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

- 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*

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

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*

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

[from Wikipedia]
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]

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**

Configuration spaces are often
- continuous
- high-dimensional

Discretize them with
- skeletonization methods (roadmaps)
- cell-decomposition methods

**Discretizing Configuration Space**

Skeletonization methods

Visibility graph

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

Discretizing Configuration Space

- Cell decomposition methods

Discretizing Configuration Space

- Coarse-grained discretization might not be able to find a path
- Fine-grained discretization is very inefficient

Discretizing Configuration Space

- Non-uniform discretization avoids these problems

Discretizing Configuration Space

- 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.

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*

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* [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:
  \[ h(s) = 0 \text{ if } s \text{ is the goal state} \]
  \[ h(s) \leq c(s,a) + h(succ(s,a)) \text{ otherwise} \]
- Consistent h-values are admissible.
- The h-values h(s) are admissible if they do not overestimate the goal distances.

A* Search problem with uniform cost

- Search problem with uniform cost
A*

- Possible consistent h-values

<table>
<thead>
<tr>
<th>Manhattan Distance</th>
<th>Octile Distance</th>
<th>Zero h-values</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2</td>
<td>5 4 3 2 2 2</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>6 5 4 3 2 1 0</td>
<td>5 4 3 2 1 1 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>6 5 4 3 2 1 0</td>
<td>5 4 3 2 1 1 0</td>
<td>0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

more informed (dominating)

4-neighbor grid

A*

- First iteration of A*

<table>
<thead>
<tr>
<th>g-values</th>
<th>+</th>
<th>h-values</th>
<th>=</th>
<th>f-values</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2</td>
<td>6 5 4 3 2 1</td>
<td>5 4 3 2 1 0</td>
<td>6 5 4 3 2 1</td>
<td></td>
</tr>
</tbody>
</table>

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

4-neighbor grid

A*

- Second iteration of A*

<table>
<thead>
<tr>
<th>g-values</th>
<th>+</th>
<th>h-values</th>
<th>=</th>
<th>f-values</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2</td>
<td>6 5 4 3 2 1</td>
<td>5 4 3 2 1 0</td>
<td>6 5 4 3 2 1</td>
<td></td>
</tr>
<tr>
<td>6 5 4 3 2 1</td>
<td>5 4 3 2 1 0</td>
<td>0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

4-neighbor grid

A*

- Third iteration of A*

<table>
<thead>
<tr>
<th>g-values</th>
<th>+</th>
<th>h-values</th>
<th>=</th>
<th>f-values</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2</td>
<td>6 5 4 3 2 1</td>
<td>5 4 3 2 1 0</td>
<td>6 5 4 3 2 1</td>
<td></td>
</tr>
<tr>
<td>6 5 4 3 2 1</td>
<td>5 4 3 2 1 0</td>
<td>0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

4-neighbor grid

A*

- Fourth iteration of A*

<table>
<thead>
<tr>
<th>g-values</th>
<th>+</th>
<th>h-values</th>
<th>=</th>
<th>f-values</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2</td>
<td>6 5 4 3 2 1</td>
<td>5 4 3 2 1 0</td>
<td>6 5 4 3 2 1</td>
<td></td>
</tr>
<tr>
<td>6 5 4 3 2 1</td>
<td>5 4 3 2 1 0</td>
<td>0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

4-neighbor grid

A*

- Fifth iteration of A*

<table>
<thead>
<tr>
<th>g-values</th>
<th>+</th>
<th>h-values</th>
<th>=</th>
<th>f-values</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2</td>
<td>6 5 4 3 2 1</td>
<td>5 4 3 2 1 0</td>
<td>6 5 4 3 2 1</td>
<td></td>
</tr>
<tr>
<td>6 5 4 3 2 1</td>
<td>5 4 3 2 1 0</td>
<td>0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

4-neighbor grid
### 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**

### Any-Angle Path Planning

**A** on eight-neighbor grids

- **A** on eight-neighbor grids

---

**Manhattan Distance**

**Octile Distance**

**Zero h-values**

<table>
<thead>
<tr>
<th>Manhattan Distance</th>
<th>Octile Distance</th>
<th>Zero h-values</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 5 6</td>
<td>3 1</td>
<td>2</td>
</tr>
<tr>
<td>6 5 4</td>
<td>5 1</td>
<td>(7)</td>
</tr>
<tr>
<td>5 4 3</td>
<td>2</td>
<td>(8)</td>
</tr>
<tr>
<td>2 1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**4-neighbor grid**

More informed (dominating)
Any-Angle Path Planning

- \( A^* \) on the grid
- \( A^* \) with Post-Smoothing on the grid
- Field \( D^* \) on the grid
- Theta* on the grid
- \( A^* \) on the visibility graph
- any-angle path planning

Runtime
Path Length

Any-Angle Path Planning

- \( A^* \) on the grid
- \( A^* \) with Post-Smoothing on the grid
- Field \( D^* \) on the grid
- Theta* on the grid
- \( A^* \) on the visibility graph
- any-angle path planning

Runtime
Path Length

A* on eight-neighbor grids

- \( A^* \) 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

A* on other tessellations

- \( A^* \) on other tessellations

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

8-neighbor grid

Any-Angle Path Planning

- A* on eight-neighbor grids with smoothing

8-neighbor grid

Any-Angle Path Planning

- A* on eight-neighbor grids with smoothing

8-neighbor grid

Any-Angle Path Planning

- A* on eight-neighbor grids with smoothing

8-neighbor grid

Any-Angle Path Planning

- A* on eight-neighbor grids with smoothing

8-neighbor grid

Any-Angle Path Planning

- A* on eight-neighbor grids with smoothing

8-neighbor grid
Any-Angle Path Planning

- \( A^* \) on visibility graphs

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*

- Field D* on eight-neighbor grids

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

Terrain often has uniform movement costs.

<table>
<thead>
<tr>
<th></th>
<th>2.00</th>
<th>2.32</th>
<th>2.83</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>1.41</td>
<td>2.41</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>3.00</td>
<td>2.00</td>
<td>3.00</td>
</tr>
<tr>
<td>1.00</td>
<td>1.41</td>
<td>2.32</td>
<td>3.27</td>
</tr>
</tbody>
</table>

8-neighbor grid

[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.

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.
If path 2 is not blocked, then it is shorter than path 1 (triangle inequality)
Theta* does not necessarily find shortest paths since the parent of a state can only be a neighbor or the parent of a neighbor.
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

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

Any-Angle Path Planning

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*

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?

20m (megahertz) RAD6000 processor

Games [from Cavedog] Robotics [from JPL]

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

- Speeding up path planning with A*
  - Incremental versions of A* (incremental heuristic search)
    - Fringe Saving A* (FSA*)
    - Adaptive A* (AA*)
    - Lifelong Planning A* (LPA*), D* Lite and Minimax LPA*
    - Comparison of D* Lite and Adaptive A*
    - Eager and Lazy Moving Target Adaptive A* (MTAA*)
  - A* with weighted h-values
    - Weighted A* (WA*)
    - Anytime Repairing A* (ARA*)
  - Real-time versions of A* (real-time heuristic search)
    - Learning-Real Time A* (LRTA*)
    - Comparison of D* Lite and Learning-Real Time A*
    - Real-Time Adaptive A* (RTAA*)

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].

<table>
<thead>
<tr>
<th>search task 1</th>
<th>slightly different search task 2</th>
<th>slightly different search task 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>search task 1</td>
<td>slightly different search task 2</td>
<td>slightly different search task 3</td>
</tr>
</tbody>
</table>
Stationary Target

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

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

- Used in robotics and usable in games

[Stentz and Hebert, 1995] [from JPL] [from Cavedog Entertainment]

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

The worst-case number of movements is \( \Omega(\log(#\text{states})/\log\log(#\text{states}) \times #\text{states}) \) on undirected vertex-blocked graphs, where #\text{states} is the number of unblocked vertices [Koenig, Tovey and Smirnov, 2003].
Stationary Target

- The worst-case number of movements is $\Omega(\log(#\text{states})/\log \log(#\text{states}) \times #\text{states})$ on undirected vertex-blocked graphs, where $#\text{states}$ is the number of unblocked vertices [Koenig, Tovey and Smirnov, 2003].

- Proof:
  - Length of rim = $n^2$ 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 $\log^2(#\text{states})$ on undirected vertex-blocked graphs and $\log(#\text{states})$ on vertex-blocked grids, where $#\text{states}$ is the number of unblocked vertices [Mudgal, Tovey, Greenberg and Koenig, 2005].

---

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

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

---

Table of Contents

- Speeding up path planning with $A^*$
  - Incremental versions of $A^*$ (Incremental heuristic search)
    - Fringe Saving $A^*$ (FSA*)
    - Adaptive $A^*$ (AA*)
    - Lifelong Planning $A^*$ (LPA*), $D^*$ Lite and Minimax LPA*
    - Comparison of $D^*$ Lite and Adaptive $A^*$
    - Eager and Lazy Moving-Target Adaptive $A^*$ (MTAA*)
  - $A^*$ with weighted $h$-values
    - Weighted $A^*$ (WA*)
    - Anytime Repairing $A^*$ (ARA*)
  - Real-time versions of $A^*$ (real-time heuristic search)
    - Learning-Real Time $A^*$ (LRTA*)
    - Comparison of $D^*$ Lite and Learning-Real-Time $A^*$
    - Real-Time Adaptive $A^*$ (RTAA*)

---

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]
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].

Consider a state s that was expanded by A* with consistent h-values $h_{old}$:

- $\text{distance}(\text{start}, s) + \text{distance}(s, \text{goal}) \geq \text{distance}(\text{start}, \text{goal})$
- $\text{distance}(s, \text{goal}) \geq \text{distance}(\text{start}, \text{goal}) - \text{distance}(\text{start}, s)$
- $\text{distance}(s, \text{goal}) \geq f(\text{goal}) - g(s) = h_{new}(s)$

- 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.
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*)

- 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

Dynamic-SWSF-FP with early termination (our addition)
[Ramalingam and Reps, 1996]

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

Lifelong Planning A* (LPA*)

Uninformed search
- Breadth-first search

Heuristic search
- A* [Hart, Nilsson, Raphael, 1968]

4-neighbor grid

goal

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

[start]
**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

**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

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

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

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**

- If the agent moves from $s_{\text{old agent}}$ to $s_{\text{new agent}}$, then the goal of the search moves from $s_{\text{old agent}}$ to $s_{\text{new agent}}$. This changes the priorities of the states in the priority queue from $[\min(g(s), \text{rhs}(s)) + h(s_{\text{old agent}}, s), \min(g(s), \text{rhs}(s))]$ to $[\min(g(s), \text{rhs}(s)) + h(s_{\text{new agent}}, s), \min(g(s), \text{rhs}(s))]$ (but not which states are in the priority queue).
- Thus, one needs to reorder the priority queue [Stentz, 1994].
D* Lite

- D* Lite: Final Version [Koenig and Likhachev, 2002]
- One uses lower bounds on the new priorities instead of the new priorities themselves
  \[ \min(g(s), \text{rhs}(s)) + h(s_{\text{oldagent}}, s_{\text{newagent}}) \leq \min(g(s), \text{rhs}(s)) + h(s_{\text{oldagent}}, s_{\text{newagent}}) + h(s_{\text{newagent}}, s) \]
  \[ \min(g(s), \text{rhs}(s)) - h(s_{\text{oldagent}}, s_{\text{newagent}}) \leq \min(g(s), \text{rhs}(s)) + h(s_{\text{newagent}}, s) \]
- The term \( h(s_{\text{oldagent}}, s_{\text{newagent}}) \) 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.

Priority queue: A [8,5]; B [8,6]; C [8,7]
Agent moves: \( h(s_{\text{oldstart}}, s_{\text{newstart}}) = 2 \) (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]
Correct priority is [9,5]
Correct priority is [8,6]
Expand B

Minimax LPA*

- Cell decomposition methods

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

<table>
<thead>
<tr>
<th></th>
<th>Planning Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>uninformed search from scratch</td>
<td>363 minutes</td>
</tr>
<tr>
<td>informed search from scratch</td>
<td>135 minutes</td>
</tr>
<tr>
<td>uninformed incremental search</td>
<td>15 minutes</td>
</tr>
<tr>
<td>informed incremental search (Minimax LPA* [Likhachev and Koenig, 2003])</td>
<td>14 minutes</td>
</tr>
</tbody>
</table>

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].

Table of Contents

- Speeding up path planning with A*
  - Incremental versions of A* (incremental heuristic search)
    - Fringe Saving A* (FSA*)
    - Adaptive A* (AA*)
    - Lifelong Planning A* (LPA*), D* Lite and Minimax LPA*
    - Comparison of D* Lite and Adaptive A*
    - Eager and Lazy Moving-Target Adaptive A* (MTAA*)
  - A* with weighted h-values
    - Weighted A* (WA*)
    - Anytime Repairing A* (ARA*)
  - Real-time versions of A* (real-time heuristic search)
    - Learning-Real-Time A* (LRTA*)
    - Comparison of D* Lite and Learning-Real-Time A*
    - Real-Time Adaptive A* (RTAA*)
- A* with weighted h-values
  - Weighted A* (WA*)
  - Anytime Repairing A* (ARA*)
- Real-time versions of A* (real-time heuristic search)
  - Learning-Real-Time A* (LRTA*)
  - Comparison of D* Lite and Learning-Real-Time A*
  - Real-Time Adaptive A* (RTAA*)
- Transforming Greedy Mapping to Planning with the Freespace Assumption [Likhachev and Koenig, 2002]

D* Lite vs AA*

<table>
<thead>
<tr>
<th>D* Lite</th>
<th>AA*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapt previous search tree</td>
<td>Improve previous h-values</td>
</tr>
<tr>
<td>Start node must remain unchanged</td>
<td>Goal node must remain unchanged</td>
</tr>
<tr>
<td>Movement cost increases only*</td>
<td>Movement cost increases only*</td>
</tr>
<tr>
<td>Can result in more node expansions than A*</td>
<td>Guaranteed no more node expansions than A*</td>
</tr>
<tr>
<td>Fewer node expansions on average</td>
<td>More node expansions on average</td>
</tr>
<tr>
<td>Slow node expansions</td>
<td>Fast node expansions</td>
</tr>
</tbody>
</table>

*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

<table>
<thead>
<tr>
<th></th>
<th>expansions per search</th>
<th>runtime per search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward A*</td>
<td>3711</td>
<td>581</td>
</tr>
<tr>
<td>Backward A*</td>
<td>4104</td>
<td>644</td>
</tr>
<tr>
<td>(Forward) AA*</td>
<td>391</td>
<td>81</td>
</tr>
<tr>
<td>(Backward) D* Lite</td>
<td>31</td>
<td>15</td>
</tr>
</tbody>
</table>

Table of Contents

- Speeding up path planning with A*:
  - Incremental versions of A* (incremental heuristic search)
    - Fringe Saving A* (FSA*)
    - Adaptive A* (AA*)
    - Lifelong Planning A* (LPA*), D* Lite and Minimax LPA*
    - Comparison of D* Lite and Adaptive A*
    - Eager and Lazy Moving-Target Adaptive A* (MTAA*)
  - A* with weighted h-values:
    - Weighted A* (WA*)
    - Anytime Repairing A* (ARA*)
  - Real-time versions of A* (real-time heuristic search)
    - Learning-Real Time A* (LRTA*)
    - Comparison of D* Lite and Learning-Real-Time A*
    - Real-Time Adaptive A* (RTAA*)

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.
## D* Lite vs AA*

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

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

## D* Lite

![4-neighbor grid](image1)

![4-neighbor grid](image2)

![4-neighbor grid](image3)

![4-neighbor grid](image4)

4-neighbor grid  

4-neighbor grid  

4-neighbor grid  

4-neighbor grid  

*target-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

<table>
<thead>
<tr>
<th></th>
<th>expansions per search</th>
<th>runtime per search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward A*</td>
<td>3703</td>
<td>570</td>
</tr>
<tr>
<td>Backward A*</td>
<td>4519</td>
<td>722</td>
</tr>
<tr>
<td>Agent-Centric D* Lite</td>
<td>2229</td>
<td>1481</td>
</tr>
<tr>
<td>Target-Centric D* Lite</td>
<td>806</td>
<td>833</td>
</tr>
</tbody>
</table>

- 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

the map needs to get shifted
- a large number of blockages change
- changed blockages can be close to the start node
Eager Moving-Target Adaptive A*

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

Consider a state s after the goal changed:

\[
\text{distance}(s, \text{newgoal}) + h_{old}(\text{newgoal}) \geq h_{old}(s)
\]

\[
\text{distance}(s, \text{newgoal}) \geq h_{old}(s) - h_{old}(\text{newgoal})
\]

\[
\text{distance}(s, \text{newgoal}) \geq \max(h_{old}(s) - h_{old}(\text{newgoal}), h_{user}(s)) = h_{new}(s)
\]

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.

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 10th move
D* Lite vs MTAA*

<table>
<thead>
<tr>
<th></th>
<th>expansions per search</th>
<th>runtime per search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward A*</td>
<td>3703</td>
<td>570</td>
</tr>
<tr>
<td>Backward A*</td>
<td>4519</td>
<td>722</td>
</tr>
<tr>
<td>Forward Lazy MTAA*</td>
<td>2334</td>
<td>465</td>
</tr>
<tr>
<td>Backward Lazy MTAA*</td>
<td>2025</td>
<td>411</td>
</tr>
<tr>
<td>Agent-Centric D* Lite</td>
<td>2229</td>
<td>1481</td>
</tr>
<tr>
<td>Target-Centric D* Lite</td>
<td>806</td>
<td>833</td>
</tr>
</tbody>
</table>

Table of Contents

- Speeding up path planning with A*
  - Incremental versions of A* (incremental heuristic search)
    - Fringe Saving A* (FSA*)
    - Adaptive A* (AA*)
    - Lifelong Planning A* (LPA*), D* Lite and Minimax LPA*
    - Comparison of D* Lite and Adaptive A*
    - Eager and Lazy Moving Target Adaptive A* (MTAA*)
  - A* with weighted h-values
    - Weighted A* (WA*)
    - Anytime Repairing A* (ARA*)
  - Real-time versions of A* (real-time heuristic search)
    - Learning-Real Time A* (LRTA*)
    - Comparison of D* Lite and Learning-Real-Time A*
    - Real-Time Adaptive A* (RTAA*)

**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.

- 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***

```
  start
  \_____
  |    |
  |    |
  \_____
```

goal

A*    Weighted A*

```
  start
  \_____
  |    |
  |    |
  \_____
```

goal

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

- Speeding up path planning with A*
  - Incremental versions of A* (incremental heuristic search)
    - Fringe Saving A* (FSA*)
    - Adaptive A* (AA*)
    - Lifelong Planning A* (LPA*), D* Lite and Minimax LPA*
    - Comparison of D* Lite and Adaptive A*
    - Eager and Lazy Moving-Target Adaptive A* (MTAA*)
  - A* with weighted h-values
    - Weighted A* (WA*)
    - Anytime Repairing A* (ARA*)
  - Real-time versions of A* (real-time heuristic search)
    - Learning-Real Time A* (LRTA*)
    - Comparison of D* Lite and Learning-Real-Time A*
    - Real-Time Adaptive A* (RTAA*)

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)

\[
\begin{align*}
  w &= 2.5 & 13 \text{ expansions} & 11 \text{ movements} \\
  w &= 1.5 & 15 \text{ expansions} & 11 \text{ movements} \\
  w &= 1.0 & 20 \text{ expansions} & 10 \text{ movements}
\end{align*}
\]

[8-neighbor grid [from Maxim Likhachev]]

\[
\begin{align*}
  w &= 2.5 & 13 \text{ expansions} & 11 \text{ movements} \\
  w &= 1.5 & 1 \text{ expansion} & 11 \text{ movements} \\
  w &= 1.0 & 9 \text{ expansions} & 10 \text{ movements}
\end{align*}
\]

[8-neighbor grid [from Maxim Likhachev]]

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

[4d search with A* [from Maxim Likhachev]]
Table of Contents

- Speeding up path planning with A*  
  - Incremental versions of A* (incremental heuristic search)  
    - Fringe Saving A* (FSA*)  
    - Adaptive A* (AA*)  
    - Lifelong Planning A* (LPA*), D* Lite and Minimax LPA*  
    - Comparison of D* Lite and Adaptive A*  
    - Eager and Lazy Moving-Target Adaptive A* (MTAA*)
- A* with weighted h-values  
  - Weighted A* (WA*)  
  - Anytime Repairing A* (ARA*)  
- Real-time versions of A* (real-time heuristic search)  
  - Learning-Real-Time A* (LRTA*)  
  - Comparison of D* Lite and Learning-Real-Time A*  
  - Real-Time Adaptive A* (RTAA*)

Learning Real-Time A* (LRTA*)

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

Learning Real-Time A* (LRTA*)

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

local minima are a problem

Learning Real-Time A* (LRTA*)

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

4-neighbor grid

local minima are overcome by updating the h-values

- 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*)

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

4-neighbor grid

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 \times \text{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*)

- 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?

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*)

Step 1: Forward A* search

First A* state expansion

4-neighbor grid

Step 1: Forward A* search

Second A* state expansion

4-neighbor grid

Step 1: Forward A* search

Third A* state expansion

4-neighbor grid
Learning Real-Time A* (LRTA*)

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

4-neighbor grid

first iteration of Dijkstra's algorithm

Learning Real-Time A* (LRTA*)

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

second iteration of Dijkstra's algorithm

Learning Real-Time A* (LRTA*)

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

third iteration of Dijkstra's algorithm

Learning Real-Time A* (LRTA*)

- Step 3: Moving along the path

follow the path
Learning Real-Time A* (LRTA*)

Step 3: Moving along the path

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

Safely explorable random grids of size 301 x 301

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

<table>
<thead>
<tr>
<th>lookahead</th>
<th>Manhattan distance</th>
<th>octile distance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>planning time</td>
<td>path length</td>
</tr>
<tr>
<td>1</td>
<td>28280</td>
<td>499</td>
</tr>
<tr>
<td>11</td>
<td>28698</td>
<td>315</td>
</tr>
<tr>
<td>21</td>
<td>29153</td>
<td>302</td>
</tr>
<tr>
<td>31</td>
<td>29615</td>
<td>299</td>
</tr>
<tr>
<td>41</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Lookahead</th>
<th>Manhattan Distance</th>
<th>Octile Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Planning Path</td>
<td>Path</td>
</tr>
<tr>
<td>1</td>
<td>985362</td>
<td>1987574</td>
</tr>
<tr>
<td>11</td>
<td>313998</td>
<td>337704</td>
</tr>
<tr>
<td>21</td>
<td>279856</td>
<td>205370</td>
</tr>
<tr>
<td>31</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>41</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Lookahead</th>
<th>LRTA* with A*</th>
<th>LRTA* with BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>State Exp.</td>
<td>Path Length</td>
</tr>
<tr>
<td>1</td>
<td>1259958</td>
<td>1259958</td>
</tr>
<tr>
<td>5</td>
<td>765645</td>
<td>477525</td>
</tr>
<tr>
<td>11</td>
<td>531955</td>
<td>272842</td>
</tr>
<tr>
<td>15</td>
<td>517913</td>
<td>239073</td>
</tr>
<tr>
<td>21</td>
<td>459566</td>
<td>177143</td>
</tr>
<tr>
<td>25</td>
<td>456752</td>
<td>155736</td>
</tr>
</tbody>
</table>

Table of Contents

- Speeding up path planning with A*
  - Incremental versions of A* (incremental heuristic search)
    - Fringe Saving A* (FSA*)
    - Adaptive A* (AA*)
    - Lifelong Planning A* (LPA*), D* Lite and Minimax LPA*
    - Comparison of D* Lite and Adaptive A*
  - Eager and Lazy Moving-Target Adaptive A* (MTAA*)
- A* with weighted h-values
  - Weighted A* (WA*)
  - Anytime Repairing A* (ARA*)
- Real-time versions of A* (real-time heuristic search)
  - Learning Real-Time A* (LRTA*)
  - Comparison of D* Lite and Learning-Real-Time A*
  - Real-Time Adaptive A* (RTAA*)
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 \(O(\#\text{states} \log \#\text{states})\)

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

### LRTA* vs D* Lite

<table>
<thead>
<tr>
<th>lookahead</th>
<th>Manhattan distance</th>
<th>octile distance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>planning time</td>
<td>path length</td>
</tr>
<tr>
<td>D* Lite</td>
<td>36826</td>
<td>309</td>
</tr>
<tr>
<td>1</td>
<td>28280</td>
<td>499</td>
</tr>
<tr>
<td>11</td>
<td>28698</td>
<td>315</td>
</tr>
<tr>
<td>21</td>
<td>29153</td>
<td>302</td>
</tr>
<tr>
<td>31</td>
<td>29615</td>
<td>299</td>
</tr>
<tr>
<td>41</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### LRTA* vs D* Lite

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

<table>
<thead>
<tr>
<th>range of (x) for LRTA*</th>
<th>optimal lookahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^4.00-10^6.09</td>
<td>1</td>
</tr>
<tr>
<td>10^0.08-10^0.14</td>
<td>3</td>
</tr>
<tr>
<td>10^0.15-10^1.06</td>
<td>5</td>
</tr>
<tr>
<td>10^1.07-10^1.07</td>
<td>7</td>
</tr>
</tbody>
</table>

### LRTA* vs D* Lite

- Safely explorable random grids of size 301 x 301

- Grids with 25% Random Obstacles
  - h-values generally misleading
  - larger lookaheads less helpful

- 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

<table>
<thead>
<tr>
<th>lookahead</th>
<th>Manhattan distance</th>
<th>octile distance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>planning time</td>
<td>path length</td>
</tr>
<tr>
<td>D* Lite</td>
<td>357417</td>
<td>21738</td>
</tr>
<tr>
<td>1</td>
<td>985362</td>
<td>1987574</td>
</tr>
<tr>
<td>11</td>
<td>313998</td>
<td>337704</td>
</tr>
<tr>
<td>21</td>
<td>279856</td>
<td>205370</td>
</tr>
<tr>
<td>31</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>41</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
LRTA* vs D* Lite

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

<table>
<thead>
<tr>
<th>range of x for LRTA*</th>
<th>optimal lookahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^{-4.00} - 10^{-0.31}</td>
<td>21</td>
</tr>
<tr>
<td>10^{-0.30} - 10^{-0.16}</td>
<td>25</td>
</tr>
<tr>
<td>10^{-0.15} - 10^{-0.09}</td>
<td>33</td>
</tr>
</tbody>
</table>

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

Real-Time Adaptive A* (RTAA*)

- We use AA* to create Real-Time Adaptive A* (RTAA*)

Learning Real-Time A* (LRTA*)

Comparison of D* Lite and Learning-Real-Time A*

Real-Time Adaptive A* (RTAA*)
Real-Time Adaptive A* (RTAA*)

- LRTA* step 1: forward A* search

4-neighbor grid
Real-Time Adaptive A* (RTAA*)

- LRTA* step 1: forward A* search

![4-neighbor grid](image1)

- LRTA* step 2: updating the h-values

![4-neighbor grid](image2)
Real-Time Adaptive A* (RTAA*)

LRTA* step 2: updating the h-values

4-neighbor grid
Properties of LRTA* \[\text{Korf, 1990}\]

- 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

4-neighbor grid

bold = g-value
regular = h-value
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 3 2 1 2
2 1 0 1 2
```

4-neighbor grid

**bold** = g-value  
**regular** = h-value

- **RTAA* step 2:** updating the h-values

```
state about to be expanded
f-value = 8
```

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

4-neighbor grid

**bold** = g-value  
**regular** = h-value
Real-Time Adaptive A* (RTAA*)

- RTAA* step 3: moving along the path

4-neighbor grid

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*
- LRTA*

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

- Safely explorable mazes of size 151 x 151

Real-Time Adaptive A* (RTAA*)

<table>
<thead>
<tr>
<th></th>
<th>RTAA*</th>
<th>LRTA*</th>
</tr>
</thead>
<tbody>
<tr>
<td>expansions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trajectory length</td>
<td></td>
<td></td>
</tr>
<tr>
<td>time per search</td>
<td>ms</td>
<td>ms</td>
</tr>
<tr>
<td>1</td>
<td>248538</td>
<td>248538</td>
</tr>
<tr>
<td>9</td>
<td>104229</td>
<td>56708</td>
</tr>
<tr>
<td>17</td>
<td>85866</td>
<td>33853</td>
</tr>
<tr>
<td>25</td>
<td>89258</td>
<td>26338</td>
</tr>
<tr>
<td>33</td>
<td>96840</td>
<td>22022</td>
</tr>
<tr>
<td>41</td>
<td>105703</td>
<td>18629</td>
</tr>
<tr>
<td>49</td>
<td>117036</td>
<td>16638</td>
</tr>
<tr>
<td>57</td>
<td>128560</td>
<td>15367</td>
</tr>
</tbody>
</table>

4-neighbor grid

Real-Time Adaptive A* (RTAA*)

Relationship of RTAA* and LRTA*

<table>
<thead>
<tr>
<th></th>
<th>RTAA*</th>
<th>LRTA*</th>
</tr>
</thead>
<tbody>
<tr>
<td>expansions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trajectory length</td>
<td></td>
<td></td>
</tr>
<tr>
<td>time per search</td>
<td>ms</td>
<td>ms</td>
</tr>
<tr>
<td>1</td>
<td>248538</td>
<td>248538</td>
</tr>
<tr>
<td>9</td>
<td>104229</td>
<td>56708</td>
</tr>
<tr>
<td>17</td>
<td>85866</td>
<td>33853</td>
</tr>
<tr>
<td>25</td>
<td>89258</td>
<td>26338</td>
</tr>
<tr>
<td>33</td>
<td>96840</td>
<td>22022</td>
</tr>
<tr>
<td>41</td>
<td>105703</td>
<td>18629</td>
</tr>
<tr>
<td>49</td>
<td>117036</td>
<td>16638</td>
</tr>
<tr>
<td>57</td>
<td>128560</td>
<td>15367</td>
</tr>
</tbody>
</table>

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