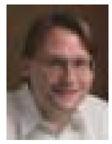
Advances in Path Planning

Sven Koenig
University of Southern California
skoenig@usc.edu



Warning!

- We try to make everything easy to understand.
- We often do not mention crucial details.
- We use both 4- and 8-neighbor grids.
- Values in cells are h-values unless stated otherwise.

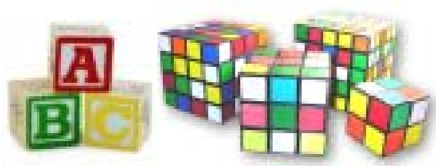
Table of Contents

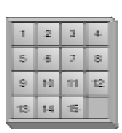
Table of Contents

AI Benchmarks

Standard Search Problems in Artificial Intelligence

- States are given and discrete
- Off-line search: one can concentrate on planning (execution follows)
- Real-time constraints do not exist
- Search space does not fit into memory
- How to search larger and larger search spaces?
- Use big-O time and space analysis





[from Wikipedia]

AI Benchmarks

Path-Planning Problems for Agents

- States are not given, continuous and often hard to characterize
- On-line search: planning and execution have to be interleaved
- Real-time constraints exist
- Search space might or might not fit into memory
- How to search faster and faster?
- Cannot use big-O time and space analysis
- Hardware and implementation details matter

Robotics [from JPL]

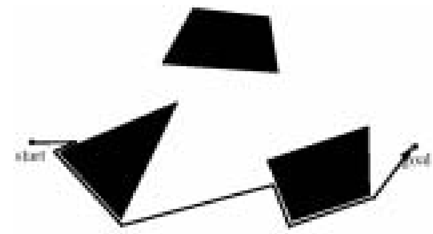
Games [from Cavedog]

Table of Contents

- Overview of path planning
  - Path planning vs AI benchmarks
  - Alternatives to path planning
  - Search spaces and their discretization
  - Searching the search space with A*
- Any-angle path planning with A*
- Speeding up Path Planning with A*

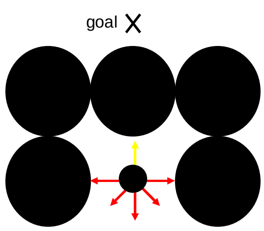
Alternatives to Path Planning

- Bug Algorithms [Lumelsky and Stepanov, 1987]



Alternatives to Path Planning

- Behavior-based methods [Arkin, 1987]



Alternatives to Path Planning

- Properties
  - \+ fast
  - \+ need only local terrain information
  - \- do not necessarily find short paths to the goal
  - \- might not find paths to the goal at all

Table of Contents

- Overview of path planning
  - Path planning vs AI benchmarks
  - Alternatives to path planning
  - Search spaces and their discretization
  - Searching the search space with A*
- Any-angle path planning with A*
- Speeding up Path Planning with A*

Work vs Configuration Space

Path Planning Problems for Agents

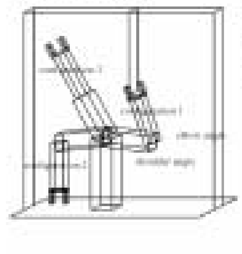
- States are not given, continuous and often hard to characterize
- On-line search: planning and execution have to be interleaved
- Real-time constraints exist
- Search space might or might not fit into memory
- How to search faster and faster?

Games [from Cavedog Entertainment]

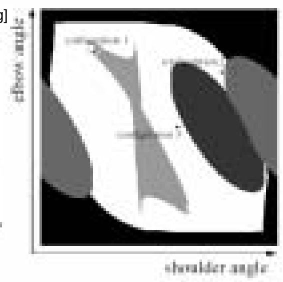
Robotics [from JPL]

Work vs Configuration Space

[from Stuart Russell and Peter Norvig]



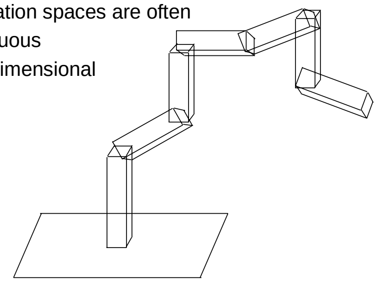
work space



configuration space

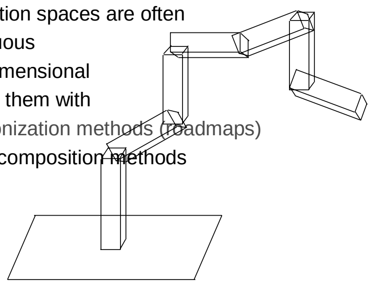
Work vs Configuration Space

- Configuration spaces are often
  - □ continuous
  - □ high-dimensional



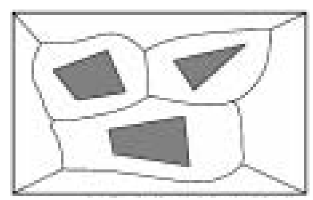
Work vs Configuration Space

- Configuration spaces are often
  - □ continuous
  - □ high-dimensional
- Discretize them with
  - □ skeletonization methods (roadmaps)
  - □ cell-decomposition methods



Discretizing Configuration Space

- Skeletonization methods

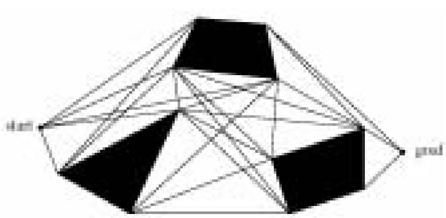


[from Stuart Russell and Peter Norvig – the figure has slight problems]

Voronoi graph

Discretizing Configuration Space

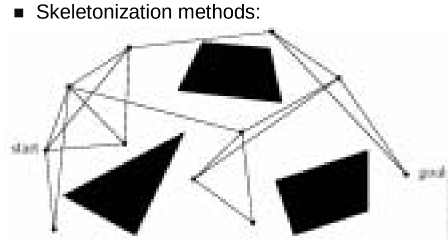
- Skeletonization methods

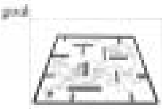


visibility graph

Discretizing Configuration Space

- Skeletonization methods:

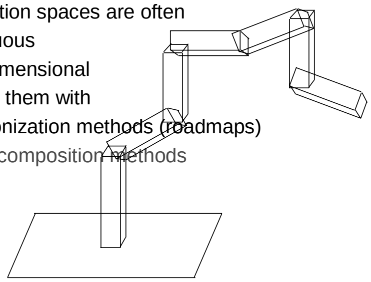




roadmap using random points [Kavraki et al, 1994]
(there are also roadmaps using RRTs [LaValle, 1998]) [from Steve LaValle]

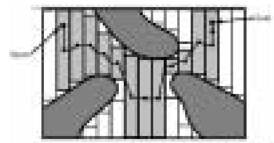
Work vs Configuration Space

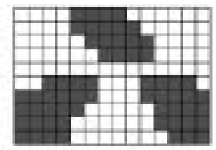
- Configuration spaces are often
  - continuous
  - high-dimensional
- Discretize them with
  - skeletonization methods (roadmaps)
  - cell-decomposition methods



Discretizing Configuration Space

- Cell decomposition methods:
  systematic and resolution complete



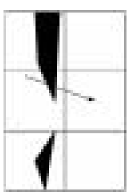


[from Stuart Russell and Peter Norvig]

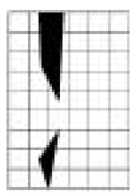
vertical strips — grid

Discretizing Configuration Space

- Cell decomposition methods



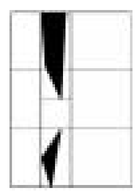
coarse-grained discretization
might not be able to find a path



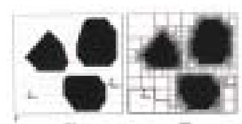
fine-grained discretization
Is very inefficient

Discretizing Configuration Space

- Cell decomposition methods



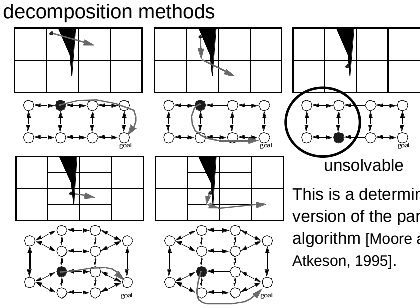
non-uniform discretization
avoids these problems



[from unknown]

Discretizing Configuration Space

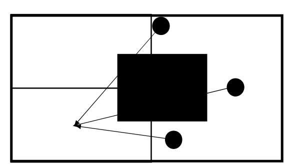
- Cell decomposition methods



This is a deterministic version of the parti-game algorithm [Moore and Atkeson, 1995].

Discretizing Configuration Space

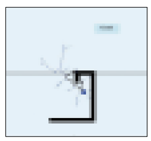
- Cell decomposition methods
- The search space is really nondeterministic and we thus need to use a minimax search



Discretizing Configuration Space

- Cell decomposition methods

- PDRRTs implements the local controllers of the parti-game algorithm with RRTs [Ranganathan and Koenig, 2004].
  - PDRRTs need no user-supplied local controllers.
  - PDRRTs need to split fewer cells.



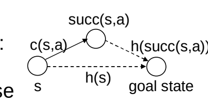
Discretizing Configuration Space

- We use examples with configuration space = 2d work space
  - increase the size of obstacles by the radius of the robot
  - make the robot a point
  - ignore kinematic constraints

Table of Contents

A*

- A* [Hart, Nilsson and Raphael, 1968] 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.

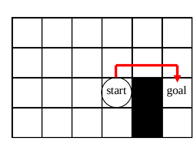
A*

A*

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
3. If state s is a goal state: stop
4. Expand state s
5. Go to 2

A*

- Search problem with uniform cost



4-neighbor grid

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

A*

- First iteration of A*

order of expansions

			1		

g-values

			0		

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

f-values

			2		

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

A*

- Second iteration of A*

order of expansions

			1		
			2		

g-values

			1		
		1	0		
			1		

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

f-values

			4		
		4	2		
			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

A*

- Third iteration of A*

order of expansions

		3	1		
			2		

g-values

			1		
		1	0		
		2	1		

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

f-values

			4		
		4	2		
		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

A*

- Fourth iteration of A*

order of expansions

			4		
		3	1		
			2		

g-values

		2	1		
	2	1	0		
		2	1		

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

f-values

		6	4		
	6	4	2		
		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

A*

- Fifth iteration of A*

order of expansions

			4	5	
		3	1		
			2		

g-values

			2		
		2	1	2	
	2	1	0		
		2	1		

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

f-values

			6		
		6	4	4	
	6	4	2		
		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

A*

- Sixth iteration of A*

order of expansions

			4	5	6
		3	1	■	
			2	■	

g-values + h-values = f-values

			2	3	
		2	1	2	3
	2	1	0	■	
		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	4 ←	4 ←	4
	6 →	4 →	2	■	
		6 →	4	■	

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

A*

- Seventh and last iteration of A*

order of expansions

			4	5	6
		3	1	■	(7)
			2	■	

g-values + h-values = f-values

			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	■	

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

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

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

			4	5	6
		3	1	■	(7)
			2	■	

			6		
			3	4	7
		5	1	■	(8)
			2	■	

	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

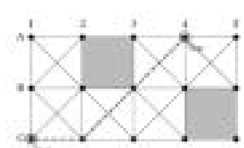
A*

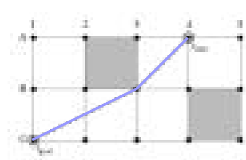
- We say that h-values $h_1(s)$ dominate h-values $h_2(s)$ iff $h_1(s) \geq h_2(s)$ for all states s.
- A* with consistent h-values h(s) [Pearl,1984]
  - expands every state at most once
  - has found a shortest path from the start state to a state when it is about to expand the state
  - has found a 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)

Table of Contents

Any-Angle Path Planning

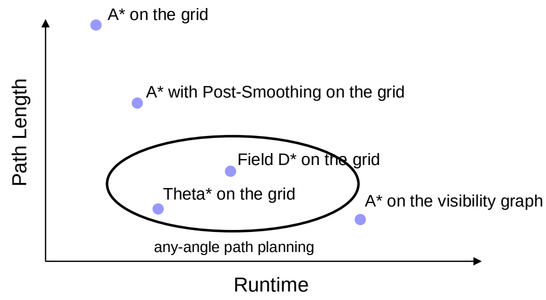
- A* on eight-neighbor grids


grid path

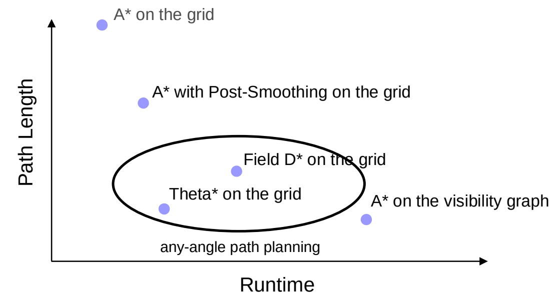

any-angle path

8-neighbor grid

Any-Angle Path Planning

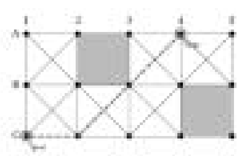


Any-Angle Path Planning

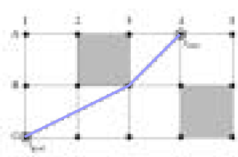


Any-Angle Path Planning

- A* on eight-neighbor grids



grid path

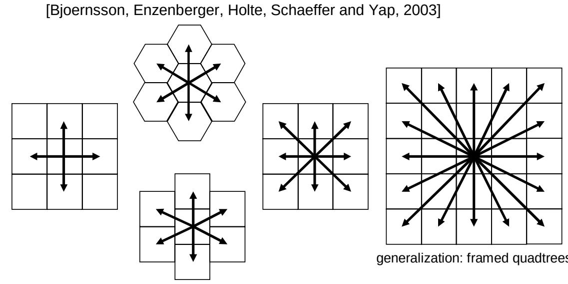


any-angle path

8-neighbor grid

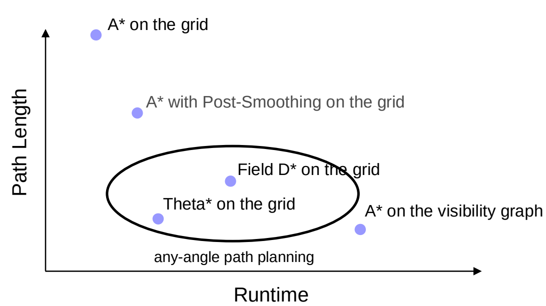
Any-Angle Path Planning

- A* on other tessellations
  [Bjoernsson, Enzenberger, Holte, Schaeffer and Yap, 2003]



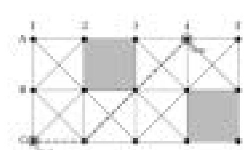
generalization: framed quadtrees

Any-Angle Path Planning

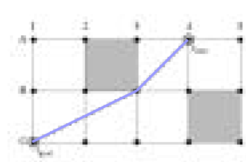


Any-Angle Path Planning

- A* on eight-neighbor grids with smoothing



grid path

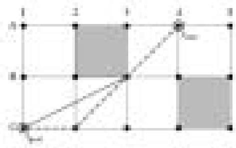


any-angle path

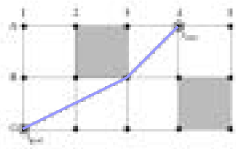
8-neighbor grid

Any-Angle Path Planning

- A* on eight-neighbor grids with smoothing



grid path

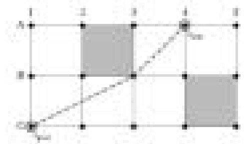


any-angle path

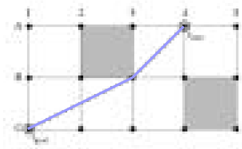
8-neighbor grid

Any-Angle Path Planning

- A* on eight-neighbor grids with smoothing



grid path

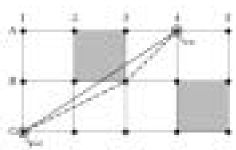


any-angle path

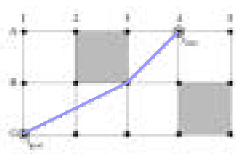
8-neighbor grid

Any-Angle Path Planning

- A* on eight-neighbor grids with smoothing



grid path

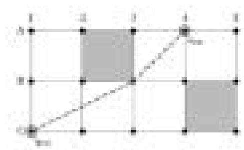


any-angle path

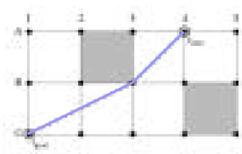
8-neighbor grid

Any-Angle Path Planning

- A* on eight-neighbor grids with smoothing



grid path

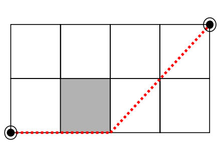


any-angle path

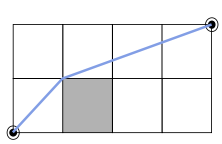
8-neighbor grid

Any-Angle Path Planning

- A* on eight-neighbor grids with smoothing



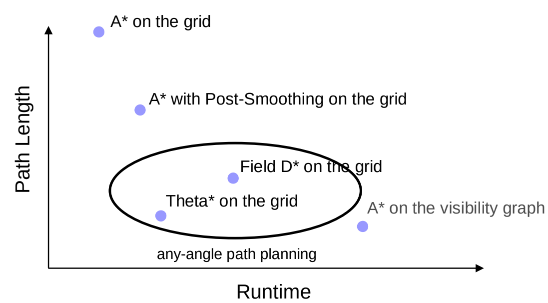
grid path



any-angle path

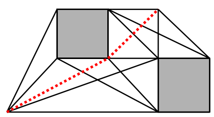
8-neighbor grid

Any-Angle Path Planning

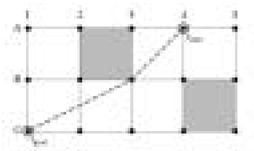


Any-Angle Path Planning

- A* on visibility graphs

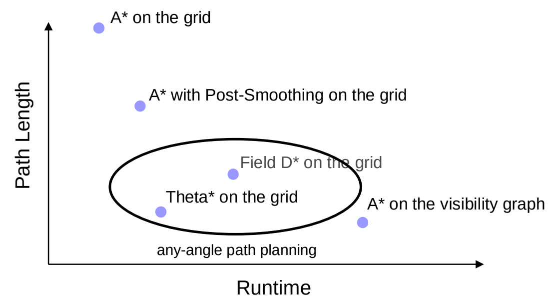


path on visibility graph



shortest path

Any-Angle Path Planning



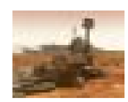
Field D*

- Field D* (a version of D* Lite with any-angle path planning) [Ferguson and Stentz, 2005] on eight-neighbor grids
  - performs an A* search
  - propagates information along the grid edges (= good runtime)
  - does not constrain the path to be on grid edges (= short paths)

Field D*

g-value

- Field D* on eight-neighbor grids



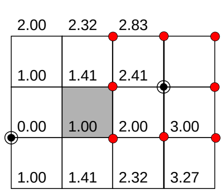
[from JPL]

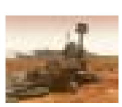
8-neighbor grid

Field D*

g-value

- Field D* on eight-neighbor grids





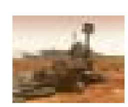
[from JPL]

8-neighbor grid

Field D*

g-value

- Field D* on eight-neighbor grids



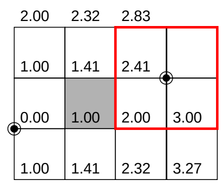
[from JPL]

8-neighbor grid

Field D*

g-value

- Field D* on eight-neighbor grids



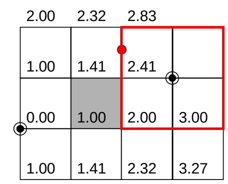
[from JPL]

8-neighbor grid

Field D*

g-value

- Field D* on eight-neighbor grids



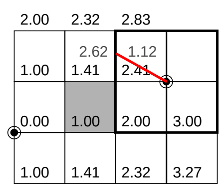
[from JPL]

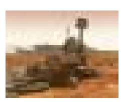
8-neighbor grid

Field D*

g-value

- Field D* on eight-neighbor grids





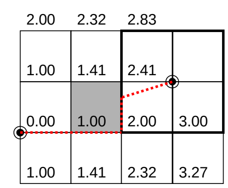
[from JPL]

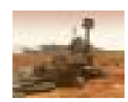
8-neighbor grid

Field D*

g-value

- Field D* on eight-neighbor grids



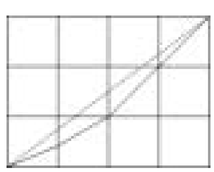


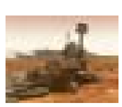
[from JPL]

8-neighbor grid

Field D*

- Field D* on eight-neighbor grids does not necessarily find shortest paths





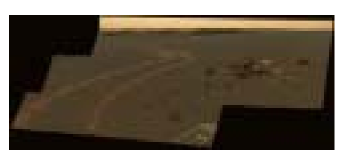
[from JPL]

Field D* path — any-angle path

8-neighbor grid

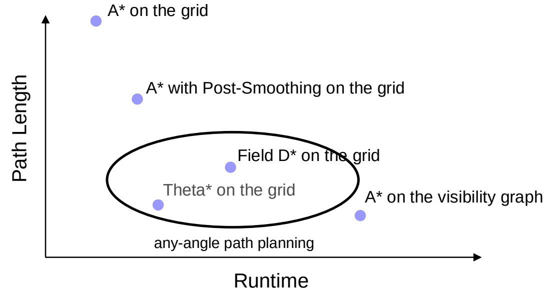
Field D*

- Terrain often has uniform movement costs



[April 29, 2007; from JPL]

Any-Angle Path Planning



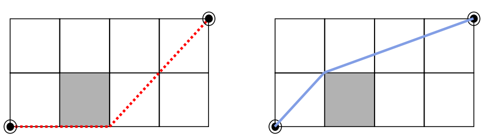
Theta*

- Theta* [Nash, Daniel, Koenig and Felner, 2007*] on eight-neighbor grids
  - performs an A* search
  - propagates information along the grid edges (= good runtime)
  - does not constrain the path to be on grid edges (= short paths)

* Note: A mistake in the pseudo code of AP-Theta* in the original paper is corrected.

Theta*

- A* on eight-neighbor grids with smoothing but now we interleave smoothing with search



grid path any-angle path

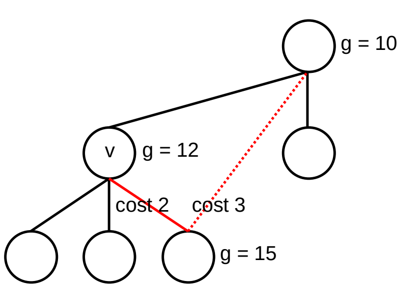
8-neighbor grid

Theta*

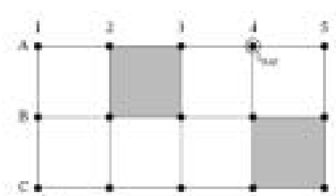
Key insight behind Theta* on eight-neighbor grids

- The parent of a state does not need to be its neighbor.
- When expanding a state s, its children consider not only state s but also the parent of state s as possible parent since it is shorter to go directly to the parent of state s (if that path is unblocked) than first to state s and then to the parent of state s, due to the triangle inequality.

Theta*



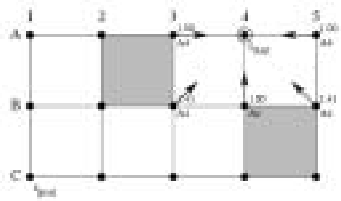
Theta*



8-neighbor grid

Theta*

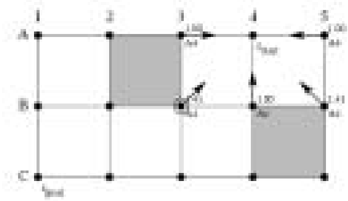
g-value
parent



8-neighbor grid

Theta*

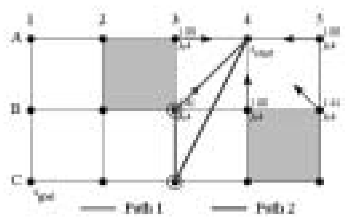
g-value
parent



8-neighbor grid

Theta*

g-value
parent

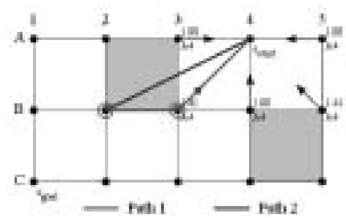


If path 2 is not blocked, then it is shorter than path 1 (triangle inequality)

8-neighbor grid

Theta*

g-value
parent

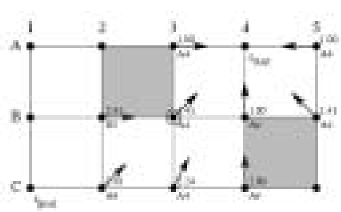


If path 2 is not blocked, then it is shorter than path 1 (triangle inequality)

8-neighbor grid

Theta*

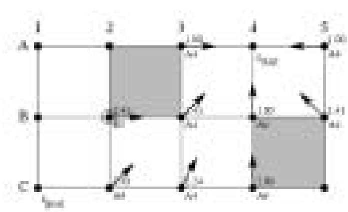
g-value
parent



8-neighbor grid

Theta*

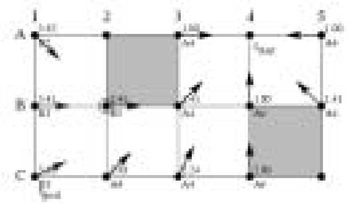
g-value
parent



8-neighbor grid

Theta*

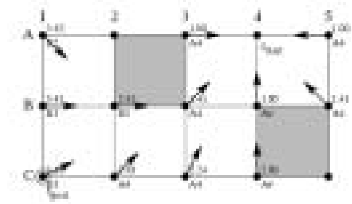
g-value
parent



8-neighbor grid

Theta*

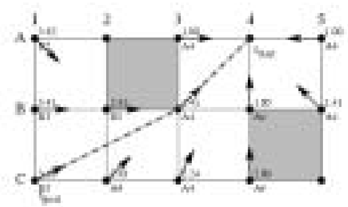
g-value
parent



8-neighbor grid

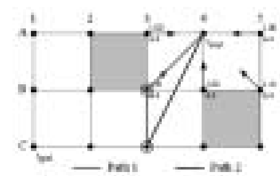
Theta*

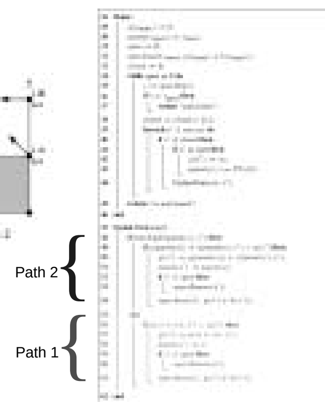
g-value
parent



8-neighbor grid

Theta*

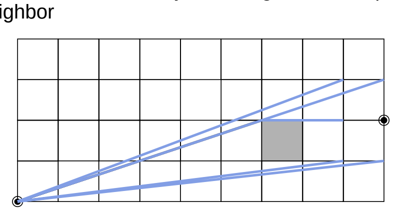




8-neighbor grid

Theta*

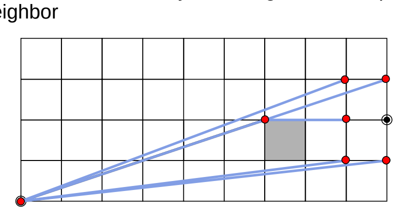
- Theta* does not necessarily find shortest paths since the parent of a state can only be a neighbor or the parent of a neighbor



8-neighbor grid

Theta*

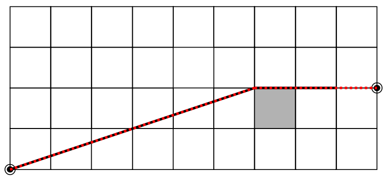
- Theta* does not necessarily find shortest paths since the parent of a state can only be a neighbor or the parent of a neighbor



8-neighbor grid

Theta*

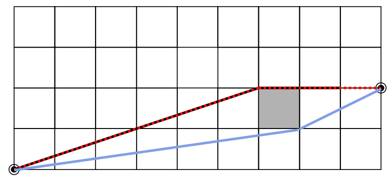
- Theta* does not necessarily find shortest paths since the parent of a state can only be a neighbor or the parent of a neighbor



8-neighbor grid

Theta*

- Theta* does not necessarily find shortest paths since the parent of a state can only be a neighbor or the parent of a neighbor



The path of Theta* is still within 0.2% of optimal for this example

8-neighbor grid

Any-Angle Path Planning

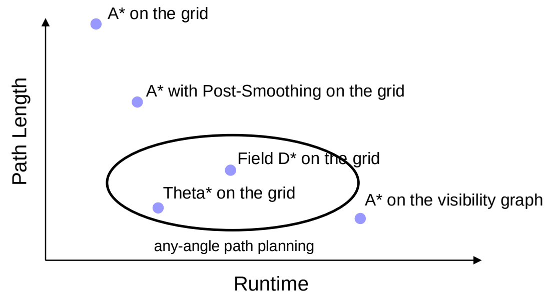
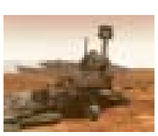


Table of Contents

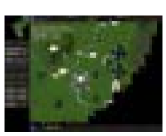
Speeding Up A* Search

Path Planning Problems for Agents

- States are not given, continuous and often hard to characterize
- On-line search: planning and execution have to be interleaved
- Real-time constraints exist
- Search space might or might not fit into memory
- How to search faster and faster?



Robotics [from JPL]



Games [from Cavedog]

20(!) megahertz RAD6000 processor

Speeding Up A* Search

How to search faster and faster is important:

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

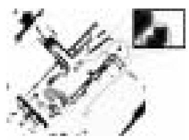
4d (x, y, Θ, v) planning
- > 20,000,000 states
- Slow planning
- Fast execution

[from Maxim Likhachev]

Speeding Up A* Search

How to search faster and faster is important:

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



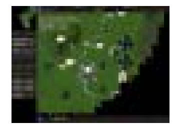
4d (x, y, Θ, v) planning
• > 20,000,000 states
• Slow planning
• Fast execution

[from Maxim Likhachev]

Speeding Up A* Search

How to search faster and faster is important:

- Games need to run on older computers
- Graphics gets most of the processor time
- The number of agents gets larger and larger



Games [from Cavedog]

Speeding Up A* Search

Ways of speeding up A*

- Incremental versions of A* (incremental heuristic search)
  - find shortest paths by exploiting experience with similar searches
  - typically run faster than A*
- A* with weighted h-values (weighted A*)
  - finds suboptimal paths by focusing the search more than A*
  - typically runs faster than A*
- Real-time versions of A* (real-time heuristic search)
  - find suboptimal paths by interleaving searches in local search spaces around the current state and executions
  - can run faster or slower than A*
  - each search runs in constant time

Table of Contents

Incremental Heuristic Search

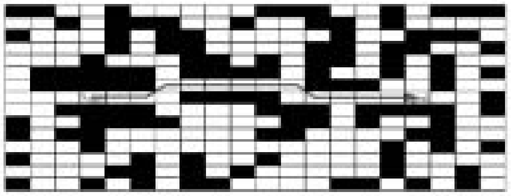
- Incremental heuristic search speeds up A* searches for a sequence of similar search problems by exploiting experience with earlier search problems in the sequence. It finds shortest paths.
- In the worst case, incremental heuristic search cannot be more efficient than A* searches from scratch [Nebel and Koehler 1995].

Incremental Heuristic Search

search task 1	slightly different search task 2	slightly different search task 2

search task 1	slightly different search task 2	slightly different search task 3	slightly different search task 4

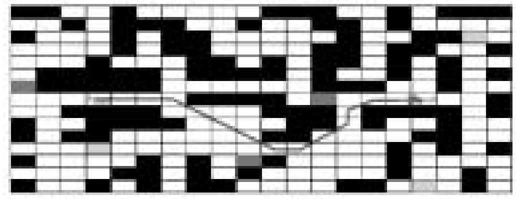
Incremental Heuristic Search



8-neighbor grid

Incremental Heuristic Search

[from slate.com]



8-neighbor grid

Stationary Target

Stationary target search:

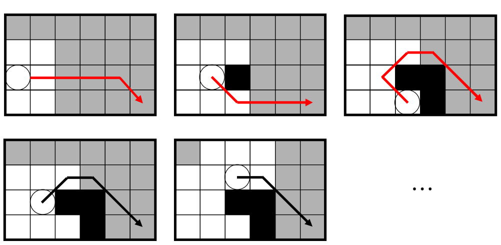
- How to move a computer-controlled agent autonomously to a goal state in initially unknown terrain?

Stationary Target

Our approach to stationary-target search,
called Planning with the Freespace Assumption:

- Repeatedly move the agent along a shortest path from its current state to the goal state under the assumption that states are unblocked unless the agent knows otherwise (freespace assumption). The agent needs to replan its path only if the path becomes blocked.
- Repeatedly find a shortest path from some start state to the same goal state with A* on a graph whose movement costs can increase over time.

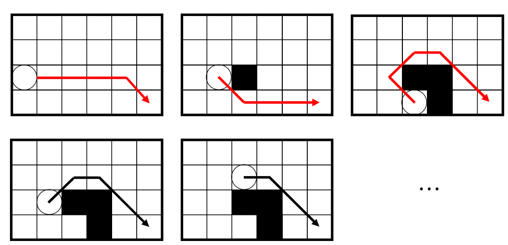
Stationary Target



…

8-neighbor grid

Stationary Target



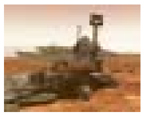
…

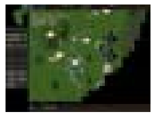
8-neighbor grid

Stationary Target

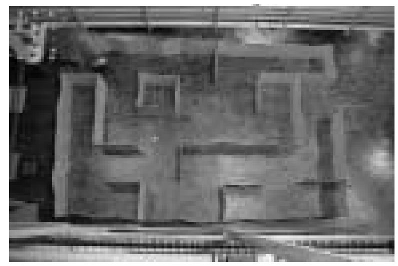
- Used in robotics and usable in games

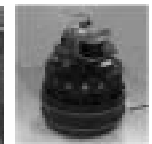

[Stentz and Hebert, 1995]


[from JPL]


[from Cavedog Entertainment]

Stationary Target



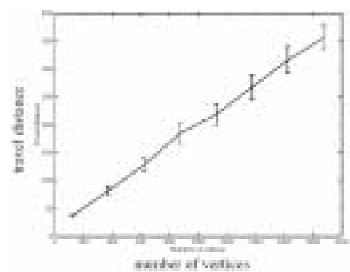


Stationary Target

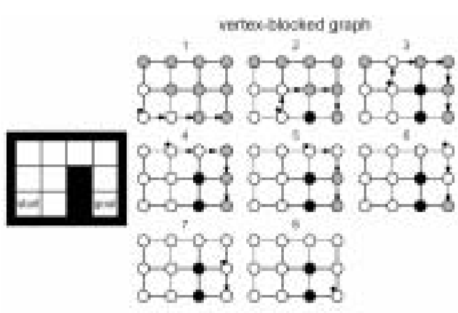
- Clearly, the number of movements is small if the freespace assumption is approximately satisfied, that is, if the obstacle density is small

Stationary Target

- Mazes of size 25 x 5 – 25 x 75



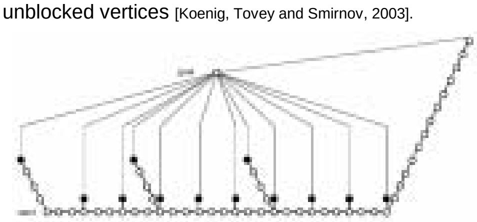
Stationary Target



4-neighbor grid

Stationary Target

- The worst-case number of movements is Ω(log(#states)/log log(#states) × #states) on undirected vertex-blocked graphs, where #states is the number of unblocked vertices [Koenig, Tovey and Smirnov, 2003].


n^n

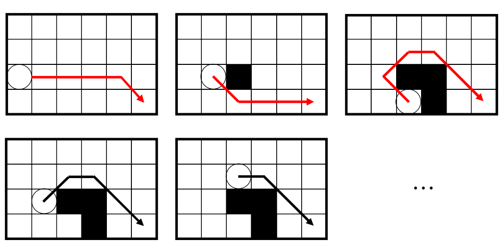
Stationary Target

- The worst-case number of movements is $\Omega(\log(\#states)/\log\log(\#states) \times \#states)$ on undirected vertex-blocked graphs, where #states is the number of unblocked vertices [Koenig, Tovey and Smirnov, 2003].
- Proof:
  - Length of rim = n^n for some n
  - Rim gets traversed n times, resulting in n^{n+1} movements
  - There are about at most n^{n-1} spokes for each of the at most n heights, resulting in n^n states

Stationary Target

- The worst-case number of movements is $\log^2(\#states)$ #states on undirected vertex-blocked graphs and log(#states) #states on vertex-blocked grids, where #states is the number of unblocked vertices [Mudgal, Tovey, Greenberg and Koenig, 2005].

Stationary Target



8-neighbor grid

Incremental Heuristic Search

Incremental heuristic search

- Fringe Saving A* (FSA*) and similar (iA*)
  - starts A* at the point where the current search could differ from the previous one
- Adaptive A* (AA*) and similar (MTAA*, RTAA*)
  - improves the h-values between searches
- Lifelong Planning A* (LPA*) and similar (D*, D* Lite, …)
  - transforms the previous search tree into the current one

runtime per expansion increases
number of expansions decreases

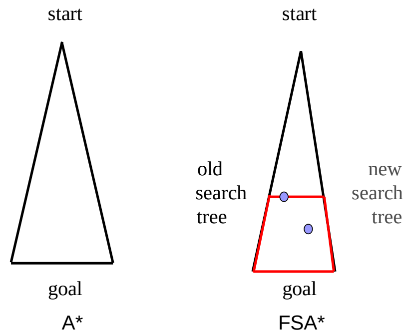
- It is future work to combine the principles behind AA* and LPA*.

Table of Contents

Fringe Saving A* (FSA*)

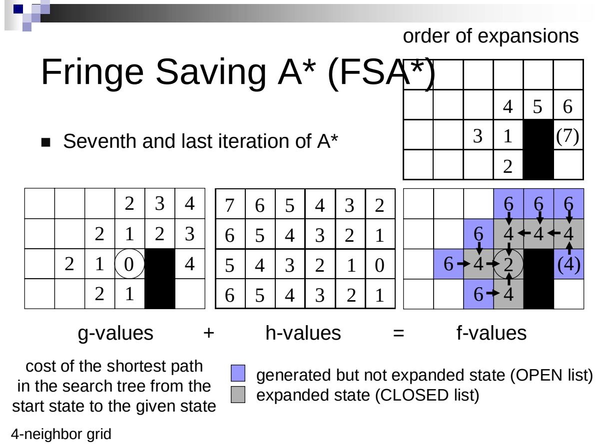
- Fringe Saving A* (FSA*) [Sun and Koenig, 2007] speeds up A* searches for a sequence of similar search problems by starting each search at the point where it could differ from the previous one
- FSA* is similar to but faster than iA* [Yap, unpublished]

Fringe Saving A* (FSA*)



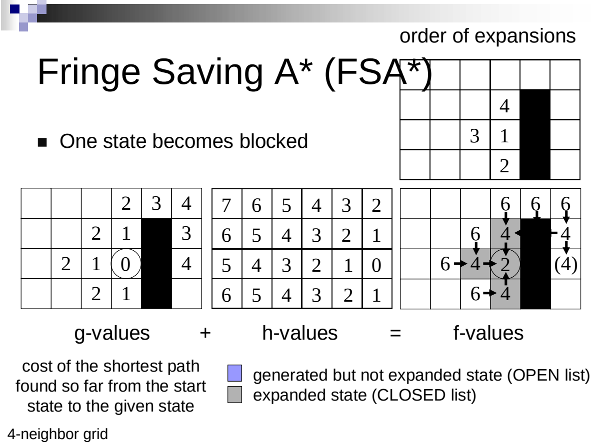
Fringe Saving A* (FSA*)

- Seventh and last iteration of A*



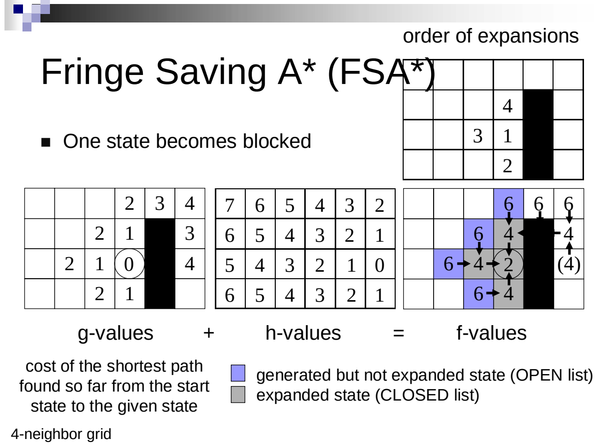
Fringe Saving A* (FSA*)

- One state becomes blocked



Fringe Saving A* (FSA*)

- One state becomes blocked



Fringe Saving A* (FSA*)

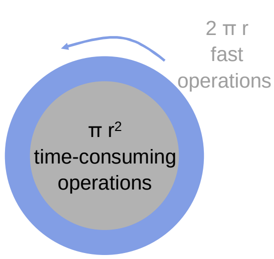
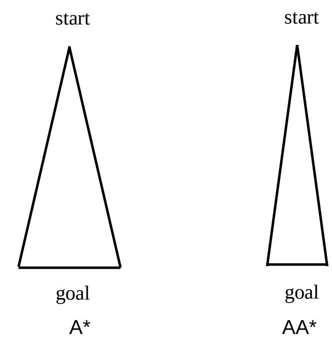


Table of Contents

Adaptive A* (AA*)

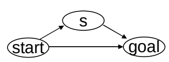
- Adaptive A* (AA*) [Koenig and Likhachev, 2005] speeds up A* searches for a sequence of similar search problems by making the h-values more informed after each search.
- The principle behind AA* was earlier used in Hierarchical A* [Holte et al., 1996].

Adaptive A* (AA*)



Adaptive A* (AA*)

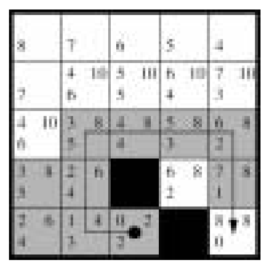
- Consider a state s that was expanded by A* with consistent h-values h_{old}:
  - distance(start,s) + distance(s,goal) ≥ distance(start,goal)
  - distance(s,goal) ≥ distance(start,goal) – distance(start,s)
  - distance(s,goal) ≥ f(goal) – g(s) = h_{new}(s)



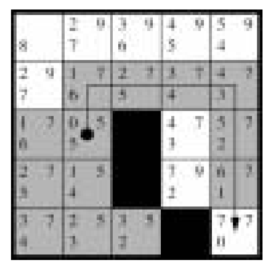
- The h-values h_{new} are again consistent.
- The h-values h_{new} dominate the h-values h_{old}.
- These properties continue to hold even if the start state changes or the movement costs increase.
- The next A* search with h-values h_{new} expands no more states than an A* search with h-values h_{old} and likely many fewer states.

Adaptive A* (AA*)

g f
h



first A* search

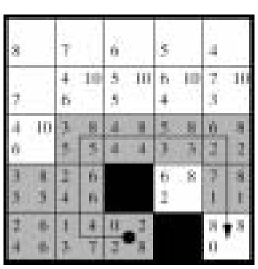


second A* search

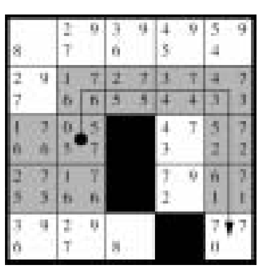
4-neighbor grid

Adaptive A* (AA*)

g f
h_{old} h_{new}



first AA* search



second AA* search

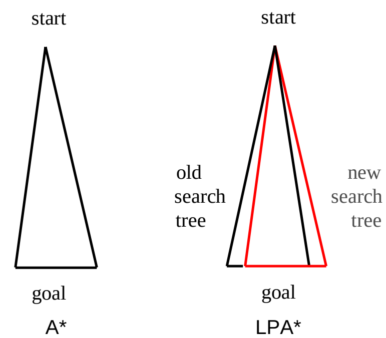
4-neighbor grid

Table of Contents

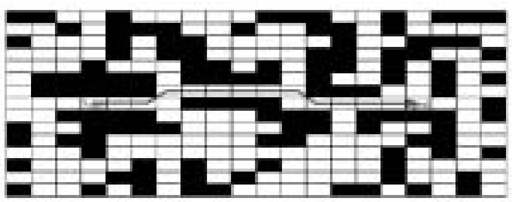
Lifelong Planning A* (LPA*)

- Lifelong Planning A* (LPA*) [Koenig and Likhachev, 2002] 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.

Lifelong Planning A* (LPA*)

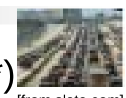


Lifelong Planning A* (LPA*)

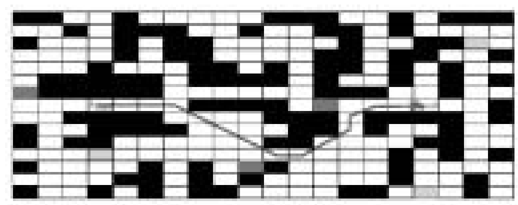


8-neighbor grid

Lifelong Planning A* (LPA*)



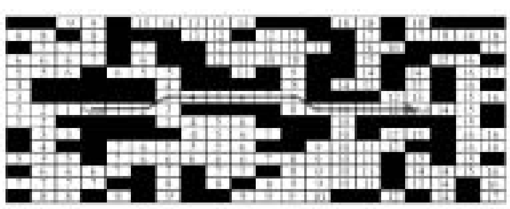
[from slate.com]



8-neighbor grid

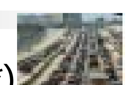
Lifelong Planning A* (LPA*)

g

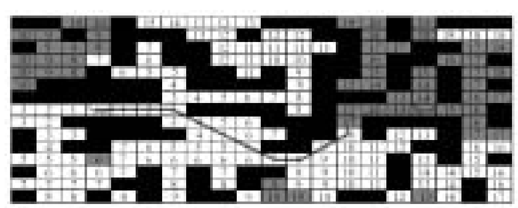


8-neighbor grid

Lifelong Planning A* (LPA*)



[from slate.com]

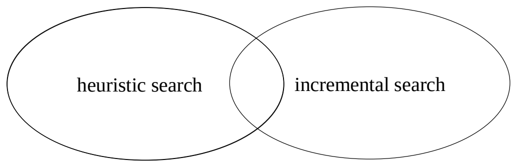


8-neighbor grid

www.slate.com

Lifelong Planning A* (LPA*)

artificial intelligence	algorithm theory
heuristic search	incremental search
how to search efficiently using h-values to focus the search	how to search efficiently by reusing information from previous similar searches



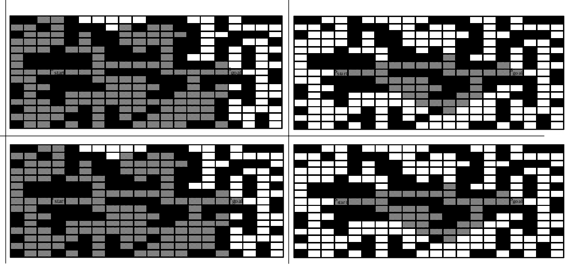
Lifelong Planning A* (LPA*)

	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*) [Koenig and Likhachev, 2002]

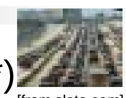
Lifelong Planning A* (LPA*)

uninformed search | heuristic search

complete search / incremental search



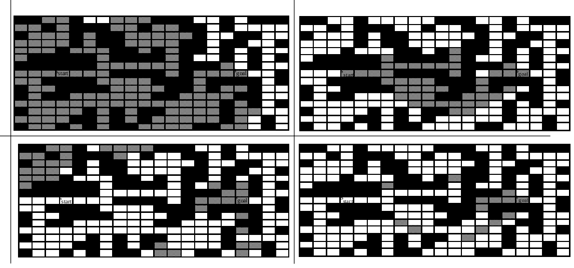
Lifelong Planning A* (LPA*)



[from slate.com]

uninformed search | heuristic search

complete search / incremental search



Lifelong Planning A* (LPA*)

Lifelong Planning A* (LPA*)

g

	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

Lifelong Planning A* (LPA*)

g

	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

Lifelong Planning A* (LPA*)

g

	1	2	start	4	5	6
A	2	1	0	1	2	3
B	3					4
C	4		*2* min(2,4)+2 min(2,4)			5
D	5	4	3	4	5	6
		goal		priority queue		

4-neighbor grid

C3:[4;2]

Lifelong Planning A* (LPA*)

g

	1	2	start	4	5	6
A	2	1	0	1	2	3
B	3					4
C	4		∞ min(∞,4)+2 min(∞,4)			5
D	5	4	*3* min(∞,5)+1 min(3,4)	4	5	6
		goal		priority queue		

4-neighbor grid

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

Lifelong Planning A* (LPA*)

g

	1	2	start	4	5	6
A	2	1	0	1	2	3
B	3					4
C	4		∞			5
D	5	*4* min(4,6)+0 min(4,6)	∞ min(∞,5)+1 min(∞,5)	*4* min(4,6)+2 min(4,6)	5	6
		goal		priority queue		

4-neighbor grid

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

Lifelong Planning A* (LPA*)

g

	1	2	start	4	5	6
A	2	1	0	1	2	3
B	3					4
C	4		∞			5
D	5	∞ min(∞,6)+0 min(∞,6)	∞ min(∞,5)+1 min(∞,5)	*4* min(4,6)+2 min(4,6)	5	6
		goal		priority queue		

4-neighbor grid

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

Lifelong Planning A* (LPA*)

g

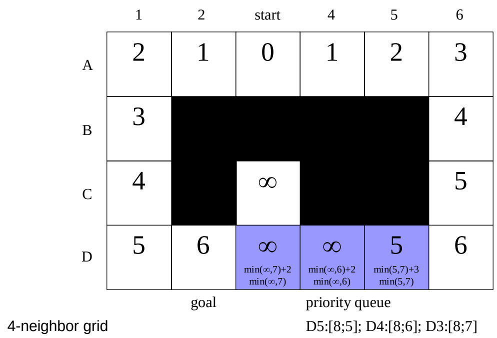
	1	2	start	4	5	6
A	2	1	0	1	2	3
B	3					4
C	4		∞			5
D	5	∞ min(∞,6)+0 min(∞,6)	∞	∞ min(∞,6)+2 min(∞,6)	5 min(5,7)+3 min(5,7)	6
		goal		priority queue		

4-neighbor grid

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

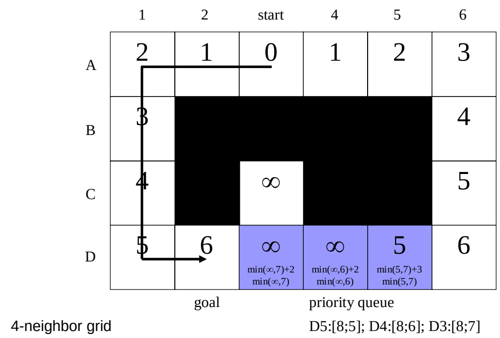
Lifelong Planning A* (LPA*)

g



Lifelong Planning A* (LPA*)

g



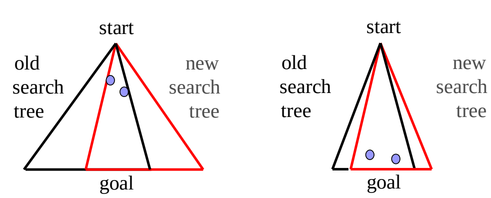
Lifelong Planning A* (LPA*)

- Theorem [Koenig, Likhachev and Furcy, 2004]
  Each search expands every state at most twice and thus terminates.
  = LPA* terminates
- Theorem [Koenig, Likhachev and Furcy, 2004]
  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

Lifelong Planning A* (LPA*)

- Theorem [Koenig, Likhachev and Furcy, 2004]
  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 [Koenig, Likhachev and Furcy, 2004]
  Each search expands at most those states s with $[f(s);g^*(s)] \leq [f(goal); g^*(goal)]$ or $[g_{old}(s) + h(s); g_{old}(s)] \leq [f(goal); g^*(goal)]$, 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

Lifelong Planning A* (LPA*)



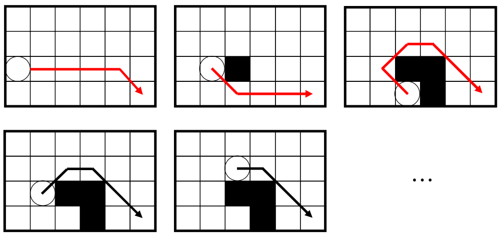
- Start of the search must remain unchanged
- LPA* can expand more states and run slower than A*
- - if the number of changes is large
- - if the changes are close to the start of the search

Lifelong Planning A* (LPA*)

- Grids of size 101 x 101
- Movement costs are one or two with equal probability

number of movement cost changes	planning time of A*	first planning time of LPA*	replanning time of LPA*	replanning time of LPA* / planning time of A*
0.2 %	0.299 ms	0.386 ms	0.029 ms	10.4 x
0.4 %	0.336 ms	0.419 ms	0.067 ms	5.0 x
0.6 %	0.362 ms	0.453 ms	0.108 ms	3.3 x
0.8 %	0.406 ms	0.499 ms	0.156 ms	2.6 x
1.0 %	0.370 ms	0.434 ms	0.174 ms	2.1 x

Stationary Target

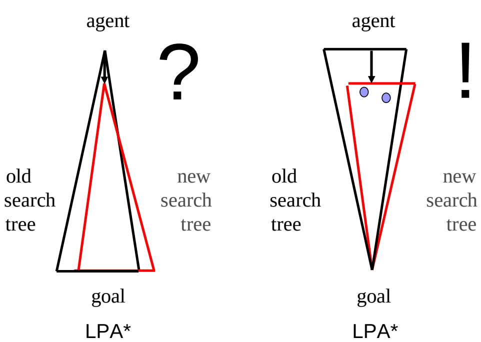


8-neighbor grid

D* Lite

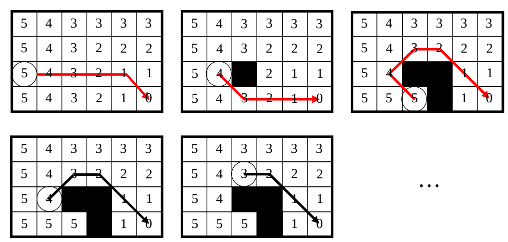
- LPA* needs to search from the goal of the agent to the agent itself because the start of the search needs to remain unchanged.
- LPA* is efficient because the agent observes blockages around itself. Thus, the changes are close to the goal of the search.

D* Lite



D* Lite

goal distance



8-neighbor grid

D* Lite

- D* Lite: Basic Version [Koenig and Likhachev, 2002]
- 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 [Stentz, 1994].

D* Lite

- D* Lite: Basic Version [Koenig and Likhachev, 2002]
- Priority queue: A [8,5]; B [8,6]; C [8,7]
- Agent moves
- Priority queue: C [7,7]; B [8,6]; A [9,5]

D* Lite

- D* Lite: Final Version [Koenig and Likhachev, 2002]
- 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, 1995].

D* Lite

- D* Lite: Final Version [Koenig and Likhachev, 2002]
- 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.

D* Lite

- D* Lite: Final Version [Koenig and Likhachev, 2002]
- 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]

  correct priority is [9,5]

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

  correct priority is [8,6]

  expand B

D* Lite

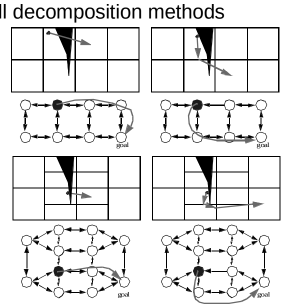
- Random Grids of size 129 x 129

	replanning time
uninformed search from scratch	296.0 ms
informed search from scratch	10.5 ms
uninformed incremental search	6.1 ms
informed incremental search D* [Stentz, 1995] D* was probably the first true incremental heuristic search algorithm, way ahead of its time!	4.2 ms
D* Lite	2.7 ms

speed-up 110x

Minimax LPA*

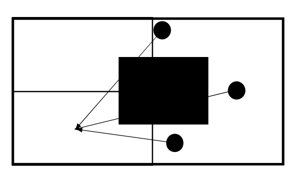
- Cell decomposition methods



This is a deterministic version of the parti-game algorithm [Moore and Atkeson, 1995]

Minimax LPA*

- Cell decomposition methods
- The search space is really nondeterministic and we thus need to use a minimax version of LPA*



Minimax LPA*

- Terrain of size 2000 x 2000

speed-up 26x

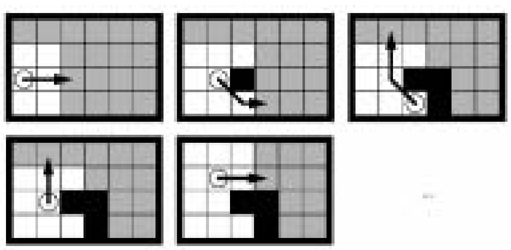
	planning time
uninformed search from scratch	363 minutes
informed search from scratch	135 minutes
uninformed incremental search	15 minutes
informed incremental search (Minimax LPA* [Likhachev and Koenig, 2003])	14 minutes

D* Lite for Mapping

Our approach to mapping, called Greedy Mapping:

- Repeatedly move the agent along a shortest path from its current state to a closest unvisited or unobserved state [Thrun et al. 1998] [Romero, Morales, Sucar, 2001] [Koenig, Tovey and Halliburton, 2001].

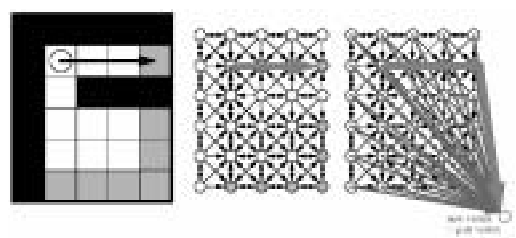
D* Lite for Mapping



8-neighbor grid

D* Lite for Mapping

- Transforming Greedy Mapping to Planning with the Freespace Assumption [Likhachev and Koenig, 2002]



8-neighbor grid

Table of Contents

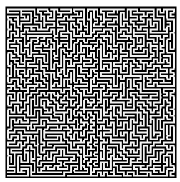
D* Lite vs AA*

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

actually, movement cost in/decreases but AA is more efficient for movement cost increases

D* Lite vs AA*

- Safely explorable torus-shaped mazes of size 100 x 100



D* Lite vs AA*

	expansions per search	runtime per search
Forward A*	3711	581
Backward A*	4104	644
(Forward) AA*	391	81
(Backward) D* Lite	31	15

Table of Contents

Moving Target

Moving-target search:

- How to move a computer-controlled agent autonomously to catch a moving target in initially unknown terrain?

Moving Target

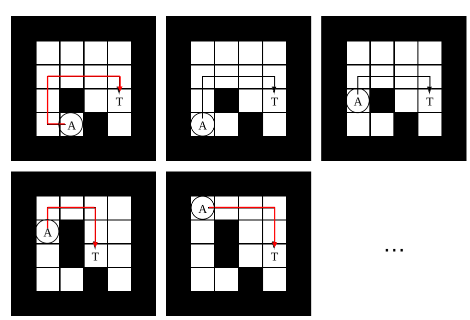
Our approach to moving-target search,
called Planning with the Freespace Assumption:

- Repeatedly move the agent along a shortest path from its current state to the current state of the target under the assumption that states are unblocked unless the agent knows otherwise (freespace assumption). The agent needs to replan its path only if the path becomes blocked or the target leaves the path.

⇩

- Repeatedly find a shortest path from some start state to some goal state with A* on a graph whose movement costs can increase over time.

Moving Target



…

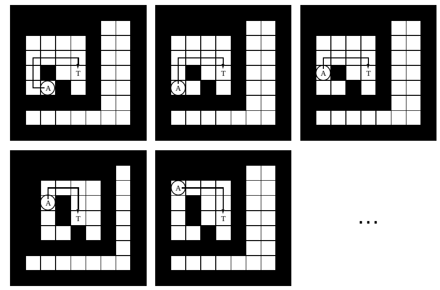
4-neighbor grid

D* Lite vs AA*

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

actually, movement cost in/decreases but AA is more efficient for movement cost increases

D* Lite

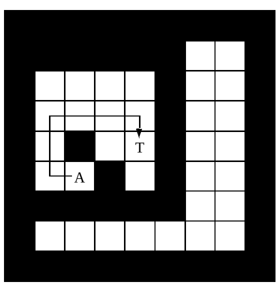


…

4-neighbor grid

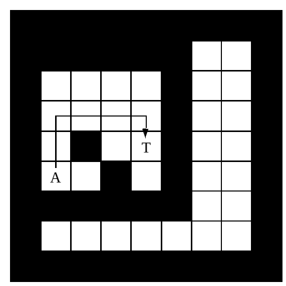
target-centric map [from Tony Stentz]

D* Lite



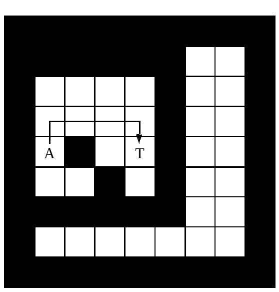
4-neighbor grid

D* Lite



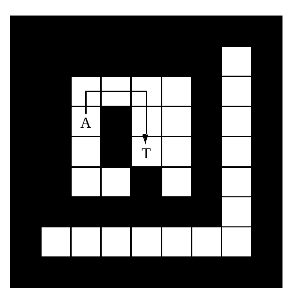
4-neighbor grid

D* Lite



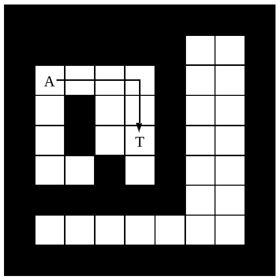
4-neighbor grid

D* Lite



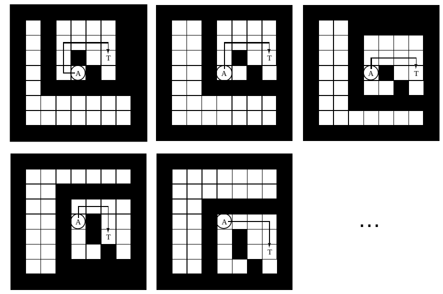
4-neighbor grid

D* Lite



4-neighbor grid

D* Lite

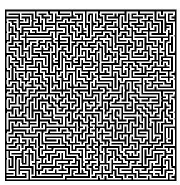


4-neighbor grid

agent-centric map [from Tony Stentz]

D* Lite

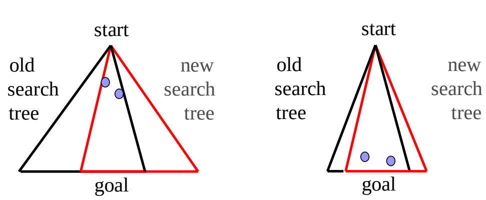
- Safely explorable torus-shaped mazes of size 100 x 100
- Randomly moving target that pauses every 10th move



D* Lite

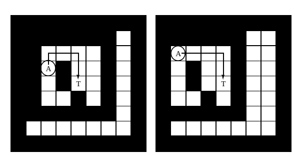
	expansions per search	runtime per search
Forward A*	3703	570
Backward A*	4519	722
Agent-Centric D* Lite	2229	1481
Target-Centric D* Lite	806	833

D* Lite



- Start of the search must remain unchanged
- LPA* can expand more states and run slower than A*
- - if the number of changes is large
- - if the changes are close to the start of the search

D* Lite



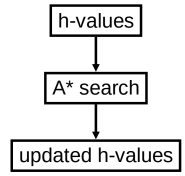
- the map needs to get shifted
- a large number of blockages change
- changed blockages can be close to the start node

4-neighbor grid

Eager Moving-Target Adaptive A*

- We can build an incremental heuristic search method that does not need to shift the map on AA*, resulting in Lazy Moving-Target (MT) AA* [Koenig, Likhachev and Sun, 2007].
- Adaptive A* ⇒ Eager Moving-Target (MT) AA* ⇒ Lazy Moving-Target (MT) AA*

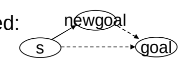
Eager Moving-Target Adaptive A*



update all expanded states
h-values become more informed

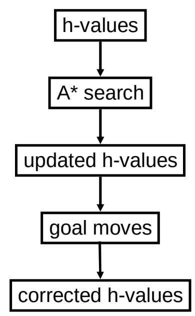
Eager Moving-Target Adaptive A*

- Consider a state s after the goal changed:



  - distance(s,newgoal) + h_{old}(newgoal) ≥ h_{old}(s)
  - distance(s,newgoal) ≥ h_{old}(s) – h_{old}(newgoal)
  - distance(s,newgoal) ≥ max(h_{old}(s) – h_{old}(newgoal), h_{user}(s)) = h_{new}(s)
- The h-values h_{new} are again consistent.
- The h-values h_{new} dominate the h-values h_{user}.
- These properties continue to hold even if the start changes or movement costs increase.
- The next A* search with h-values h_{new} expands no more states than an A* search with h-values h_{user} and likely many fewer states.

Eager Moving-Target Adaptive A*



update all expanded states
h-values become more informed

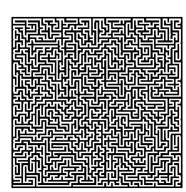
update all states
h-values become less informed
but remain more informed
than the user-supplied h-values

Lazy Moving-Target Adaptive A*

update the h-values only when they are needed

D* Lite vs MTAA*

- Safely explorable torus-shaped mazes of size 100 x 100
- Randomly moving target that pauses every 10^{th} move



D* Lite vs MTAA*

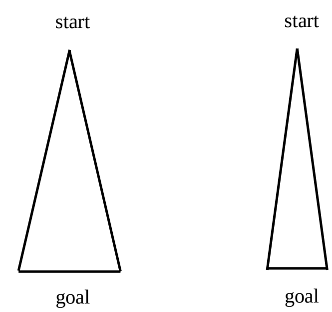
	expansions per search	runtime per search
Forward A*	3703	570
Backward A*	4519	722
Forward Lazy MTAA*	2334	465
Backward Lazy MTAA*	2025	411
Agent-Centric D* Lite	2229	1481
Target-Centric D* Lite	806	833

Table of Contents

Weighted A*

- Weighted A* [Pohl, 1970] solves search problems faster than A* by multiplying consistent h-values with a constant larger than one. It typically does not find shortest paths.

Weighted A*

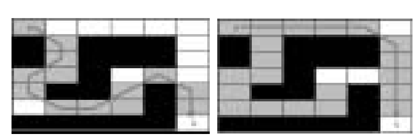


A* Weighted A*

Weighted A*

- Assume that the h-values h(s) are consistent
- A* with the h-values w h(s) for w > 1 [Pearl, 1984; Likhachev, Gordon and Thrun, 2004]
  - can be forced to expand every state at most once
  - typically expands many fewer states the larger w is
  - has found a path from the start state to a state that is at most a factor of w longer than minimal when it is about to expand the state
  - has found a path from the start state to the goal state that is at most a factor of w longer than minimal when it terminates

Weighted A*



w = 2.5
13 expansions
11 movements

w = 1.0 (A*)
20 expansions
10 movements

8-neighbor grid

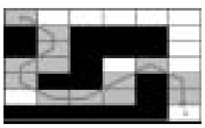
[from Maxim Likhachev]

Table of Contents

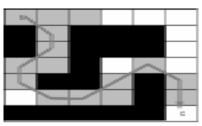
Anytime Repairing A* (ARA*)

- Find a suboptimal path quickly and then make it shorter and shorter (while the agent starts to traverse the path)
- ARA* [Likhachev, Gordon and Thrun, 2004] runs a series of WA* searches with smaller and smaller weights w until a shortest path has been found (or the agent reaches the goal)

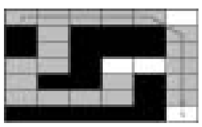
Anytime Repairing A* (ARA*)



w = 2.5
13 expansions
11 movements



w = 1.5
15 expansions
11 movements

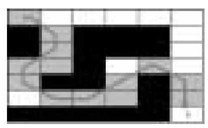


w = 1.0
20 expansions
10 movements

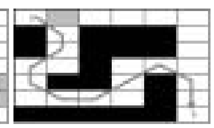
8-neighbor grid

[from Maxim Likhachev]

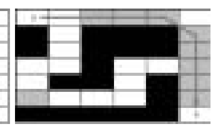
Anytime Repairing A* (ARA*)



w = 2.5
13 expansions
11 movements



w = 1.5
1 expansion
11 movements



w = 1.0
9 expansions
10 movements

8-neighbor grid

[from Maxim Likhachev]

Anytime Repairing A* (ARA*)

4d search with A* (after 25 s)

4d search with ARA* (after 25 s, w = 1.0)

[from Maxim Likhachev]

Anytime Repairing A* (ARA*)

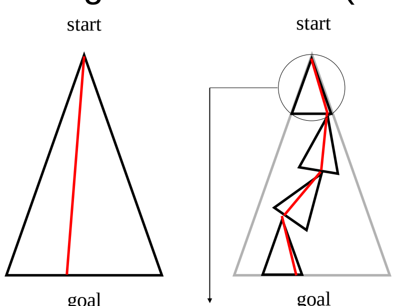
[from Maxim Likhachev]

Table of Contents

Learning Real-Time A* (LRTA*)

- Real-time heuristic search [Korf, 1990] solves search problems with a constant search time between movements by interleaving partial searches around the current state with movements. It updates the h-values after every search to avoid cycling without reaching the goal state. It typically does not follow a shortest trajectory.
- There are many different real-time heuristic search algorithms. We present one of them.

Learning Real-Time A* (LRTA*)



A*

agent-centered search [Koenig, 2001] (e.g. LRTA*)

Learning Real-Time A* (LRTA*)

- Repeatedly move to the most promising adjacent state, using the h-values

5	4	3	2	1	0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

5	4	3	2		0
6	5			2	1
7	6	5	4	3	2
8	7	6	5	4	3

5	4	3	2		0
6	5			2	1
7	6	5	4	3	2
8	7	6	5	4	3

5	4	3	2		0
6	5			2	1
7	6	5	4	3	2
8	7	6	5	4	3

5	4	3	2		0
6	5			2	1
7	6	5	4	3	2
8	7	6	5	4	3

5	4	3	2		0
6	5			2	1
7	6	5	4	3	2
8	7	6	5	4	3

4-neighbor grid

local minima are a problem

Learning Real-Time A* (LRTA*)

- Repeatedly move to the most promising adjacent state, using and updating the h-values

5	4	3	2	1	0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

5	4	3	4		0
6	5			2	1
7	6	5	4	3	2
8	7	6	5	4	3

5	4	5	4		0
6	5			2	1
7	6	5	4	3	2
8	7	6	5	4	3

5	4	5	6		0
6	5			2	1
7	6	5	4	3	2
8	7	6	5	4	3

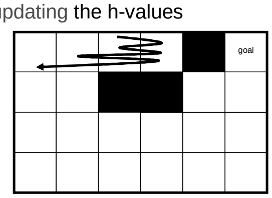
5	4	5	6		0
6	5			2	1
7	6	5	4	3	2
8	7	6	5	4	3

5	6	5	6		0
6	5			2	1
7	6	5	4	3	2
8	7	6	5	4	3

local minima are overcome by updating the h-values

Learning Real-Time A* (LRTA*)

- Repeatedly move to the most promising adjacent state, using and updating the h-values



4-neighbor grid

Learning Real-Time A* (LRTA*)

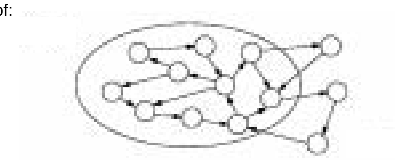
Properties of Learning Real-Time A* (LRTA*) [Korf, 1990]:

- The h-values of the same state are monotonically nondecreasing over time and thus indeed become more informed over time.
- The h-values remain consistent.
- The agent reaches a goal state with O(#states^2) movements if the goal distance of every state is finite [Koenig, 2001].
- If the agent is reset into the start state whenever it reaches a goal state then the number of times that it does not follow a cost-minimal trajectory from the start state to a goal state is bounded from above by a constant if the cost increases are bounded from below by a positive constant.

Learning Real-Time A* (LRTA*)

- LRTA* reaches the goal state if it is reachable from every state (= the search space is safely explorable).

- Proof:



Learning Real-Time A* (LRTA*)

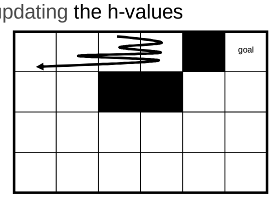
- The worst-case number of movements is O(#states^2) if the goal state is reachable from every state and all movement costs are one, where #states is the number of unblocked vertices [Koenig, 2001].
- Proof under the assumption that all movements change state: Consider the sum of all h-values minus the h-value of the current state. The initial sum is at least zero. The final sum is at most #states × diameter since the h-value of every state is at most its goal distance. Every movement increases the sum by at least one.

before: 5 → 4 afterwards: 5 → 4

before: 5 → 6 afterwards: 7 → 6

Learning Real-Time A* (LRTA*)

- Repeatedly move to the most promising adjacent state, using and updating the h-values



4-neighbor grid

Learning Real-Time A* (LRTA*)

We need larger lookaheads.
The possible design choices differ as follows:

- Which states to search?
- The h-values of which states to update?
- How many moves to make before the next search?

Learning Real-Time A* (LRTA*)

We need larger lookaheads.
We make the following design choices [Koenig, 2004]:

- Which states to search?
  The number x of states to search is determined by the available time and is thus a parameter. We use the first x states expanded by an A* search. An A* search uses h-values to focus the search and always tries to disprove the path currently believed to be shortest.
- The h-values of which states to update?
  We use Dijkstra's algorithm to update the h-values of all x states searched.
- How many moves to make before the next search?
  We move the agent until it reaches a state different from the x states searched.

Learning Real-Time A* (LRTA*)

5	4	3	2	1	0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

4-neighbor grid

Learning Real-Time A* (LRTA*)

- Step 1: Forward A* search

5	4		2	1	0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

first A* state expansion

4-neighbor grid

Learning Real-Time A* (LRTA*)

- Step 1: Forward A* search

5	4			1	0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

second A* state expansion

4-neighbor grid

Learning Real-Time A* (LRTA*)

- Step 1: Forward A* search

5	4				0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

third A* state expansion

4-neighbor grid

Learning Real-Time A* (LRTA*)

- Step 1: Forward A* search

5	4				0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

third A* state expansion

4-neighbor grid

Learning Real-Time A* (LRTA*)

- Step 1: Forward A* search

5	4				0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

third A* state expansion

4-neighbor grid

Learning Real-Time A* (LRTA*)

5	4	∞	∞	∞	0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

4-neighbor grid

Learning Real-Time A* (LRTA*)

- Step 2: Updating the h-values with Dijkstra's algorithm

5	4	∞	∞	1	0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

first iteration of Dijkstra's algorithm

4-neighbor grid

Learning Real-Time A* (LRTA*)

- Step 2: Updating the h-values with Dijkstra's algorithm

5	4	∞	2	1	0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

second iteration of Dijkstra's algorithm

4-neighbor grid

Learning Real-Time A* (LRTA*)

- Step 2: Updating the h-values with Dijkstra's algorithm

5	4	3	2	1	0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

third iteration of Dijkstra's algorithm

4-neighbor grid

Learning Real-Time A* (LRTA*)

- Step 2: Updating the h-values with Dijkstra's algorithm

5	4	3	2	1	0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

4-neighbor grid

Learning Real-Time A* (LRTA*)

- Step 3: Moving along the path

5	4	3	2	1	0
6	5		3	2	1
7	6	5	4	3	2
8	7	6	5	4	3

follow the path

4-neighbor grid

Learning Real-Time A* (LRTA*)

- Step 3: Moving along the path

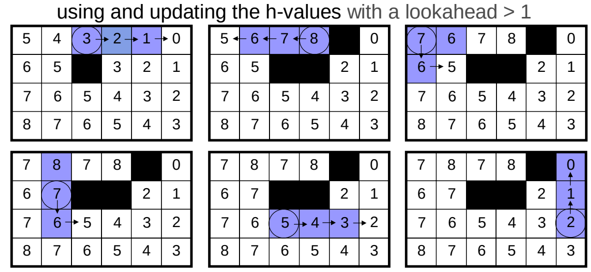
5	4	3	2		0
6	5			2	1
7	6	5	4	3	2
8	7	6	5	4	3

follow the path

4-neighbor grid

Learning Real-Time A* (LRTA*)

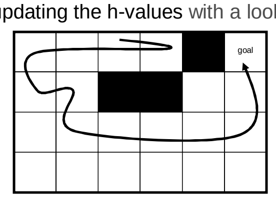
- Repeatedly move to the most promising adjacent state, using and updating the h-values with a lookahead > 1



4-neighbor grid

Learning Real-Time A* (LRTA*)

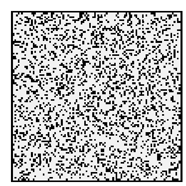
- Repeatedly move to the most promising adjacent state, using and updating the h-values with a lookahead > 1



4-neighbor grid

Learning Real-Time A* (LRTA*)

- Safely explorable random grids of size 301 x 301



Grids with 25% Random Obstacles
h-values generally not misleading
larger lookaheads less helpful

Learning Real-Time A* (LRTA*)

lookahead	Manhattan distance		octile distance	
	planning time	path length	planning time	path length
1	28280	499	28293	363
11	28698	315	28878	315
21	29153	302	29477	311
31	29615	299	…	…
41	…	…	…	…

Learning Real-Time A* (LRTA*)

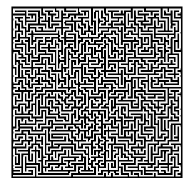
lookahead	LRTA* with A*		LRTA* with BFS	
	state exp.	path length	state exp.	path length
1	499	499	497	497
5	686	338	883	341
11	1014	315	1377	318
15	1238	307	1717	314
21	1579	302	2169	310
25	1822	301	2465	308

Learning Real-Time A* (LRTA*)

- LRTA* with small lookaheads does well in terms of path length since the h-values are generally not misleading.
- Dominating h-values draw the agent towards the goal and result in smaller planning time and path lengths for LRTA* because the h-values are generally not misleading and there are thus only a small number of local minima.
- LRTA* with A* to determine which states to search does better than LRTA* with breadth-first search, both in terms of "planning time" and path length, because the h-values are generally not misleading.

Learning Real-Time A* (LRTA*)

- Safely explorable mazes of size 301 x 301



Acyclic Mazes (generated with DFS)
h-values generally misleading
larger lookaheads very helpful

Learning Real-Time A* (LRTA*)

lookahead	Manhattan distance		octile distance	
	planning time	path length	planning time	path length
1	985362	1987574	628175	1259958
11	313998	337704	277974	272842
21	279856	205370	273280	177143
31	…	…	310131	135554
41	…	…	348330	114917

Learning Real-Time A* (LRTA*)

lookahead	LRTA* with A*		LRTA* with BFS	
	state exp.	path length	state exp.	path length
1	1259958	1259958	1244573	1244573
5	765645	477525	608564	339733
11	531955	272842	437527	189937
15	517913	239073	460207	177181
21	459566	177143	448383	144254
25	456752	155736	473433	138035

Learning Real-Time A* (LRTA*)

- Mazes are easier than grids with random obstacles since their branching factor is smaller. They are harder than grids with random obstacles since the paths between locations are longer and the h-values are generally misleading.
- LRTA* with small lookaheads does poorly in terms of path length since the h-values are generally misleading
- Dominating h-values draw the agent towards the goal and result in larger planning time and path lengths for LRTA* because the h-values are generally misleading and it takes longer to update the h-values to eliminate local minima.
- LRTA* with A* to determine which states to search does worse than LRTA* with breadth-first search, both in terms of "planning time" and path length, because the h-values are generally misleading.

Table of Contents

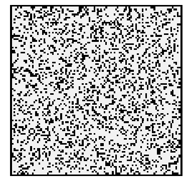
LRTA* vs D* Lite

D* Lite
- can detect that the goal state is unreachable
- cannot satisfy hard real-time requirements
- has worst-case number of movements of O(#states log #states)

LRTA*
- cannot detect that the goal state is unreachable
- can satisfy hard real-time requirements
- has worst-case number of movements of O($\#states^2$)

LRTA* vs D* Lite

- Safely explorable random grids of size 301 x 301



Grids with 25% Random Obstacles
h-values generally not misleading
larger lookaheads less helpful

LRTA* vs D* Lite

lookahead	Manhattan distance		octile distance	
	planning time	path length	planning time	path length
D* Lite	36826	309	40737	314
1	28280	499	28293	363
11	28698	315	28878	315
21	29153	302	29477	311
31	29615	299	…	…
41	…	…	…	…

LRTA* vs D* Lite

- Minimize sum of planning and plan-execution time: planning time + x plan-execution time

range of x for LRTA*	optimal lookahead
$10^{-4.00}$-$10^{-0.09}$	1
$10^{-0.08}$-$10^{+0.14}$	3
$10^{+0.15}$-$10^{+1.06}$	5
$10^{+1.07}$-$10^{+1.07}$	7
…	…

planning is slow
plan-execution is fast

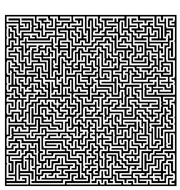
planning is fast
plan-execution is slow

minimum planning time of LRTA*

lookahead increases

LRTA* vs D* Lite

- Safely explorable mazes of size 301 x 301

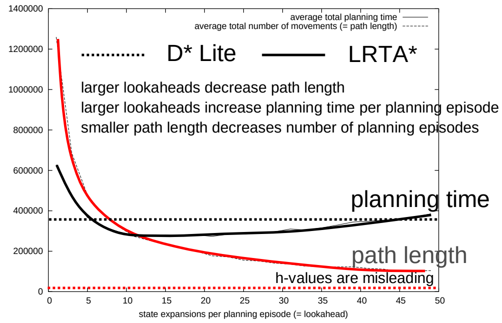


Acyclic Mazes (generated with DFS)
h-values generally misleading
larger lookaheads very helpful

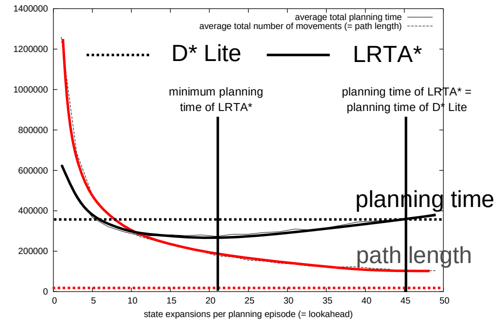
LRTA* vs D* Lite

lookahead	Manhattan distance		octile distance	
	planning time	path length	planning time	path length
D* Lite	357417	21738	373561	21140
1	985362	1987574	628175	1259958
11	313998	337704	277974	272842
21	279856	205370	273280	177143
31	…	…	310131	135554
41	…	…	348330	114917

LRTA* vs D* Lite



LRTA* vs D* Lite



LRTA* vs D* Lite

- Minimize sum of planning and plan-execution time: planning time + x plan-execution time

	range of x for LRTA*	optimal lookahead	
planning is slow, plan-execution is fast	$10^{-4.00}$-$10^{-0.31}$	21	minimum planning time of LRTA*
	$10^{-0.30}$-$10^{-0.16}$	25	lookahead increases
	$10^{-0.15}$-$10^{+0.29}$	33	
planning is fast, plan-execution is slow	…	…	

D* Lite should be preferred for $x > 10^{-0.27}$

Table of Contents

Real-Time Adaptive A* (RTAA*)

- We use AA* to create Real-Time Adaptive A* (RTAA*) [Koenig and Likhachev, 2006], a real-time heuristic search method with similar properties as LRTA*. RTAA* improves on LRTA* by updating the h-values much faster although they are not quite as informed.

Real-Time Adaptive A* (RTAA*)

- LRTA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	4		2	1
4	3	2		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	4		2	1
4	3			0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	4		2	1
4				0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	4		2	1
				0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5			2	1
				0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
			2	1
				0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6		4	3	2
			2	1
				0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6			3	2
			2	1
				0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 1: forward A* search

state about to be expanded

8	7	6	5	4
7	6	5	4	3
6			3	2
			2	1
				0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 2: updating the h-values

8	7	6	5	4
7	6	5	4	3
6			3	2
			2	1
				0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 2: updating the h-values

8	7	6	5	4
7	6	5	4	3
6	∞	∞	3	2
∞	∞		2	1
∞	∞	∞		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 2: updating the h-values

8	7	6	5	4
7	6	5	4	3
6	∞	4	3	2
∞	∞		2	1
∞	∞	∞		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 2: updating the h-values

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
∞	∞		2	1
∞	∞	∞		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 2: updating the h-values

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
∞	6		2	1
∞	∞	∞		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 2: updating the h-values

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
7	6		2	1
∞	∞	∞		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 2: updating the h-values

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
7	6		2	1
∞	7	∞		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 2: updating the h-values

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
7	6		2	1
8	7	∞		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 2: updating the h-values

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
7	6		2	1
8	7	8		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 2: updating the h-values

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
7	6		2	1
8	7	8		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 3: moving along the path

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
7	6		2	1
8	7	8		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 3: moving along the path

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
7	6		2	1
8	7	8		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 3: moving along the path

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
7	6		2	1
8	7	8		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- LRTA* step 3: moving along the path

8	7	6	5	4
7	6	5	4	3
6	5		3	2
7	6		2	1
8	7	8		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

Properties of LRTA* [Korf, 1990]

- The h-values of the same state are monotonically nondecreasing over time and thus indeed become more informed over time.
- The h-values remain consistent.
- The agent reaches a goal state if the goal distance of every state is finite.
- If the agent is reset into the start state whenever it reaches a goal state then the number of times that it does not follow a cost-minimal trajectory from the start state to a goal state is bounded from above by a constant if the cost increases are bounded from below by a positive constant.

Real-Time Adaptive A* (RTAA*)

- RTAA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	4		2	1
4	3	2		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	4		2	1
4	3	**0**		0

bold = g-value
regular = h-value

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	4		2	1
4	**1**	**0**		0

bold = g-value
regular = h-value

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	4		2	1
2	**1**	**0**		0

bold = g-value
regular = h-value

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	**2**		2	1
2	**1**	**0**		0

bold = g-value
regular = h-value

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
3	**2**		2	1
2	**1**	**0**		0

bold = g-value
regular = h-value

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	**3**	4	3	2
3	**2**		2	1
2	**1**	**0**		0

bold = g-value
regular = h-value

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	3	4	3	2
3	2		2	1
2	1	0		0

bold = g-value
regular = h-value

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 1: forward A* search

8	7	6	5	4
7	6	5	4	3
6	3	4	3	2
3	2		2	1
2	1	0		0

state about to be expanded
g-value = 5
h-value = 3
f-value = 8

bold = g-value
regular = h-value

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 2: updating the h-values
  - RTAA*: For each expanded state s: $h_{new}(s)$ = f(~~goal~~) – g(s) f(state about to be expanded)
  - LRTA*: For each expanded state s: use Dijkstra to determine $h_{new}(s)$

8	7	6	5	4
7	6	5	4	3
6	3	4	3	2
3	2		2	1
2	1	0		0

state about to be expanded
g-value = 5
h-value = 3
f-value = 8

bold = g-value
regular = h-value

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 2: updating the h-values

8	7	6	5	4
7	6	5	4	3
6	8-3	8-4	3	2
8-3	8-2		2	1
8-2	8-1	8-0		0

state about to be expanded
g-value = 5
h-value = 3
f-value = 8

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 2: updating the h-values

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	6		2	1
6	7	8		0

state about to be expanded
g-value = 5
h-value = 3
f-value = 8

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 2: updating the h-values

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	6		2	1
6	7	8		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 3: moving along the path

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	6		2	1
6	7	8		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 3: moving along the path

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	6		2	1
6	7	8		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 3: moving along the path

8	7	6	5	4
7	6	5	4	3
6	5	4	3	2
5	6		2	1
6	7	8		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA* step 3: moving along the path

8	7	6	5	4
7	6	5	4	3
6	5		3	2
5	6		2	1
6	7	8		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

Properties of RTAA* [Koenig and Likhachev, 2006]

- The h-values of the same state are monotonically nondecreasing over time and thus indeed become more informed over time.
- The h-values remain consistent.
- The agent reaches a goal state if the goal distance of every state is finite.
- If the agent is reset into the start state whenever it reaches a goal state then the number of times that it does not follow a cost-minimal trajectory from the start state to a goal state is bounded from above by a constant if the cost increases are bounded from below by a positive constant.

Real-Time Adaptive A* (RTAA*)

- RTAA*

8	7	6	5	4
7	6	5	4	3
6	5		3	2
5	6		2	1
6	7	8		0

- LRTA*

8	7	6	5	4
7	6	5	4	3
6	5		3	2
7	6		2	1
8	7	8		0

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- RTAA*

8	7	6	5	4
7				3
			3	2
			2	1
6	7	8		0

- LRTA*

8	7	6	5	4
7				
			3	2
7			2	1
8	7	8		0

4-neighbor grid

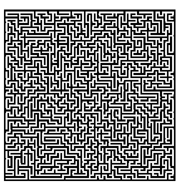
Real-Time Adaptive A* (RTAA*)

Relationship of RTAA* and LRTA*

- RTAA* with only one expanded state per A* search behaves exactly like LRTA* with only one expanded state per A* search.
- If RTAA* and LRTA* have the same h-values before they update the h-values then the h-values of RTAA* after the update are dominated by the h-values of LRTA*.

Real-Time Adaptive A* (RTAA*)

- Safely explorable mazes of size 151 x 151



Real-Time Adaptive A* (RTAA*)

	RTAA*			LRTA*		
	expansions	trajectory length	time per search [ms]	expansions	trajectory length	time per search [ms]
1	248538	248538	0.20	248538	248538	0.27
9	104229	56708	2.01	87613	47291	2.80
17	85866	33853	4.37	79313	30470	6.25
25	89258	26338	6.86	82851	23270	10.23
33	96840	22022	9.41	92908	20016	14.31
41	105703	18629	11.99	102788	17274	18.50
49	117036	16638	14.46	113140	15398	22.67
57	128560	15367	16.83	125013	14285	26.69

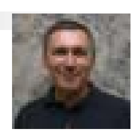
+7%

-59%

Real-Time Adaptive A* (RTAA*)

	RTAA*			LRTA*		
	expansions	trajectory length	time per search [ms]	expansions	trajectory length	time per search [ms]
1	248538	248538	0.20	248538	248538	0.27
9	104229	56708	2.01	87613	47291	2.80
17	85866	33853	4.37	79313	30470	6.25
25	89258	26338	6.86	82851	23270	10.23
33	96840	22022	9.41	92908	20016	14.31
41	105703	18629	11.99	102788	17274	18.50
49	117036	16638	14.46	113140	15398	22.67
57	128560	15367	16.83	125013	14285	26.69

Tom Mitchell Slide



- We are only at the beginning of exploring the theory and applications of incremental heuristic search algorithms.
- This is a good topic for dissertations!
  - What other principles exist?
  - What are the properties of these principles?
  - How can these principles be combined?
  - How to broaden their applications?
    - How to do memory-limited incremental heuristic search?
    - How to do probabilistic incremental heuristic search?
  - What other problems can they be applied to?
    - How to apply them to symbolic planning?
    - How to apply them to constraint optimization?

Summary

- Joint work with K. Daniel, A . Felner, S. Greenberg, W. Halliburton, M. Likhachev, A. Mudgal, A. Nash, A. Ranganathan, Y. Smirnov, X. Sun and C. Tovey
- Many thanks to Vadim Bulitko and Maxim Likhachev for making their movies available
- Funded in part by **NSF**, IBM and JPL

- For more information, see idm-lab.org/projects.html