determine whether a given $2/\overline{2}/3$-SAT instance is satisfiable

- Each variable appears in exactly 3 clauses
- Each variable appears uncomplemented in a clause of size two
- Each variable appears complemented in a clause of size two
- Each variable appears in a clause of size three

- Example: $(X_1 \vee \overline{X}_2) \wedge (\overline{X}_1 \vee X_3) \wedge (X_2 \vee \overline{X}_3) \wedge (X_1 \vee X_2 \vee \overline{X}_3)$

Target Assignment and Path Finding (TAPF)

- Example: $(X_1 \vee \overline{X}_2) \wedge (\overline{X}_1 \vee X_3) \wedge (X_2 \vee \overline{X}_3) \wedge (X_1 \vee X_2 \vee \overline{X}_3)$



Target Assignment and Path Finding (TAPF)

- CBM combines the max-flow algorithm and conflict-based search to minimize makespan for TAPF instances
  - CBM uses the max-flow algorithm to assign goal locations and plan paths for all agents in a group (to solve the corresponding anonymous MAPF instance)
    CBM actually uses a min-cost max-flow algorithm since is important to choose paths that result in few collisions with agents from other groups
  - CBM treats each group as a meta-agent and uses conflict-based search to plan sets of paths for all meta-agents (to solve the corresponding non-anonymous MAPF problem)

Target Assignment and Path Finding (TAPF)

- Theorem: CBM is complete and optimal for minimizing makespan for TAPF instances
- Experimental results

	CBM		Mixed Integer Program	
Agents	Time	Success	Time	Success
10	0.34	100%	18.24	100%
20	0.78	100%	62.85	94%
30	1.71	100%	108.75	66%
40	2.95	100%	152.98	14%
50	5.32	100%	161.95	4%

30x30 4-neighbor grids with 10% randomly blocked cells and a 5-minute time limit

Execution of MAPF Plans

- Planning uses models that are not completely accurate
  - Robots are not completely synchronized
  - Robots do not move exactly at the nominal speed
  - Robots have unmodeled kinematic constraints
  - …
- Plan execution will therefore likely deviate from the plan
- Replanning whenever plan execution deviates from the plan is intractable since it is NP-hard to find good plans

Plan Execution

- Main loop
  - Run Conflict-Based Search with Highways to find paths (slow)

Plan Execution

- Main loop
  - Run Conflict-Based Search with Highways to find paths (slow)
  - Construct a simple temporal network for the paths

Plan Execution

- Main loop
  - Run Conflict-Based Search with Highways to find paths (slow)
  - Construct a simple temporal network for the paths
  - Determine the earliest arrival times in the nodes (fast)

Plan Execution

- Main loop
  - Run Conflict-Based Search with Highways to find paths (slow)
  - Construct a simple temporal network for the paths
  - Determine the earliest arrival times in the nodes (fast)

  - Calculate speeds for the robots from the earliest arrival times

Plan Execution

- Main loop
  - Run Conflict-Based Search with Highways to find paths (slow)
  - Construct a simple temporal network for the paths
  - Determine the earliest arrival times in the nodes (fast)

  - Calculate speeds for the robots from the earliest arrival times
  - Move robots along their paths with these speeds

Plan Execution

- Main loop
  - Run Conflict-Based Search with Highways to find paths (slow)
  - Construct a simple temporal network for the paths
  - Determine the earliest arrival times in the nodes (fast)
  - Calculate speeds for the robots from the earliest arrival times
  - Move robots along their paths with these speeds
  - If plan execution deviates from the plan, then

Plan Execution

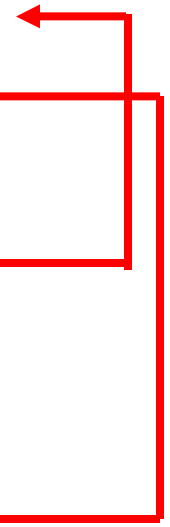
- Main loop
  - Run Conflict-Based Search with Highways to find paths (slow)
  - Construct a simple temporal network for the paths
  - Determine the earliest arrival times in the nodes (fast)
  - Calculate speeds for the robots from the earliest arrival times
  - Move robots along their paths with these speeds
  - If plan execution deviates from the plan, then

Plan Execution

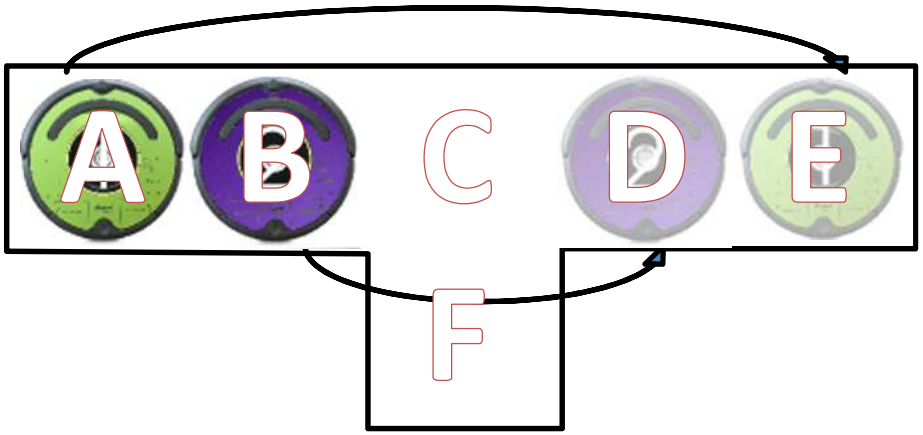
- Main loop
  - Run Conflict-Based Search with Highways to find paths (slow)
  - Construct a simple temporal network for the paths
  - Determine the earliest arrival times in the nodes (fast)
  - Calculate speeds for the robots from the earliest arrival times
  - Move robots along their paths with these speeds
  - If plan execution deviates from the plan, then

Plan Execution

- Main loop
  - Run Conflict-Based Search with Highways to find paths
  - Construct a simple temporal network for the paths
  - Determine the earliest arrival times in the nodes (fast)
  - If they do not exist, then
  - Calculate speeds for the robots from the earliest arrival times
  - Move robots along their paths with these speeds
  - If plan execution deviates from the plan, then



Execution of MAPF Plans

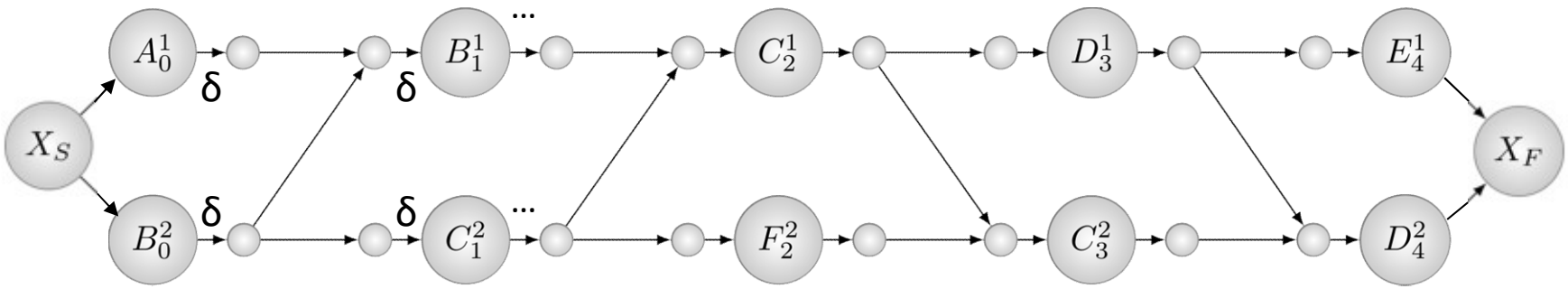


Agent 1 $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$

Agent 2 $B \rightarrow C \rightarrow F \rightarrow C \rightarrow D$

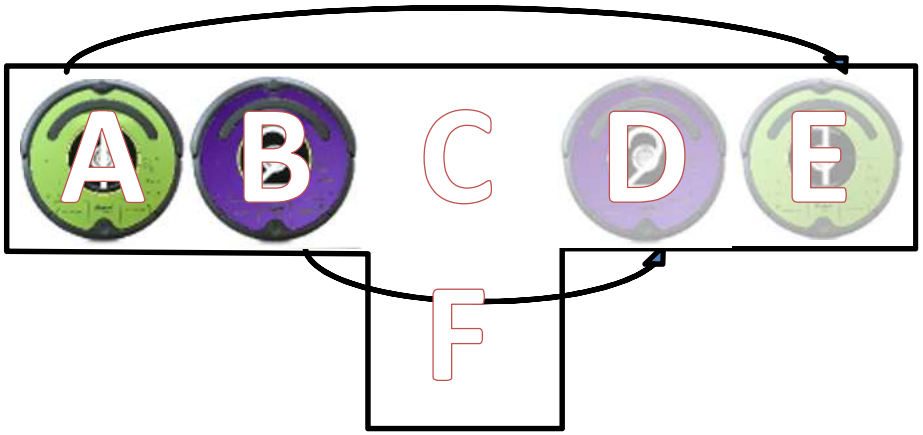
Precedence Graph

vertex = event that an agent arrives at a location



4-neighbor grid

Execution of MAPF Plans

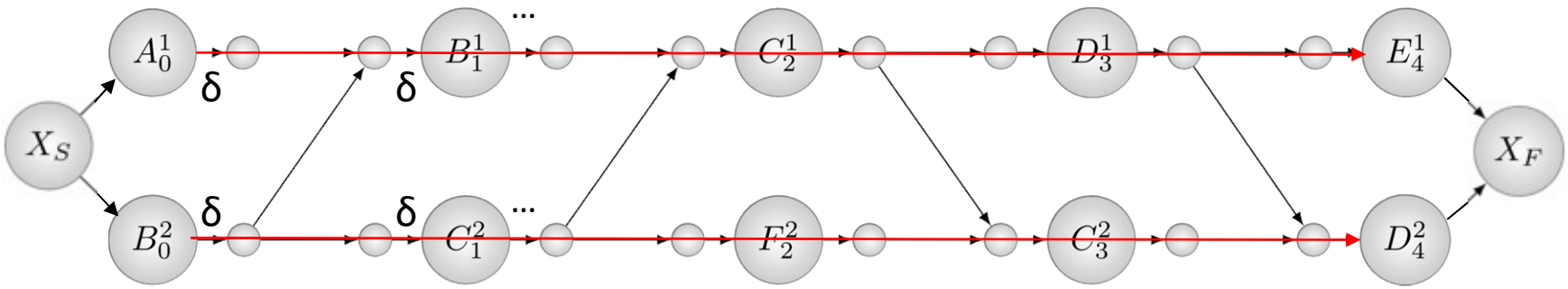


Agent 1 A → B → C → D → E

Agent 2 B → C → F → C → D

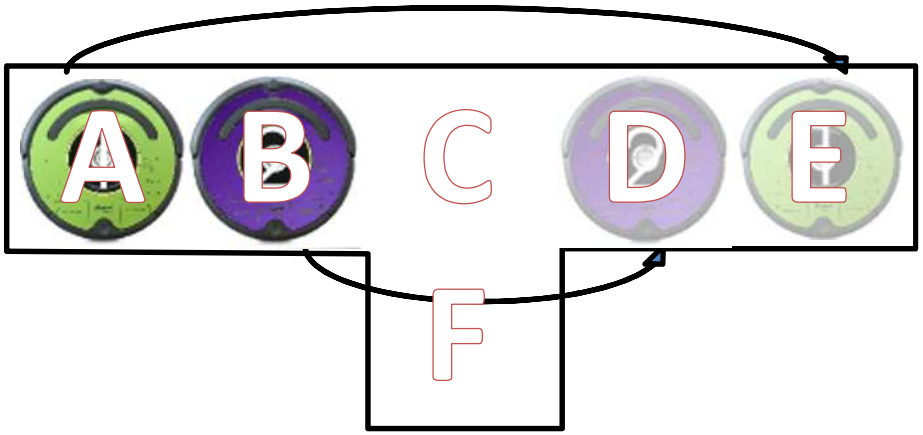
Precedence Graph

Type 1 edge = order in which the same agent arrives at locations



4-neighbor grid

Execution of MAPF Plans

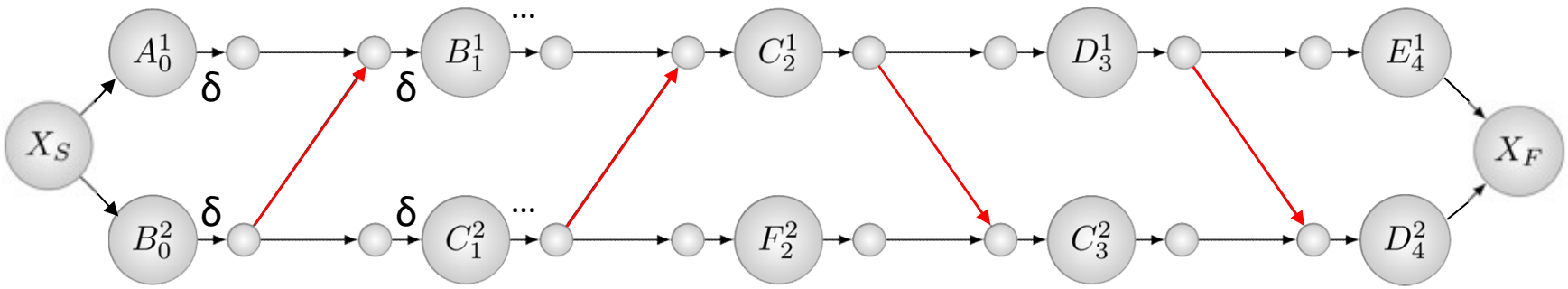


Agent 1 A → B → C → D → E

Agent 2 B → C → F → C → D

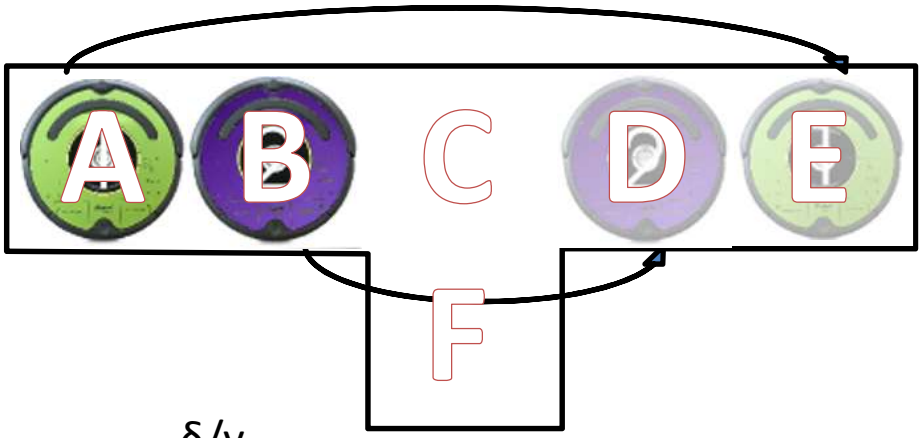
Precedence Graph

Type 2 edge = order in which two different agents arrive at the same location



4-neighbor grid

Execution of MAPF Plans

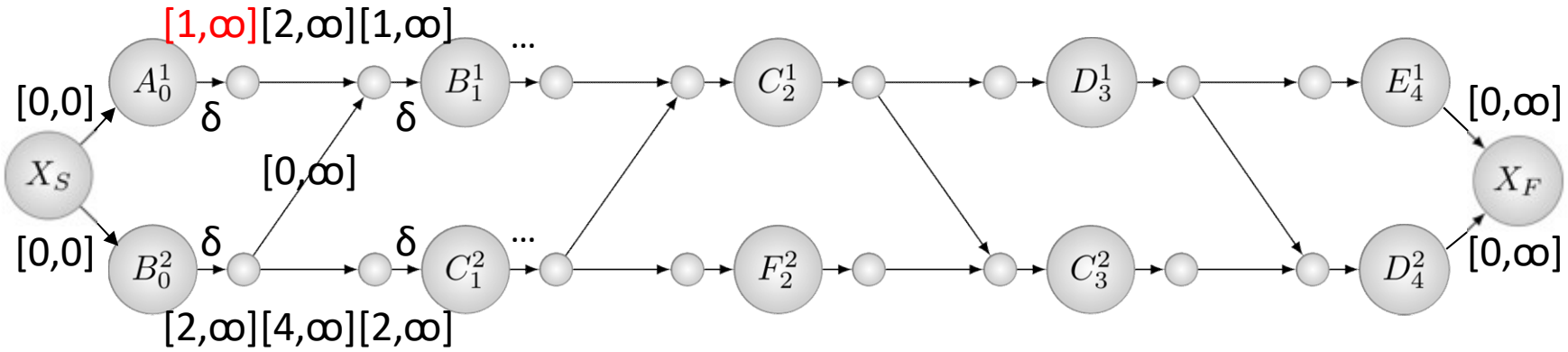


Agent 1 $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$

Agent 2 $B \rightarrow C \rightarrow F \rightarrow C \rightarrow D$

δ/v_{max}

Simple Temporal Network [Dechter et al.]



4-neighbor grid

Execution of MAPF Plans

- Minimize makespan and flowtime
  - Schedule each arrival in a location as early as allowed by the constraints

Minimize $\sum_{j=1}^{K} t(v^j)$
such that $t(X_S) = 0$
and, for all $e = (v, v') \in \mathcal{E}'$,
$t(v') - t(v) \geq LB(e)$
$t(v') - t(v) \leq UB(e)$

polynomial time

Execution of MAPF Plans

- Maximize safety distance
  - Assume that each agent moves with a constant velocity of at most v_{min} along each of its edges
  - Then, the safety distance is $2\delta v_{min}/v_{max}$

Maximize v^*_{min}
such that $t(X_S) = 0$
and, for all $e = (v, v') \in \mathcal{E}'$,
$t(v') - t(v) \geq LB(e)$
$t(v') - t(v) \leq UB(e)$
$t(v') - t(v) \leq l(e)(v^*_{\min})^{-1}$ if e is a Type 1 edge

polynomial time

Execution of MAPF Plans

- MAPF solver: ECBS+HWY
- MAPF-POST: C++, boost graph library, Gurobi LP solver
- PC: i7-4600U 2.1 GHz, 12 GB RAM
- Terrain: 4x3 gridworld with $1m^2$ cells and $\delta = 0.4m$
- Architecture: ROS with decentralized execution
  - Robot controller with state [x,y,Θ] (attempts to meet deadline)
  - PID controller (corrects for heading error and drift)
- Robot simulator: V-REP
- Robots: iRobot Create2 robots

- Test environment: VICON MX Motion Capture System

Execution of MAPF Plans

4-neighbor grid

8x

Challenge: Planning for the TERMES Robots

- Write a planner for the Harvard TERMES robots
  - We have written one - you might have much better ideas!
  - Send me an email for the description of the planning problem

[work by Harvard University, not me]

Acknowledgments

- This talk reported on joint work with a large number of collaborators and students from the University of Southern California and elsewhere. We would like to acknowledge their contributions.
- Special thanks to K. Arras, N. Ayanian, T. Cai, L. Cohen, K. Daniel, A. Felner, C. Hernandez, W. Hoenig, T. K. S. Kumar, M. Likhachev, H. Ma, P. Meseguer, A. Nash, L. Palmieri, G. Sharon, X. Sun, C. Tovey, T. Uras, H. Xu, W. Yeoh, S. Young and D. Zhang.
- Thanks to ARO, IBM, JPL, NSF and ONR for funding.

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.

Acknowledgments

- Please visit idm-lab.org/projects.html for more information, pointers to the literature and our publications.
- If you have any interesting ideas, please send me an email: skoenig@usc.edu.