Greedy On-Line Planning

Sven Koenig

http://www.cc.gatech.edu/fac/Sven.Koenig/

Collaborators:
Craig Tovey, Maxim Likhachev,
David Furcy, Yaxin Liu, Yuri Smirnov
(Additional Programming: Colin Bauer, William Halliburton)

Part 1: greedy on-line planning makes planning tractable
   example: greedy localization

Part 2: greedy on-line planning is reactive to the current situation
   (plus other advantages)
   example: greedy mapping
   example: moving a robot to goal coordinates in unknown terrain

Part 3: fast replanning for greedy on-line planning
   example: replanning of shortest paths
   example: moving a robot to goal coordinates in unknown terrain
   example: greedy mapping
   example: symbolic planning
   heuristic search-based replanning
   calculating the heuristics for heuristic search-based planning

Nondeterministic Planning - The Problem

Planning in nondeterministic domains is time consuming
due to the many contingencies

Nondeterministic Planning - A Solution

Agent-Centered Search [Koenig; 2001]

Planning in nondeterministic domains is time consuming
due to the many contingencies
Agent-centered search makes it more efficient by
interleaving planning with limited lookahead and plan execution

State space can even become deterministic
**Nondeterministic Planning - A Solution**

Agent-Centered Search

Planning in nondeterministic domains is time consuming due to the many contingencies. Agent-centered search makes it more efficient by interleaving planning with limited lookahead and plan execution. Agent-centered search makes it more efficient by interleaving planning with limited lookahead and plan execution.

![Diagram of state space becoming deterministic]

State space can even become deterministic.

**Nondeterministic Planning - Another Solution**

Assumption-Based Planning

Planning in nondeterministic domains is time consuming due to the many contingencies. Assumption-based planning makes it more efficient by making assumptions about the outcomes of action executions.

![Diagram of desired trajectory vs. actual trajectory]

State space can even become deterministic.

**Nondeterministic Planning: Greedy On-Line Planning**

Both agent-centered search and assumption-based planning are greedy planning methods because they make simplifying assumptions to make planning tractable. On-line planning methods because they interleave planning and plan execution.

Note: without additional assumptions, it is not guaranteed that greedy on-line planning methods achieve the goal!

**Nondeterministic Planning: Robot Navigation under Incomplete Information**

Sensor-Based Planning [Choset and Burdick, 1994]

Robot knows the map but not its location
- Localization

Robot knows its location but not the map
- Mapping
- Goal-directed navigation in unknown terrain

Both agent-centered search and assumption-based planning are greedy planning methods because they make simplifying assumptions to make planning tractable.
Part 1

Greedy On-line Planning makes Planning Tractable

Greedy Localization

The robot is always in exactly one cell.*
The robot has a compass on board.
The robot has no sensor or actuator uncertainty and knows the map.
The robot initially does not know where it is.

* We also have results for continuous terrain that are similar to the ones presented in the following for discretized terrain.

Hardness of (Approximately) Optimal Localization

Theorem [Tovey and Koenig, 2000]

It is in NP to determine whether there exists a valid localization plan that executes no more movements than a given value.

It is NP-hard to find a localization plan in gridworlds of size $m \times n$ whose worst-case number of movements to localization is within a factor $O(\log(mn))$ of optimum, even in connected gridworlds in which localization is possible.

To prove the theorem, we reduce set cover problems to our localization problems.

Set Cover

- number of elements
- number of sets
- number of sets that form a smallest set cover

Theorem

It is NP-hard to find a set cover whose number of sets is within a factor $O(\log(x))$ of optimum (for sufficiently small constants). [Lund and Yannakakis, 1994]
Consider the following localization plan: Find the closest signature (= gives the robot its current column). Then move into all vertical corridors that correspond to a smallest set cover (= gives the robot its current row).

The number of movements of this localization plan is at most $3y^*xy$.

Thus, the number of movements of an optimal localization plan is at most $3y^*xy$.

Thus, the number of movements of a localization plan whose worst-case number of movements to localization is within a factor $O(\log(mn))$ of optimum is at most $O(\log(mn)) 3y^*xy \leq O(3x^3y)$.

Thus, such a plan cannot leave its current east-west corridor and can only localize by moving into all corridors that correspond to a set cover. Let $y'$ denote the cardinality of this set cover. Then, the number of movements is at least $(2y'-1)(xy-x-1)$.

Thus, the number of movements is at least $(2y'-1)(xy-x-1)$ and at most $O(\log(x)) 3y^*xy$, implying that $y' = O(\log(x)) y'$ and thus that the set cover is within a factor $O(\log(x))$ of minimum.

However, it is NP-hard to find a set cover whose number of sets is within a factor $O(\log(x))$ of minimum.

**Theorem** [Tovey and Koenig, 2000]

For every gridworld of size $m \times n$, there exists a valid localization plan that executes $O(mn)$ movements to localization and that can be found in time $O(mn)$.

This result is the best possible in the sense that there exist gridworlds of size $m \times n$ in which every valid localization plan must execute $\Omega(mn)$ movements to localization and can only be found in time $\Omega(mn)$. 
Greedy Localization

Greedy Localization repeatedly makes the robot execute a shortest (deterministic) movement sequence (subplan) that is guaranteed to reduce the number of possible robot cells by at least one.

[Genesereth and Nourbakhsh, 1993][Koenig and Simmons, 1998]

Greedy localization uses new information right away.

Note: Assume localization is possible. The state space is safely explorable. Greedy Localization always achieves a gain in information. Thus, Greedy Localization localizes the robot.
**Theorem**

The planning and plan-execution times of Greedy Localization are guaranteed to be low-order polynomials in the size of the gridworld.

**Cost of (Approximately) Optimal Localization**

Greedy Localization makes planning tractable. Greedy Localization is fast in practice.

**Random Acyclic Mazes**

<table>
<thead>
<tr>
<th>gridworld size</th>
<th>obstacle density</th>
<th>av. number of subplans to localization</th>
<th>av. number of steps per subplan to localization</th>
<th>av. total number of movements to localization</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 x 11</td>
<td>41.3 %</td>
<td>2.4 x 1.5</td>
<td></td>
<td>3.6</td>
</tr>
<tr>
<td>21 x 21</td>
<td>45.4 %</td>
<td>3.3 x 1.7</td>
<td></td>
<td>5.4</td>
</tr>
<tr>
<td>31 x 31</td>
<td>46.8 %</td>
<td>3.8 x 1.7</td>
<td></td>
<td>6.6</td>
</tr>
<tr>
<td>41 x 41</td>
<td>47.6 %</td>
<td>4.1 x 1.8</td>
<td></td>
<td>7.5</td>
</tr>
<tr>
<td>51 x 51</td>
<td>48.1 %</td>
<td>4.5 x 1.8</td>
<td></td>
<td>8.0</td>
</tr>
<tr>
<td>61 x 61</td>
<td>48.4 %</td>
<td>4.7 x 1.8</td>
<td></td>
<td>8.6</td>
</tr>
<tr>
<td>71 x 71</td>
<td>48.6 %</td>
<td>4.9 x 1.9</td>
<td></td>
<td>9.1</td>
</tr>
</tbody>
</table>

(5041 cells)

Example for a Corridor-Like Terrain [Tovey and Koenig, 2000]

The worst-case number of movements of Greedy Localization can be a factor $\Omega(\sqrt{mn})$ worse than the optimal worst-case number of movements to localization in gridworlds of size $m \times n$, even in connected gridworlds in which localization is possible.
Our Acyclic Mazes

<table>
<thead>
<tr>
<th>gridworld size</th>
<th>obstacle density</th>
<th>av. number of subplans to localization</th>
<th>av. number of steps per subplan to localization</th>
<th>av. total number of movements to localization</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 x 25</td>
<td>50.2 %</td>
<td>4.5 x 2.3</td>
<td>10.2</td>
<td></td>
</tr>
<tr>
<td>13 x 36</td>
<td>50.2 %</td>
<td>5.9 x 2.9</td>
<td>16.9</td>
<td></td>
</tr>
<tr>
<td>15 x 49</td>
<td>50.2 %</td>
<td>7.4 x 3.2</td>
<td>23.7</td>
<td></td>
</tr>
<tr>
<td>17 x 64</td>
<td>50.2 %</td>
<td>8.9 x 3.4</td>
<td>30.6</td>
<td></td>
</tr>
<tr>
<td>19 x 81</td>
<td>50.2 %</td>
<td>10.4 x 4.0</td>
<td>42.0</td>
<td></td>
</tr>
<tr>
<td>21 x 100</td>
<td>50.1 %</td>
<td>11.5 x 4.4</td>
<td>50.0</td>
<td></td>
</tr>
<tr>
<td>23 x 121</td>
<td>50.1 %</td>
<td>13.4 x 4.5</td>
<td>60.4</td>
<td></td>
</tr>
<tr>
<td>25 x 144</td>
<td>50.1 %</td>
<td>14.4 x 4.9</td>
<td>71.1</td>
<td></td>
</tr>
<tr>
<td>27 x 169</td>
<td>50.1 %</td>
<td>16.0 x 5.2</td>
<td>82.5 (4563 cells)</td>
<td></td>
</tr>
<tr>
<td>29 x 196</td>
<td>50.1 %</td>
<td>18.0 x 5.4</td>
<td>98.0 (5684 cells)</td>
<td></td>
</tr>
</tbody>
</table>

The worst-case number of movements of Greedy Localization can be a factor $\Omega((mn)/(\log(mn)))$ worse than the optimal worst-case number of movements to localization in gridworlds of size $m \times n$, even in connected gridworlds in which localization is possible.

* We also have even better lower bounds (although in more complex gridworlds) and small upper bounds.

Example for a Room-Like Terrain*

However, its plan-execution time cannot be optimal.

Cost of (Approximately) Optimal Localization

<table>
<thead>
<tr>
<th>planning time</th>
<th>(Approximately) Optimal Localization</th>
<th>Greedy Localization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(likely) exponential</td>
<td>low-order polynomial</td>
</tr>
<tr>
<td></td>
<td>low-order polynomial</td>
<td>low-order polynomial</td>
</tr>
</tbody>
</table>
Extension: Actuator and Sensor Noise

so far:
no sensor uncertainty, no actuator uncertainty
minimax model

more realistic on robots:
sensor uncertainty, actuator uncertainty
probabilistic model
POMDP-based ("Markov") Localization

Mobile robots have
- noisy actuators
- noisy sensors

no sensor uncertainty, no actuator uncertainty
minimax model

more realistic on robots:
sensor uncertainty, actuator uncertainty
probabilistic model
POMDP-based ("Markov") Localization

Landmark-Based Navigation
“sensitive to the environment”
be sensitive to both the environment and the robot movements
maintain a probability distribution over all locations (location distribution)

Metric-Based Navigation
“sensitive to robot movements”
restrict location distributions,
but don’t discretize the locations
allow arbitrary location distributions

POMDP-Based Navigation on Xavier
operated for three years with > 200 km travel distance
now very popular with large amount of follow-up work

Landmark-Based Navigation
“sensitive to the environment”
be sensitive to both the environment and the robot movements
maintain a probability distribution over all locations (location distribution)

Metric-Based Navigation
“sensitive to robot movements”
restrict location distributions,
but don’t discretize the locations
allow arbitrary location distributions

POMDP-Based Navigation on Xavier
operated for three years with > 200 km travel distance
now very popular with large amount of follow-up work

[Simmons and Koenig, 1995]
[Thrun, 2000]
[Simmons and Koenig, 1995]
POMDP-based ("Markov") Localization
- uniform, theoretically grounded framework for localization
- maintains arbitrary probability distributions over the locations
- explicitly models all uncertainty using probabilities
- utilizes all available sensor data (landmarks, robot movements)
- robust towards sensor errors (no explicit exception handling required)

Greedy Localization repeatedly makes the robot execute a shortest (deterministic) movement sequence (subplan) that is guaranteed to reduce the number of possible robot cells by at least one.

It is PSPACE-hard to find an optimal policy for a POMDP. [Papadimitriou, Tsitsiklis, 1987]

Greedy Localization repeatedly makes the robot execute a shortest (deterministic) movement sequence (subplan) that is guaranteed to reduce the entropy of the probability distribution over the possible robot cells.

[Burgard, Fox, Thrun, 1997]
Greedy On-Line Planning is Reactive to the Current Situation (plus other advantages)

Greedy Mapping = Agent-Centered Search

Greedy Mapping always moves the robot on a shortest path to closest unobserved (or unvisited) cell.

Thus, it plans in the deterministic part of the nondeterministic state space until a plan is found that achieves a gain in information.

Note: Assume mapping is possible. The state space is safely explorable. Greedy Mapping always achieves a gain in information. Thus, Greedy Mapping maps the terrain.

Greedy Mapping - Advantages

can easily be integrated into robot architectures (“reactive planning”)

for example, our implementation combines greedy mapping and schema-based navigation (MissionLab) [Mackenzie, Arkin, Cameron, 1997]

does not need to be in control of the robot at all times (“reactive planning”)
Greedy Mapping - Advantages

we assume here that the robot can move in eight directions

utilizes prior map knowledge, if available

can be used by multiple robots that share their maps

---

Greedy Mapping - Robot Implementation

---

Greedy Mapping - Travel Distance

---

Greedy Mapping - Robot Implementation

---
Greedy Mapping - Travel Distance

Here: Greedy Mapping always moves the robot on a shortest path to the closest unvisited cell. This version of Greedy Mapping works on any strongly connected undirected graph.

The worst-case number of movements of Greedy Mapping is $\Omega(s)$ and $O(s^2)$, where $s$ is the number vertices of the graph, even for undirected planar graphs.

Theorem: [Koenig, Tovey, Smirnov, 2001]

More Interesting Theorem

The worst-case number of movements of Greedy Mapping is $\Omega(\log_s)$ and $O(v\log s)$, where $s$ is the number vertices of the graph, even for undirected planar graphs.
Greedy Mapping - Travel Distance

<table>
<thead>
<tr>
<th>n</th>
<th>travel distance</th>
<th></th>
<th>travel distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>207</td>
<td>80</td>
<td>2.59</td>
</tr>
<tr>
<td>4</td>
<td>2279</td>
<td>778</td>
<td>2.93</td>
</tr>
<tr>
<td>5</td>
<td>31253</td>
<td>9612</td>
<td>3.25</td>
</tr>
<tr>
<td>6</td>
<td>515085</td>
<td>144014</td>
<td>3.58</td>
</tr>
<tr>
<td>7</td>
<td>9928271</td>
<td>2542528</td>
<td>3.90</td>
</tr>
<tr>
<td>8</td>
<td>219130987</td>
<td>51744018</td>
<td>4.23</td>
</tr>
<tr>
<td>9</td>
<td>5448100629</td>
<td>1195201300</td>
<td>4.57</td>
</tr>
<tr>
<td>10</td>
<td>150617283953</td>
<td>30753086422</td>
<td>4.90</td>
</tr>
</tbody>
</table>

**Greedy Mapping - Travel Distance**

<table>
<thead>
<tr>
<th>order of</th>
<th>upper bound for Greedy Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>travel distance</td>
</tr>
<tr>
<td></td>
<td>log</td>
</tr>
<tr>
<td></td>
<td>log</td>
</tr>
<tr>
<td></td>
<td>log</td>
</tr>
<tr>
<td></td>
<td>order of</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Planning with the Freespace Assumption

Planning with the Freespace Assumption always moves the robot on a shortest potentially unblocked path to the goal cell.

Planning with the Freespace Assumption = Assumption-Based Planning

Planning with the Freespace Assumption always moves the robot on a shortest potentially unblocked path to the goal cell.

Thus, it makes assumptions about outcomes of actions that make the nondeterministic state space deterministic.

![Diagram of HMMWV](image)

HMMWV that navigated 1,410 meters of natural outdoor terrain in 1995

[Stentz and Hebert, 1995]

Note: Assume moving to the goal is possible. The state space is safely explorable.

Planning with the Freespace Assumption always achieves a gain in information.

Thus, Planning with the Freespace Assumption moves to the goal.

![Diagram of trajectory](image)
Freespace Assumption - Travel Distance

Here: Planning with the Freespace Assumption always moves the robot on a shortest (potentially unblocked) path to the goal vertex.

Planning with the Freespace Assumption results in small travel distances if the freespace assumption is approximately satisfied, that is, if the obstacle density is small.

However, the travel distances are also small if the freespace assumption is not satisfied.
Freespace Assumption - Travel Distance

Here: Planning with the Freespace Assumption always moves the robot on a shortest (potentially unblocked) path to the goal vertex.

Theorem: [Koenig, Tovey, Smirnov, 2001]*

The worst-case number of movements of Planning with the Freespace Assumption is \( \Omega \left( \frac{s}{\log \log s} \right) \) and \( O(s^{3/2}) \), where \( s \) is the number vertices of the graph, even for undirected planar graphs.

* we also have even better bounds

---

Part 3

Fast Replanning for Greedy On-line Planning

---

Greedy Mapping - Implementation

we assume here that the robot can move in eight directions

Greedy Mapping always moves the robot on a shortest path to the closest unobserved (or unvisited) cell.

---

Freespace Assumption - Implementation

we assume here that the robot can move in eight directions

Planning with the Freespace Assumption always moves the robot on a shortest potentially unblocked path to the goal cell.
Path Planning - Example
we assume here that the robot can move in eight directions

original eight-connected gridworld

Path Planning - Example
we assume here that the robot can move in eight directions

changed eight-connected gridworld
Path Planning - Example

Artificial Intelligence

Heuristic Search

Algorithm Theory

Incremental Search

how to search efficiently using heuristic to guide the search

how to search efficiently by reusing information from previous searches

Path Planning - Lifelong Planning A*

<table>
<thead>
<tr>
<th>uninformed search</th>
<th>heuristic search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-First Search</td>
<td>A* \cite{Ramalingam, Reps, 1996}</td>
</tr>
</tbody>
</table>

Path Planning - Experimental Evaluation

original eight-connected gridworld

<table>
<thead>
<tr>
<th>uninformed search</th>
<th>heuristic search</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete search</td>
<td>Lifelong Planning A*</td>
</tr>
</tbody>
</table>

Path Planning - Experimental Evaluation

changed eight-connected gridworld

<table>
<thead>
<tr>
<th>uninformed search</th>
<th>heuristic search</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete search</td>
<td>Lifelong Planning A*</td>
</tr>
</tbody>
</table>

\[ \text{Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.} \]
Path Planning - Experimental Evaluation

changed eight-connected gridworld - first implementation

uninformed search

heuristic search

(with the same tie-breaking as LPA*)

<table>
<thead>
<tr>
<th>ve</th>
<th>va</th>
<th>hp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1331.7</td>
<td>26207.2</td>
<td>5985.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ve</th>
<th>va</th>
<th>hp</th>
</tr>
</thead>
<tbody>
<tr>
<td>284.0</td>
<td>6177.3</td>
<td>1697.3</td>
</tr>
</tbody>
</table>

Lifelong Planning A*

<table>
<thead>
<tr>
<th>ve</th>
<th>va</th>
<th>hp</th>
</tr>
</thead>
<tbody>
<tr>
<td>173.0</td>
<td>5697.4</td>
<td>956.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ve</th>
<th>va</th>
<th>hp</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.6</td>
<td>1235.9</td>
<td>240.1</td>
</tr>
</tbody>
</table>

ve = vertex expansions, va = vertex accesses, hp = heap percolates

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.  SA3 - 69 of 141

Path Planning - Experimental Evaluation

changed eight-connected gridworld - second implementation

uninformed search

heuristic search

(with the same tie-breaking as LPA*)

<table>
<thead>
<tr>
<th>ve</th>
<th>va</th>
<th>hp</th>
</tr>
</thead>
<tbody>
<tr>
<td>801.76</td>
<td>2359.60</td>
<td>561.48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ve</th>
<th>va</th>
<th>hp</th>
</tr>
</thead>
<tbody>
<tr>
<td>115.95</td>
<td>561.48</td>
<td>182.15</td>
</tr>
</tbody>
</table>

Lifelong Planning A*

<table>
<thead>
<tr>
<th>ve</th>
<th>va</th>
<th>hp</th>
</tr>
</thead>
<tbody>
<tr>
<td>172.20</td>
<td>724.60</td>
<td>18.80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ve</th>
<th>va</th>
<th>hp</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.66</td>
<td>18.80</td>
<td>39.9</td>
</tr>
</tbody>
</table>

ve = vertex expansions, hp = heap percolates

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.  SA3 - 71 of 141

Path Planning - Lifelong Planning A*

[Koenig, Likhachev, 2001]

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

procedure Initialize()
U = Ø; for all s ∈ S rhs(s) = g(s) = ∞; rhs(s\text{ start }) = 0; U.Insert(s\text{ start }, CalculateKey(s\text{ start }));

procedure UpdateVertex(u)
if (u ≠ s\text{ start }) rhs(u) = \min s' ∈ Pred(u) (g(s')+c(s',u));
if (u ∈ U) U.Remove(u);
if (g(u) ≠ rhs(u)) U.Insert(u, CalculateKey(u));

procedure ComputeShortestPath()
while (U.TopKey < CalculateKey(s\text{ goal }) OR rhs(s\text{ goal }) ≠ g(s\text{ goal }));
U.Pop();
g(u) = rhs(u);
for all s ∈ Succ(u) UpdateVertex(s);
else
g(u) = ∞;
for all s ∈ Succ(u) \cup \{u\} UpdateVertex(s);

procedure Main()
Initialize();
forever
ComputeShortestPath();
Wait for changes in edge costs;
for all directed edges (u, v) with changed edge costs
Update the edge cost c(u,v);
UpdateVertex(v);

U.TopKey() returns the smallest priority of all vertices in the priority queue U.
If U is empty, then U.TopKey() returns \([∞; ∞]\).
U.Pop() deletes the vertex with the smallest priority in priority queue U and
returns the vertex. U.Insert(s,k) inserts vertex s into priority queue U with
priority k. Finally, U.Remove(s) removes vertex s from priority queue U.

The heuristics need to be nonnegative and (forward) consistent:
\[h(s,s\text{ goal }) ≤ h(s',s\text{ goal }) + c(s,s') \leq h(s,s') + h(s',s\text{ goal })\]
for all vertices s ∈ S and s' ∈ Succ(s).

This version of LPA* can be optimized further without changing its overall operation.
We also have versions of LPA* that:
- break ties differently
- work with inconsistent heuristics
- terminate earlier
- contain several runtime optimizations.

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.  SA3 - 77 of 141
Lifelong Planning A*:
- applies to the same finite search problems as A*
- handles arbitrary edge cost changes
- produces the same (optimal) solution as A*
- is algorithmically very similar to A*
- is more efficient than A* in many situations
- has nice theoretical properties
- applies to
  - route planning problems (traffic, networking, ...)
  - robot control
  - symbolic artificial intelligence planning
  - ...

The priority queue contains exactly the locally inconsistent vertices \( s \) their priority is \([\min(g(s),rhs(s))+h(s,s_{goal})]; \min(g(s),rhs(s))]\) smaller priorities first, according to a lexicographic ordering.

\[ g\text{-value} = rhs\text{-value}: \text{cell is locally consistent} \]
\[ g\text{-value} \neq rhs\text{-value}: \text{cell is locally inconsistent} \]
\[ g\text{-value} > rhs\text{-value}: \text{cell is locally overconsistent} \]
\[ g\text{-value} < rhs\text{-value}: \text{cell is locally underconsistent} \]
**Path Planning - Lifelong Planning A***

**start**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

goal

**priority queue**

C3: [4;2]

**Path Planning - Lifelong Planning A***

**start**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

goal

**priority queue**

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

**Path Planning - Lifelong Planning A***

**start**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

goal

**priority queue**

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

**Path Planning - Lifelong Planning A***

**start**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

goal

**priority queue**

D4: [6;4]; D3: [6;5]; D2: [6;6]
Theorem: [Likhachev and Koenig, 2001]

ComputeShortestPath() expands every vertex at most twice and thus terminates.

Theorem: [Likhachev and Koenig, 2001]

After ComputeShortestPath() terminates, one can trace back a shortest path from the start to the goal by always moving from the current vertex s, starting at the goal, to any predecessor s' that minimizes \(g(s') + c(s',s)\) until the start is reached (ties can be broken arbitrarily).
**Path Planning - Lifelong Planning A**

In the worst case, replanning cannot be more efficient than planning from scratch. [Nebel, Koehler, 1995]

![Diagram](image1)

**Theorem:** [Likhachev and Koenig, 2001]

ComputeShortestPath() does not expand any vertices whose g-values were equal to their respective start distances before ComputeShortestPath() was called.

= LPA* is efficient because it uses incremental search

**Theorem:** [Likhachev and Koenig, 2001]

ComputeShortestPath() expands at most those vertices \( s \) with \( [f(s); g^*(s)] \) ≤ \( [f(s_{start}); g^*(s_{start})] \) or \( [g_{old}(s) + h(s); g_{old}(s)] \) ≤ \( [f(s_{start}); g^*(s_{start})] \), where \( f(s) = g^*(s) + h(s) \) and \( g_{old}(s) \) is the g-value of \( s \) directly before the call to ComputeShortestPath().

= LPA* is efficient because it uses heuristic search

**Path Planning - Lifelong Planning A**

The first search of Lifelong Planning A* is the same as that of A*. Afterwards, Lifelong Planning A* operates in a very similar way to A*. (The theorem makes this more concrete. For example, ComputeShortestPath() expands locally overconsistent vertices with finite f-values in the same order as A*.)

![Diagram](image2)

**Freespace Assumption - Implementation**

we assume here that the robot can move in eight directions

Planning with the Freespace Assumption always moves the robot on a shortest potentially unblocked path to the goal cell.
Transforming Planning with the Freespace Assumption to Path Planning

here: search from the goal location towards the robot location
- allows one to reuse parts of the search tree after the robot has moved
- allows one to use heuristics to focus the search
  (this additional argument holds for Greedy Mapping later)

\[
h(s_{\text{start}}, s) = \text{approximation of the distance from the robot to vertex } s\]
\[
g(s) = \text{approximation of the goal distance of vertex } s\]

Assumption to Path Planning

- \(s_{\text{start}}\) is the start location
- \(s_{\text{goal}}\) is the goal location

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

Freespace Assumption - D* Lite (Basic Version)

\[\text{procedure } \text{CalculateKey}(s)\]
\[\text{return } [\min(g(s), \text{rhs}(s)) + h(s_{\text{start}}, s), \min(g(s), \text{rhs}(s))];\]
\[\text{procedure } \text{Initialize}()\]
\[U = \emptyset;\]
for all \(s \in S\) \(\text{rhs}(s) = g(s) = \infty;\)
\[\text{U.Insert}(s_{\text{goal}}, \text{CalculateKey}(s_{\text{goal}}));\]
\[\text{procedure } \text{UpdateVertex}(u)\]
if \(u \neq s_{\text{goal}}\) \(\text{rhs}(u) = \min_{s' \in \text{Succ}(u)} (\text{rhs}(s') + g(s'));\)
if \((u \in U)\) \(\text{U.Remove}(u);\)
if \(g(s) = \text{rhs}(s)\) \(\text{U.Insert}(s, \text{CalculateKey}(s));\]
\[\text{procedure } \text{ComputeShortestPath}()\]
while \((U.\text{TopKey} < \text{CalculateKey}(s_{\text{start}}) \text{ OR rhs}(s_{\text{start}}) \neq g(s_{\text{start}}))\)
\[u = U.\text{Pop}();\]
if \(g(s) > \text{rhs}(s)\)
\[g(s) = \text{rhs}(s);\]
for all \(s' \in \text{Pred}(u)\) \(\text{UpdateVertex}(s);\)
else
\[g(s) = \infty;\]
for all \(s' \in \text{Pred}(u) \cup \{u\}\) \(\text{UpdateVertex}(s);\]
\[\text{procedure } \text{Main}()\]
\[\text{Initialize}();\]
\[\text{ComputeShortestPath}();\]

Freese Assumption - D* Lite (Basic Version)

\[\text{Idea}\]

When the robot moves, the goal of the search (\(s_{\text{start}}\)) moves.
This influences the priorities of the vertices in the priority queue
(but not which vertices are in the priority queue).

\[\text{vertex } s \text{ is locally inconsistent iff}\]
\[\text{vertex } s \text{ is in the priority queue with priority } [\min(g(s), \text{rhs}(s)) + h(s_{\text{start}}, s); \min(g(s), \text{rhs}(s))].\]
\[h(s_{\text{newstart}}, s)\]
\[\text{This value changes when the robot moves from } s_{\text{oldstart}} \text{ to } s_{\text{newstart}.}\]

Thus, one needs to reorder the priority queue. [Stentz, 1994]

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.
Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

**Freespace Assumption - D* Lite (Basic Version)**

**Fictitious Example**

priority queue A: [8;5]; B: [8;6]; C: [8;7]

priority queue C: [7;7]; B: [8;6]; A: [9;5]

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

**Freespace Assumption - D* Lite (Final Version)**

**Idea**

Reordering the priority queue is time consuming.

vertex s is locally inconsistent iff vertex s is in the priority queue with priority \([\min(g(s),\text{rhs}(s)) + h(s_{\text{oldstart}},s); \min(g(s),\text{rhs}(s))].\]

We use lower bounds on the new priorities instead of the new priorities themselves.

\[
\begin{align*}
\min(g(s),\text{rhs}(s)) + h(s_{\text{oldstart}},s) - h(\text{newstart},s) & \leq \min(g(s),\text{rhs}(s)) + h(s_{\text{oldstart}}-\text{newstart}), h(\text{newstart},s) + h(s_{\text{newstart}},s) - h(s_{\text{oldstart}},s) \\
\min(g(s),\text{rhs}(s)) + h(s_{\text{oldstart}},s) - h(\text{newstart},s) & \leq \min(g(s),\text{rhs}(s)) + h(\text{newstart},s) - h(s_{\text{newstart}},s) - h(s_{\text{oldstart}},s)
\end{align*}
\]

The term \(h(s_{\text{oldstart}}-\text{newstart})\) is the same across vertices in the priority queue. Instead of deleting it from all vertices in the priority queue, we add it to the vertices added to the priority queue in the future. [Stentz, 1995]

When ComputeShortestPath() selects a vertex for expansion, it checks first whether its priority is correct. If so, it expands the vertex. If it is a lower bound, it calculates the correct priority and reinserts the vertex into the queue.

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

**Freespace Assumption - D* Lite (Final Version)**

**Fictitious Example**

priority queue A: [8;5]; B: [8;6]; C: [8;7]

add vertex D with priority [10;5]

priority queue A: [8;5]; B: [8;6]; C: [8;7]

add vertex D with priority [12;5]

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

The heuristics need to be nonnegative and forward-backward consistent:

\[h(s,s') \leq h(s,s)+h(s',s'')\]

for all vertices \(s,s',s'' \in S\).

The heuristics also need to be admissible no matter what the goal vertex is:

\[h(s,s') \leq \text{shortest distance from } s \text{ to } s'\]

for all vertices \(s,s' \in S\).

\[[Koenig, Likhachev, 2002]\]

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

**The heuristics need to be nonnegative and forward-backward consistent:**

\[h(s,s') \leq h(s,s)+h(s',s'')\]

for all vertices \(s,s',s'' \in S\).

**The heuristics also need to be admissible no matter what the goal vertex is:**

\[h(s,s') \leq \text{shortest distance from } s \text{ to } s'\]

for all vertices \(s,s' \in S\).

\[[Stentz, 1995]\]

We use lower bounds on the new priorities instead of the new priorities themselves.

\[\min(g(s),\text{rhs}(s)) + h(s_{\text{oldstart}},s) - h(s_{\text{newstart}},s) \leq \min(g(s),\text{rhs}(s)) + h(s_{\text{oldstart}}-\text{newstart}), h(\text{newstart},s) + h(s_{\text{newstart}},s) - h(s_{\text{oldstart}},s)\]

The term \(h(s_{\text{oldstart}}-\text{newstart})\) is the same across vertices in the priority queue.

Instead of deleting it from all vertices in the priority queue, we add it to the vertices added to the priority queue in the future. [Stentz, 1995]

When ComputeShortestPath() selects a vertex for expansion, it checks first whether its priority is correct. If so, it expands the vertex. If it is a lower bound, it calculates the correct priority and reinserts the vertex into the queue.

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

**The heuristics need to be nonnegative and forward-backward consistent:**

\[h(s,s') \leq h(s,s)+h(s',s'')\]

for all vertices \(s,s',s'' \in S\).

**The heuristics also need to be admissible no matter what the goal vertex is:**

\[h(s,s') \leq \text{shortest distance from } s \text{ to } s'\]

for all vertices \(s,s' \in S\).

\[[Stentz, 1995]\]

We use lower bounds on the new priorities instead of the new priorities themselves.

\[\min(g(s),\text{rhs}(s)) + h(s_{\text{oldstart}},s) - h(s_{\text{newstart}},s) \leq \min(g(s),\text{rhs}(s)) + h(s_{\text{oldstart}}-\text{newstart}), h(\text{newstart},s) + h(s_{\text{newstart}},s) - h(s_{\text{oldstart}},s)\]

The term \(h(s_{\text{oldstart}}-\text{newstart})\) is the same across vertices in the priority queue.

Instead of deleting it from all vertices in the priority queue, we add it to the vertices added to the priority queue in the future. [Stentz, 1995]

When ComputeShortestPath() selects a vertex for expansion, it checks first whether its priority is correct. If so, it expands the vertex. If it is a lower bound, it calculates the correct priority and reinserts the vertex into the queue.

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

**The heuristics need to be nonnegative and forward-backward consistent:**

\[h(s,s') \leq h(s,s)+h(s',s'')\]

for all vertices \(s,s',s'' \in S\).

**The heuristics also need to be admissible no matter what the goal vertex is:**

\[h(s,s') \leq \text{shortest distance from } s \text{ to } s'\]

for all vertices \(s,s' \in S\).

\[[Stentz, 1995]\]

We use lower bounds on the new priorities instead of the new priorities themselves.

\[\min(g(s),\text{rhs}(s)) + h(s_{\text{oldstart}},s) - h(s_{\text{newstart}},s) \leq \min(g(s),\text{rhs}(s)) + h(s_{\text{oldstart}}-\text{newstart}), h(\text{newstart},s) + h(s_{\text{newstart}},s) - h(s_{\text{oldstart}},s)\]

The term \(h(s_{\text{oldstart}}-\text{newstart})\) is the same across vertices in the priority queue.

Instead of deleting it from all vertices in the priority queue, we add it to the vertices added to the priority queue in the future. [Stentz, 1995]

When ComputeShortestPath() selects a vertex for expansion, it checks first whether its priority is correct. If so, it expands the vertex. If it is a lower bound, it calculates the correct priority and reinserts the vertex into the queue.

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.
Freespace Assumption - D* Lite

we assume here that the robot can move in eight directions

knowledge before the movement sequence of the robot

knowledge after the movement sequence of the robot
Freespace Assumption - D* Lite

A = overhead of D* Lite without incremental Search (A*)
B = overhead of D* Lite without heuristic search

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

Freespace Assumption - D* Lite

overhead of Focussed D* = probably the first truly incremental heuristic search method
(note: Focussed D* is likely a bit faster than D* Lite per vertex expansion)

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

Greedy Mapping - Implementation

Greedy Mapping always moves the robot on a shortest path to closest unobserved (or unvisited) cell.

Transforming Greedy Mapping to Planning with the Freespace Assumption

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.
Greedy Mapping - D* Lite

we assume here that the robot can move in eight directions

knowledge before the movement sequence of the robot

<table>
<thead>
<tr>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

knowledge after the movement sequence of the robot

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
Greedy Mapping - D* Lite

\[ A = \text{overhead of D* Lite without incremental Search (A*)} \]
\[ B = \text{overhead of D* Lite without heuristic search} \]

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

Other Examples of Lifelong Planning

- replanning (and plan reuse) is important!
- world changes over time
- model of the world changes over time
- what-if analyses

- emergency management

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

Other Examples of Lifelong Planning

- mobile robotics
- mapping
- goal-directed navigation in unknown terrain

- route planning
- in traffic networks
- in computer networks

- computer games
- symbolic planning (with HSP)
- continual planning
- one-time planning
- reinforcement learning and on-line dynamic programming
- control (with the Parti-Game algorithm)

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.
Game Playing

Symbolic Planning (with HSP) - Continual Planning
- plan adaptation
- repair-based planning
- learning search control knowledge
- case-based planning
- transformational planning
- iterative repair methods in scheduling

CHEF, GORDIUS, LS-ADJUST-PLAN, MRL, NoLimit, PLEXUS, PRIAR, SPA...

plan quality of replanning is usually worse than plan quality of planning from scratch

- lifelong planning

SHERPA
plan quality of replanning is as good as plan quality of planning from scratch

Symbolic Planning (with HSP) - Continual Planning
SHERPA
Speedy HEuristic search-based RePlAnner
[S. Koenig, D. Furcy, C. Bauer, 2002]

planning problem 1  planning problem 2  planning problem 3

...  ...  ...

note: in the following, we consider only finding shortest plans

Symbolic Planning (with HSP) - Continual Planning
STIRPS-type planning in the elevator domain

Operators:
- The elevator moves from floor $f_i$ to floor $f_j$ with $i \neq j$.
- Person $p_k$ boards the elevator on floor $f_i$ provided that the elevator is currently on floor $f_i$ and floor $f_i$ is the origin of person $p_k$.
- Person $p_k$ gets off the elevator on floor $f_i$, provided that person $p_k$ is in the elevator, the elevator is currently on floor $f_i$, and floor $r_i$ is the destination of person $p_k$. 
Symbolic Planning (with HSP) - Continual Planning

first search in the elevator domain using SHERPA

similar to HSP 2.0 with the $h_{\text{max}}$ heuristic

[Bonet, Geffner, 2001]

![Diagram showing search in the elevator domain with SHERPA]

second search in the elevator domain using SHERPA from scratch

similar to HSP 2.0 with the $h_{\text{max}}$ heuristic

[Bonet, Geffner, 2001]

second search in the elevator domain using SHERPA

![Diagram showing search in the elevator domain with SHERPA]

SHERPA achieves speedups up to 80 percent

![Graph showing savings percentage vs. number of people]

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.  SA3 - 117 of 141
Symbolic Planning (with HSP) - Continual Planning

PINCH
Prioritized, INCremental Heuristics calculation

$Liu, Koenig, Furcy, 2002$

tens of thousands of calculations of heuristic values for each planning problem
**Symbolic Planning (with HSP) - One-Time Planning**

**PINCH**

Prioritized, INCremental Heuristics calculation

Here: for HSP 2.0 with the \( h_{\text{add}} \) heuristic \([\text{Bonet, Geffner, 2001}]\)

\[
\begin{align*}
    h_{\text{add}}(\text{state}) &= \sum_{\text{proposition in goal state}} g_{\text{state}}(\text{proposition}) \\
    g_{\text{state}}(\text{proposition}) &= \begin{cases} 
        0 & \text{if proposition in state} \\
        \min_{\text{operator with proposition in add list}} (1 + g_{\text{state}}(\text{operator})) & \text{otherwise}
    \end{cases} \\
    g_{\text{state}}(\text{operator}) &= \sum_{\text{proposition on precondition list of operator}} g_{\text{state}}(\text{proposition})
\end{align*}
\]

Order of state expansions:

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

**Symbolic Planning (with HSP) - One-Time Planning**

PINCH achieves speedups up to (another!) 80 percent.

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.

**Reinforcement Learning and On-Line DP**

While there exists at least one state with \( g(s) = \text{rhs}(s) \)

- Pick a state \( s \) with \( g(s) = \text{rhs}(s) \) and then set \( g(s) := \text{rhs}(s) \)

**Prioritized Sweeping** \([\text{Moore and Atkeson, 1993}]\)

- Chooses the g-value of which state to update
- Updates the g-value of the chosen state in a particular way
- Minimizes the expected or worst-case plan-execution cost for MDPs

**Minimax LPA***

- Chooses the g-value of which state to update
- Updates the g-value of the chosen state in a particular way
- Terminates immediate once a shortest path is found
- Uses heuristics to focus the search
- Minimizes the worst-case plan-execution cost for MDPs

And so on, for a total of 22 g-value updates. Minimax LPA* needs only 6.

Note: Minimax LPA* expands every state at most twice.
Control (with the Parti-Game algorithm)

state spaces of control problems are often continuous and sometimes high-dimensional

coarse-grained discretization might not be able to find a plan fine-grained discretization is very inefficient

Control (with the Parti-Game algorithm)

Parti-Game algorithm [Moore and Atkeson; 1995]

nonuniform discretization avoids these problems

Control (with the Parti-Game algorithm)

here: using a deterministic state space for illustration

Control (with the Parti-Game algorithm)

the state space is really nondeterministic we thus use Minimax LPA* instead of LPA*
Control (with the Parti-Game algorithm)

terrains of size 2000 x 2000

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Planning Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uninformed Search from Scratch</td>
<td>362 minutes 55 seconds</td>
</tr>
<tr>
<td>Informed Search from Scratch</td>
<td>135 minutes 15 seconds</td>
</tr>
<tr>
<td>Uninformed Incremental Search</td>
<td>14 minutes 53 seconds</td>
</tr>
<tr>
<td>Informed Incremental Search (Minimax LPA*)</td>
<td>13 minutes 53 seconds</td>
</tr>
</tbody>
</table>

References (in order of their appearance)

C. Tovey and S. Koenig, Gridworlds as Testbeds for Planning with Incomplete Information, Proceedings of the National Conference on Artificial Intelligence, 819-824, 2000.

References (in order of their appearance)

Greedy On-Line Planning and Lifelong Planning

Related Work:

- and many more

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.  SA3 - 137 of 141

Greedy On-Line Planning and Lifelong Planning

Artificial Intelligence

Algorithm Theory

Related Work:

- and many more

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.  SA3 - 138 of 141

Greedy On-Line Planning and Lifelong Planning

Robotics

Theoretical Results

Related Work:

- and many more

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.  SA3 - 139 of 141

Greedy On-Line Planning and Lifelong Planning

Theoretical Results

Related Work:

- and many more

Greedy On-Line Planning; (c) Sven Koenig; Georgia Tech; January 2002.  SA3 - 140 of 141
Lifelong Planning Techniques - Our Work

Please see
http://www.cc.gatech.edu/fac/Sven.Koenig/greedyonline

We gratefully acknowledge funding from NSF and IBM.

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