Greedy On-Line Planning

Sven Koenig

Georgia Tech College of Computing

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)

Greedy On-Line Planning

- abstract overview: what is greedy on-line planning?

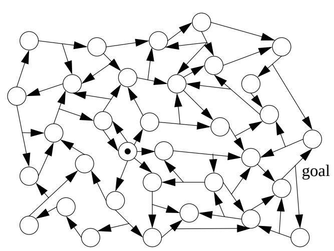
Part 1: - greedy on-line planning makes planning tractable
    example: greedy localization

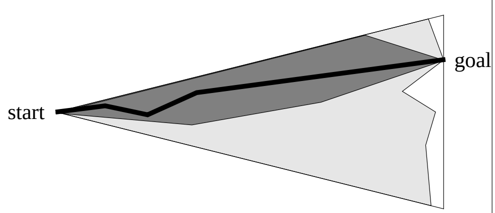
Part 2: - greedy on-line planning is reactive to the current situation (plus other advantages)
    example: greedy mapping
    example: moving a robot to goal coordinates in unknown terrain

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



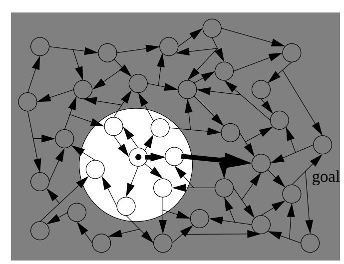


Nondeterministic Planning - A Solution

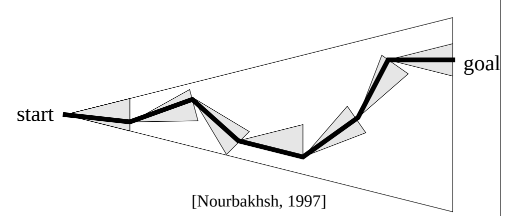
Agent-Centered Search [Koenig; 2001]

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



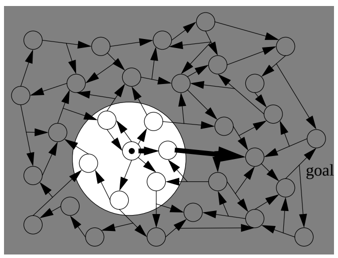
state space can even become
deterministic



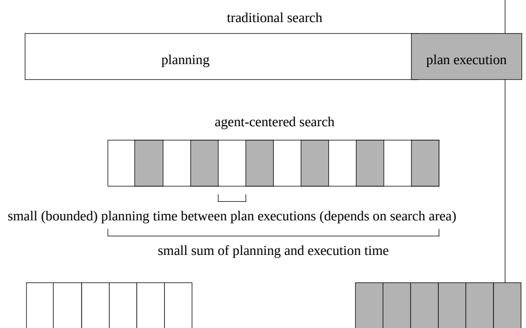
Nondeterministic Planning - A Solution
Agent-Centered Search

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



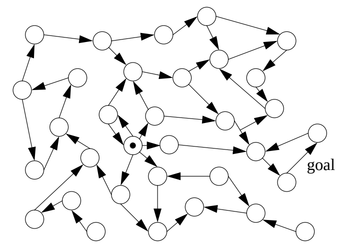
state space can even become
deterministic



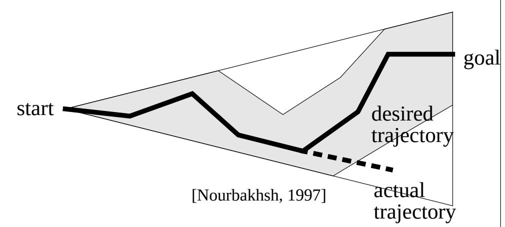
Nondeterministic Planning - Another Solution
Assumption-Based Planning

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



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]

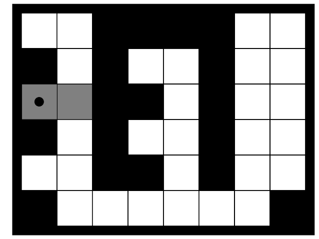
robot knows the map but not its location
- localization

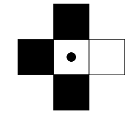
robot knows its location but not the map
- mapping
- goal-directed navigation in unknown terrain

Part 1

Greedy On-line Planning makes Planning Tractable

Greedy Localization





short-range sensor
discretized space

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.
The robot can move to one of the four adjacent empty cells.
The robot always senses which of the four adjacent cells is empty.

The task of the robot is to find out where it is with a shortest travel distance in the worst case (that is, for the worst possible start location) or detect that this is impossible. (Example: 5 moves)

* 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

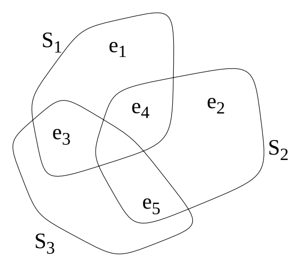
Theorem [Tovey and Koenig, 2000]

> 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.

contrast with: [Dudek, Romanik, Whitesides, 1995]

Hardness of (Approximately) Optimal Localization

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



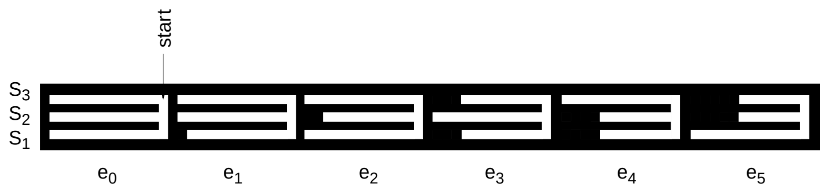
Set Cover

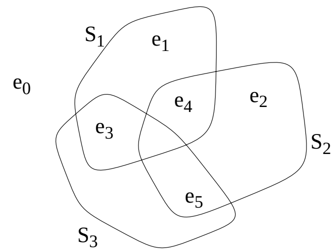
number of elements	$x = 5$
number of sets	$y = 3$
number of sets that form a smallest set cover	$y^* = 2$

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]

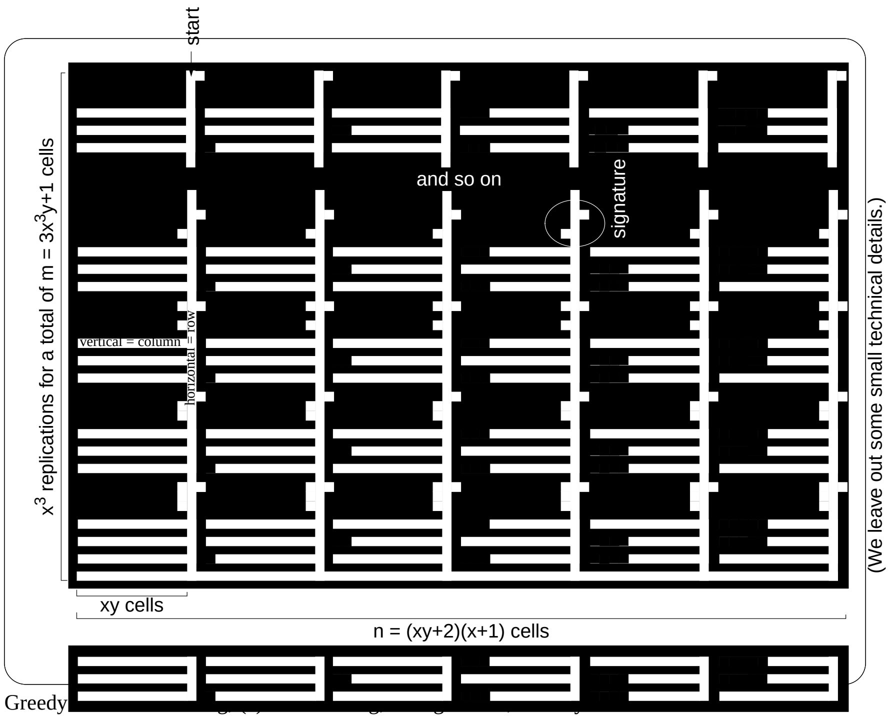
Hardness of (Approximately) Optimal Localization





Whenever $e_i \in S_j$,
we make the corresponding
horizonal corridor i cells shorter.

To localize,
the robot has to visit
all the horizontal corridors
that correspond to a set cover.



Hardness of (Approximately) Optimal Localization

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)) $3y^*xy$ = O(log(x)) $3y^*xy \leq O(3x^3y)$.

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.

qed

Cost of ~~(Approximately) Optimal~~ Localization

Theorem [Tovey and Koenig, 2000]

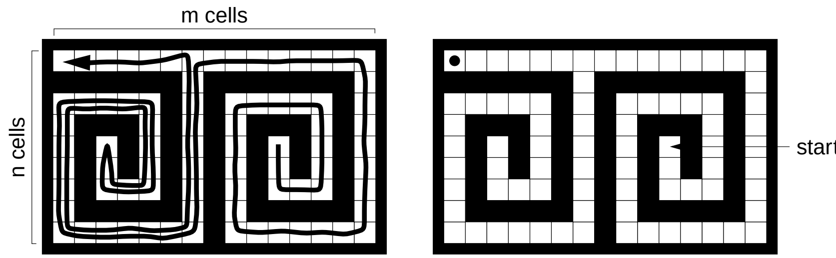
For every gridworld of size $m \times n$, there exists a valid localization plan that executes $O(mn)$ movements to localization and that can be found in time $O(mn)$.

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 $\Omega(mn)$.

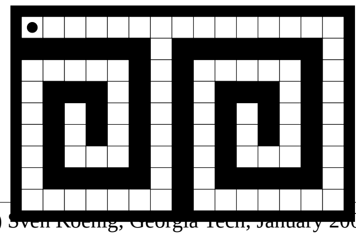
Cost of ~~(Approximately) Optimal~~ Localization

Map and Robot Trajectory

Knowledge of the Robot

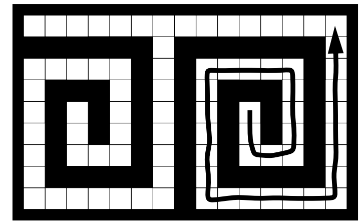


Matching the Map and Knowledge of the Robot



qed

Cost of ~~(Approximately) Optimal~~ Localization

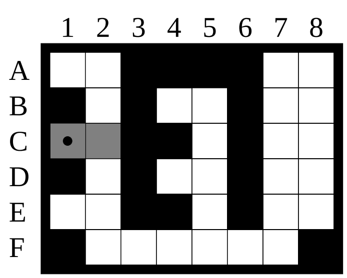


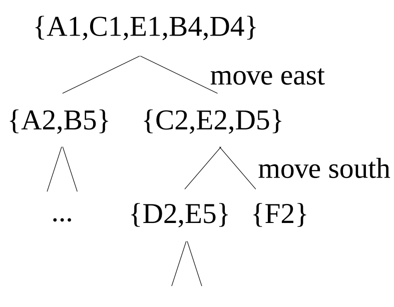
Greedy Localization

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.

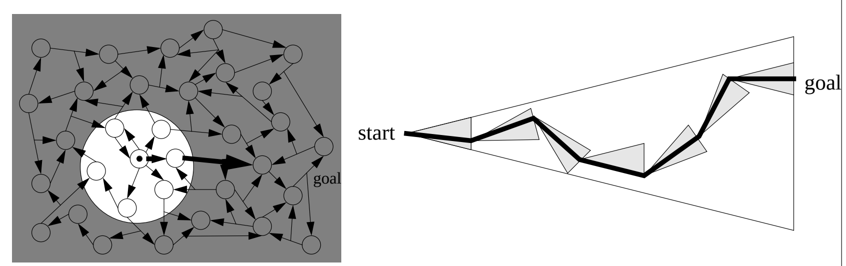




Greedy Localization = Agent-Centered Search

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.

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

Cost of ~~(Approximately) Optimal~~ Greedy Localization

Greedy Localization makes planning tractable.

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~~ Greedy Localization

Greedy Localization is fast in practice.

Random Acyclic Mazes

gridworld size	obstacle density	av. number of subplans to localization		av. number of steps per subplan to localization		av. total number of movements to localization	
11 x 11	41.3 %	2.4	x	1.5	=	3.6	
21 x 21	45.4 %	3.3	x	1.7	=	5.4	
31 x 31	46.8 %	3.8	x	1.7	=	6.6	
41 x 41	47.6 %	4.1	x	1.8	=	7.5	
51 x 51	48.1 %	4.5	x	1.8	=	8.0	
61 x 61	48.4 %	4.7	x	1.8	=	8.6	
71 x 71	48.6 %	4.9	x	1.9	=	9.1	(5041 cells)

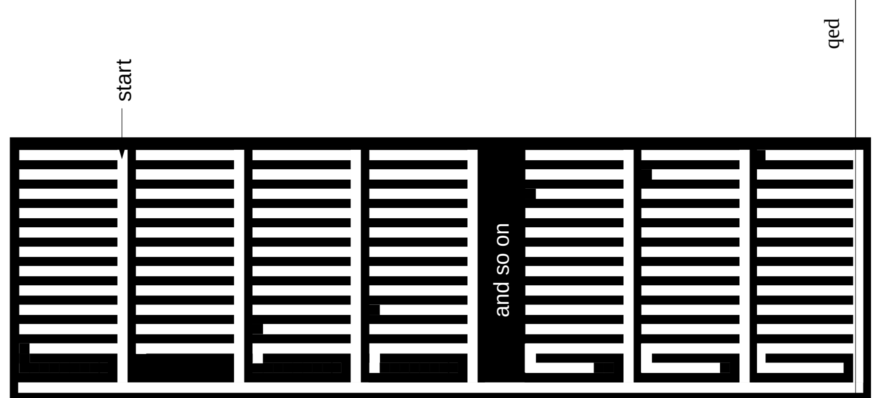
Cost of ~~(Approximately) Optimal~~ Greedy Localization

However, its plan-execution time cannot be optimal.

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

> The worst-case number of movements of Greedy Localization can be a factor $\Omega(\sqrt[3]{mn})$ 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.

Cost of ~~(Approximately) Optimal~~ Greedy Localization



Cost of ~~(Approximately) Optimal~~ Greedy Localization

Our Acyclic Mazes

gridworld size	obstacle density	av. number of subplans to localization		av. number of steps per subplan to localization		av. total number of movements to localization	
11 x 25	50.2 %	4.5	x	2.3	=	10.2	
13 x 36	50.2 %	5.9	x	2.9	=	16.9	
15 x 49	50.2 %	7.4	x	3.2	=	23.7	
17 x 64	50.2 %	8.9	x	3.4	=	30.6	
19 x 81	50.2 %	10.4	x	4.0	=	42.0	
21 x 100	50.1 %	11.5	x	4.4	=	50.0	
23 x 121	50.1 %	13.4	x	4.5	=	60.4	
25 x 144	50.1 %	14.4	x	4.9	=	71.1	
27 x 169	50.1 %	16.0	x	5.2	=	82.5	(4563 cells)
29 x 196	50.1 %	18.0	x	5.4	=	98.0	(5684 cells)
31 x 225	50.1 %	19.4	x	5.7	=	110.5	
33 x 256	50.1 %	20.8	x	5.8	=	121.5	
35 x 289	50.1 %	22.5	x	6.1	=	137.7	

Cost of ~~(Approximately) Optimal~~ Greedy Localization

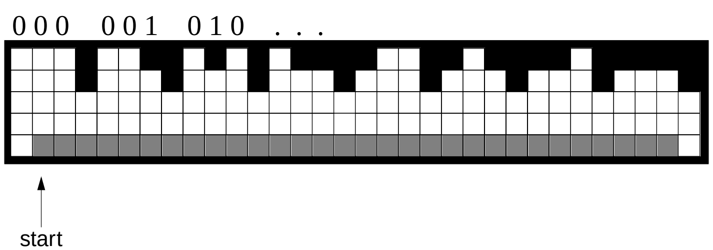
However, its plan-execution time cannot be optimal.

Example for a Room-Like Terrain*

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.

Cost of ~~(Approximately) Optimal~~ Greedy Localization



qed

Cost of ~~(Approximately) Optimal~~ Greedy Localization Summary

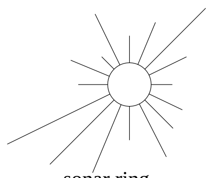
	(Approximately) Optimal Localization	Greedy Localization
planning time	(likely) exponential	low-order polynomial
plan-execution time	low-order polynomial	low-order polynomial

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


sonar ring


occupancy grid

Extension: Actuator and Sensor Noise

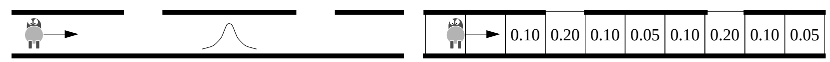
Landmark-Based Navigation
"sensitive to the environment"

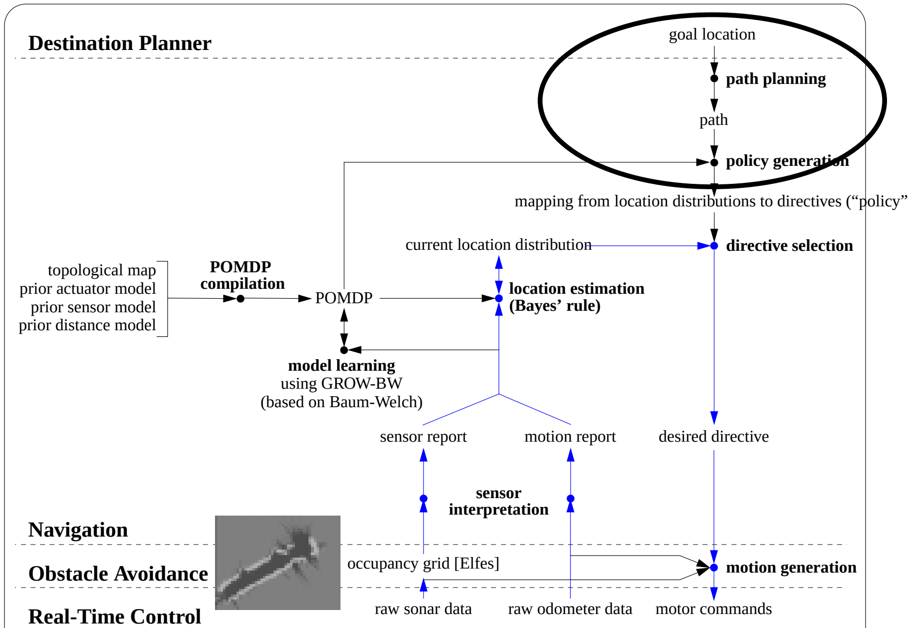
Metric-Based Navigation
"sensitive to robot movements"

be sensitive to both the environment and the robot movements
+
maintain a probability distribution over all locations (location distribution)

Kalman Filters
restrict location distributions,
but don't discretize the locations

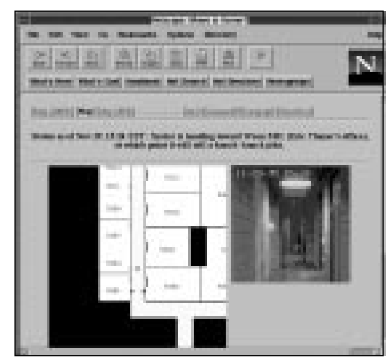
POMDPs
(Partially Observable Markov Decision Process Models)
discretize the locations, but
allow arbitrary location distributions





Extension: Actuator and Sensor Noise

Xavier



POMDP-Based Navigation on Xavier [Simmons and Koenig, 1995] [Koenig and Simmons, 1998]

operated for three years with > 200 km travel distance

now very popular with large amount of follow-up work [Thrun, 2000]

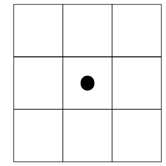
Extension: Actuator and Sensor Noise

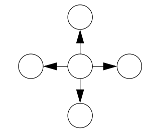
POMDP-based (“Markov”) Localization

- 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)

Extension: Actuator and Sensor Noise

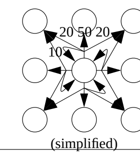
no sensor uncertainty, no actuator uncertainty
minimax model



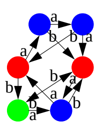


sensor uncertainty, actuator uncertainty
probabilistic model
POMDP-based (“Markov”) Localization

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1



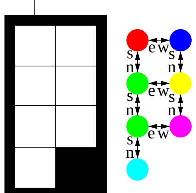
Extension: Actuator and Sensor Noise



no sensor uncertainty, no actuator uncertainty
minimax model

It is NP-hard to find an optimal homing sequence
for a colored finite state automaton. [Schapire, 1992]

add more structure
the robot can only move north, east, south, or west



It is NP-hard to find an optimal localization sequence
in a gridworld.

sensor uncertainty, actuator uncertainty
probabilistic model
POMDP-based (“Markov”) Localization

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

add more structure
the robot can only move north, east, south, or west

?????

Extension: Actuator and Sensor Noise

no sensor uncertainty, no actuator uncertainty
minimax model

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.

sensor uncertainty, actuator uncertainty
probabilistic model
POMDP-based (“Markov”) Localization

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]

Part 2

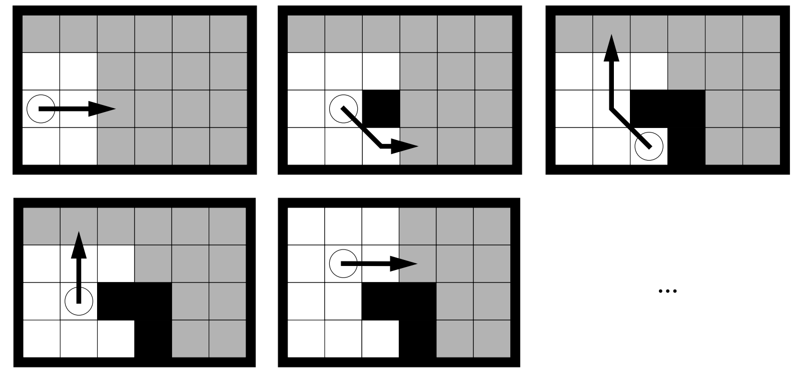
Greedy On-line Planning
is Reactive to the Current Situation
(plus other advantages)

Greedy Mapping

we assume here that the robot can move in eight directions

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

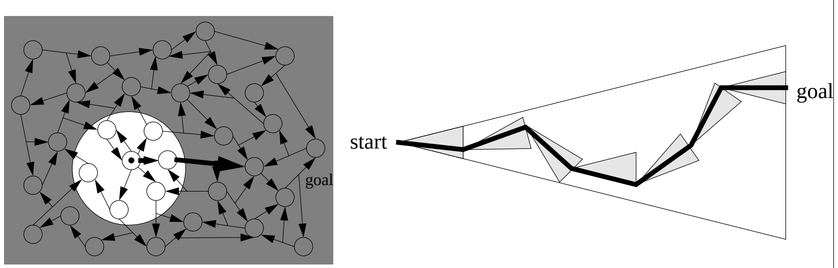
[Koenig, Tovey, Halliburton, 2001] [Thrun et al. 1998] [Romero, Morales, Sucar, 2001]



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.

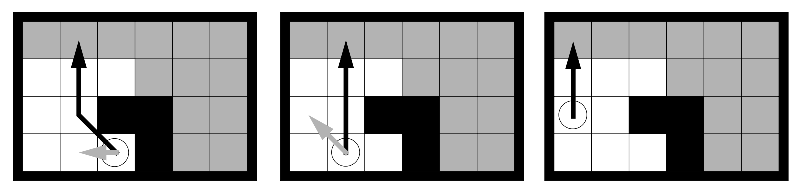


Note: Assume mapping is possible. The state space is safely explorable.
Greedy Mapping always achieves a gain in information.
Thus, Greedy Mapping maps the terrain.

Greedy Mapping - Advantages

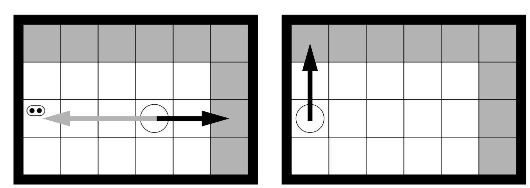
we assume here that the robot can move in eight directions

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]

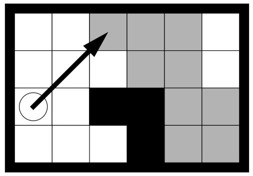
does not need to be in control of the robot at all times (“reactive planning”)



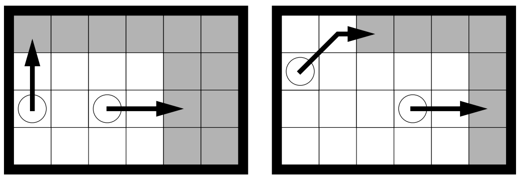
Greedy Mapping - Advantages

we assume here that the robot can move in eight directions

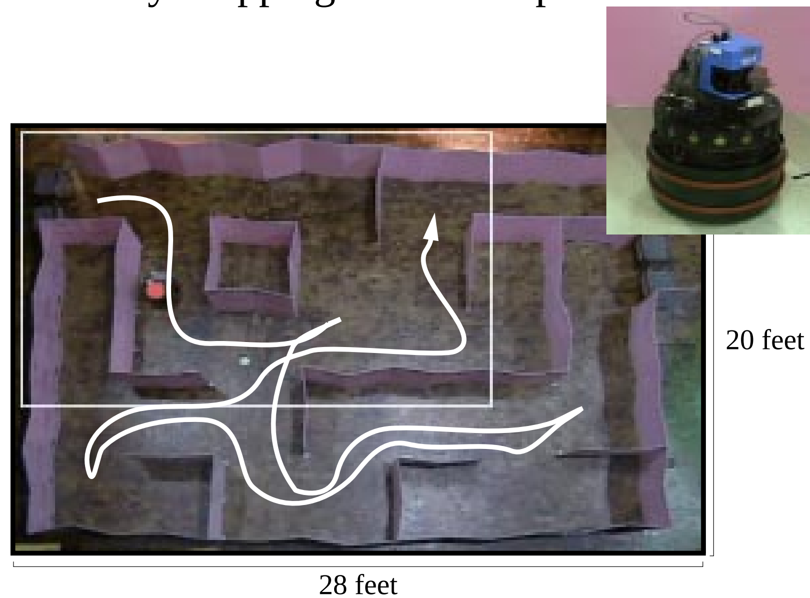
utilizes prior map knowledge, if available



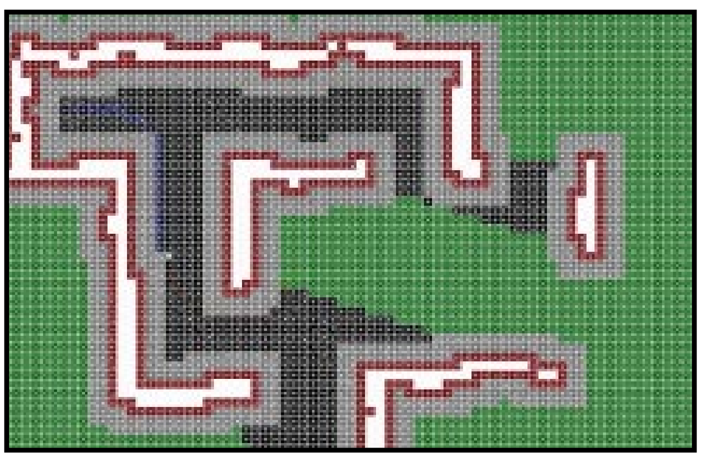
can be used by multiple robots that share their maps



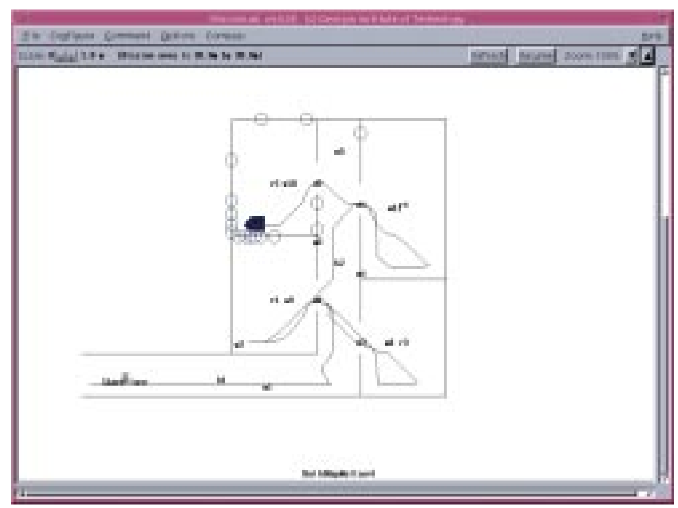
Greedy Mapping - Robot Implementation



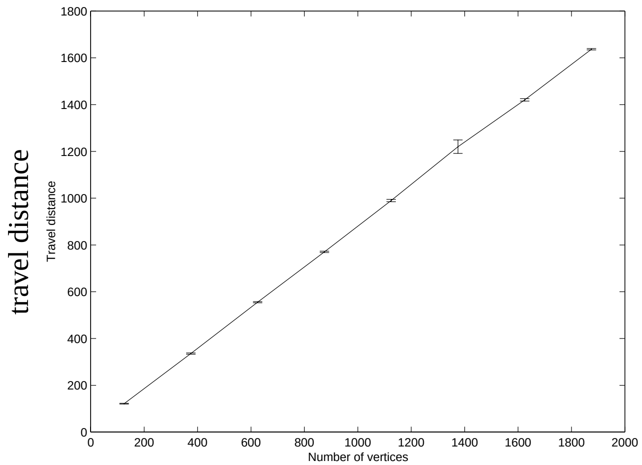
Greedy Mapping - Robot Implementation



Greedy Mapping - Travel Distance



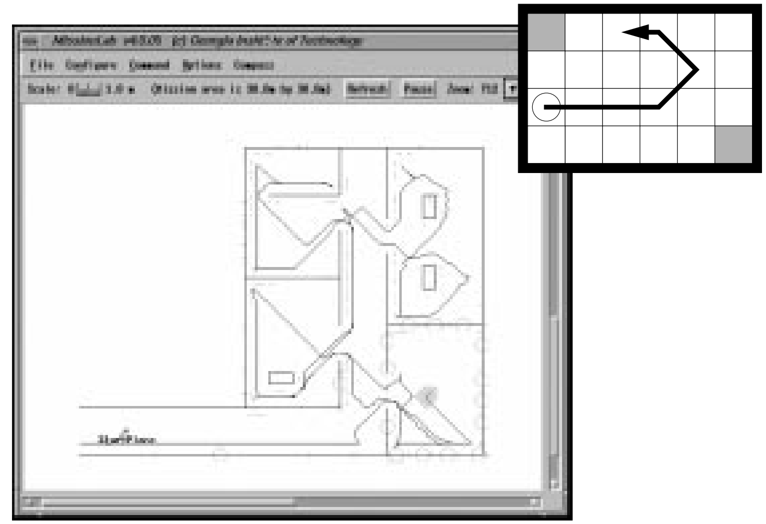
Greedy Mapping - Travel Distance



number of vertices

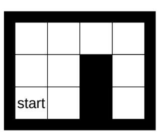
Greedy Mapping - Travel Distance

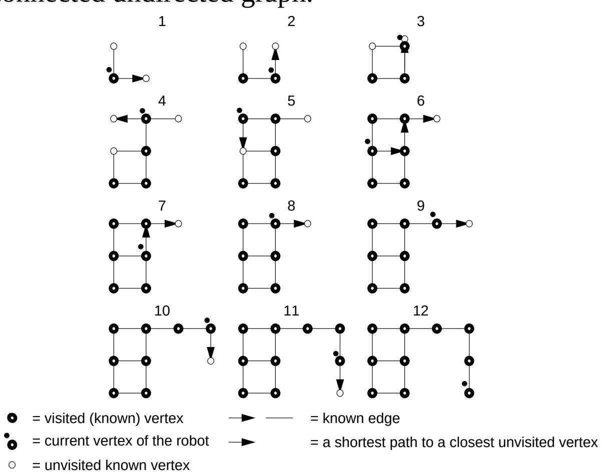
we assume here that the robot can move in eight directions



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.





Greedy Mapping - Travel Distance

Here: Greedy Mapping always moves the robot on a shortest path to the closest **unvisited** cell.

Trivial Theorem

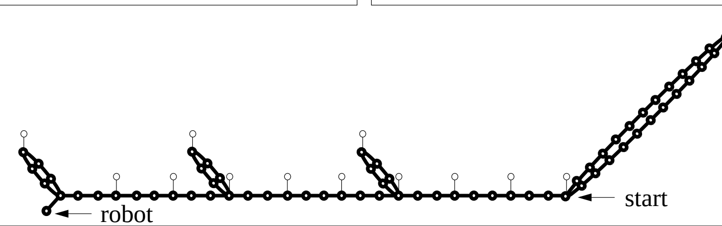
Theorem:

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.

More Interesting Theorem

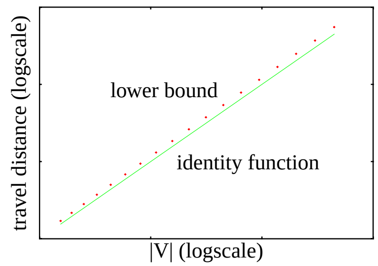
Theorem: [Koenig, Tovey, Smirnov, 2001]

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



Greedy Mapping - Travel Distance

n	travel distance	|V|	travel distance / |V|
3	207	80	2.59
4	2279	778	2.93
5	31253	9612	3.25
6	515085	144014	3.58
7	9928271	2542528	3.90
8	219130987	51744018	4.23
9	5448100629	1193201300	4.57
10	150617283953	30753086422	4.90



- order of |V|log |V| — upper bound for Greedy Mapping
- order of $\frac{\log |V|}{\log \log |V|} |V|$ — lower bound for Greedy Mapping

can we use structure to decrease the travel distance?

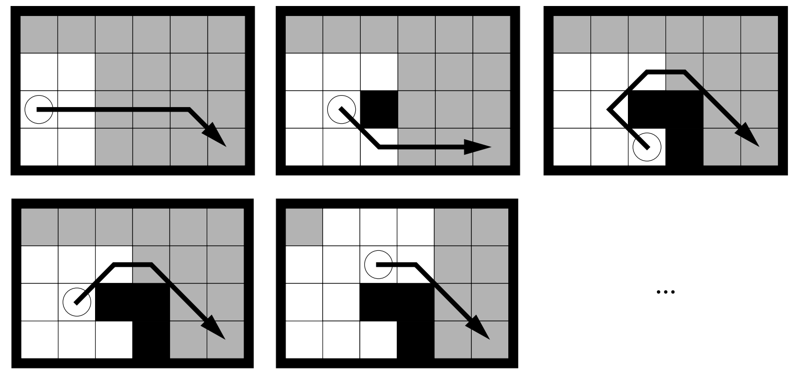
- order of |V| — tight bound for chronological backtracking

Planning with the Freespace Assumption

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.

[Brumitt and Stentz, 1998] [Hebert, McLachlan, Chang, 1999] [Matthies et al., 2000] [Stentz and Hebert, 1995] [Thayer et al., 2000]



Planning with the Freespace Assumption

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

[Brumitt and Stentz, 1998] [Hebert, McLachlan, Chang, 1999] [Matthies et al., 2000] [Thayer et al., 2000]

- Demo Vehicles of the Darpa UGV II Program
- Mars Rover Prototype
- Prototypes of Urban Reconnaissance Robots

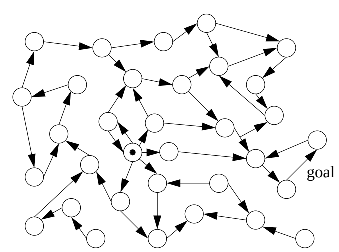
HMMWV that navigated 1,410 meters of natural outdoor terrain in 1995

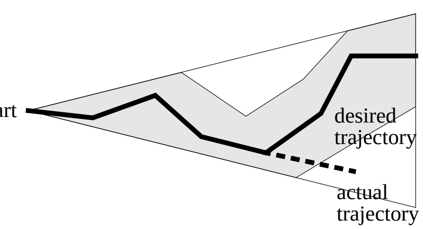
[Stentz and Hebert, 1995]

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.

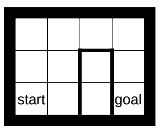




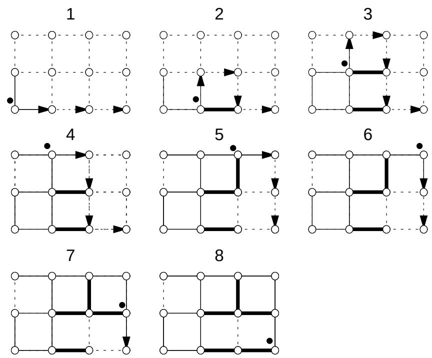
Note: Assume moving to the goal is possible. The state space is safely explorable. Planning with the Freespace Assumption always achieves a gain in information. Thus, Planning with the Freespace Assumption moves to the goal.

Freespace Assumption - Travel Distance

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



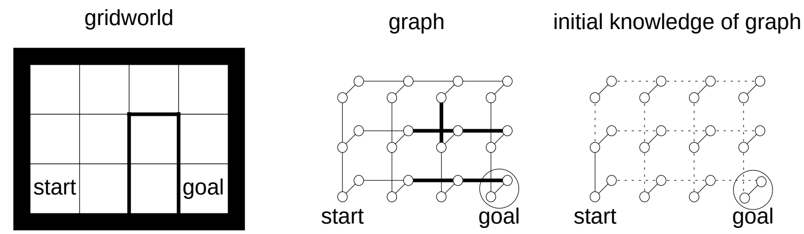
robot can
- move north
- move east
- move south
- move west



= edge known to be unblocked
= edge assumed to be unblocked
= edge known to be blocked
= a shortest potentially traversable path to the goal

Freespace Assumption - Travel Distance

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



robot can
- move forward
- move backward
- turn 90 degree left (or right)

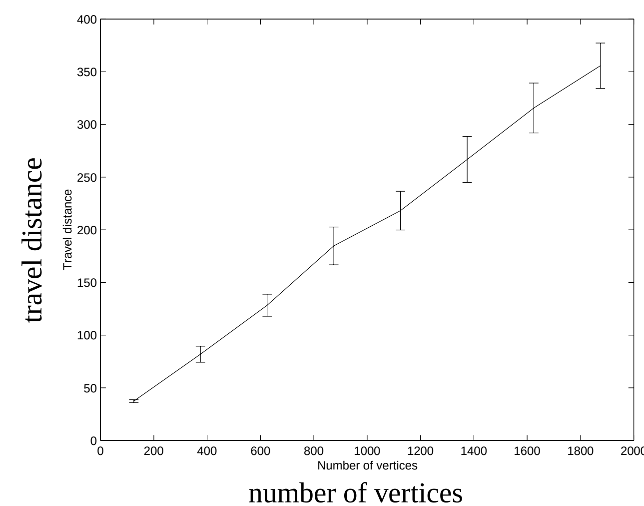
= unblocked edge / edge known to be unblocked
= edge assumed to be unblocked
= blocked edge / edge known to be blocked

Freespace Assumption - Travel Distance

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



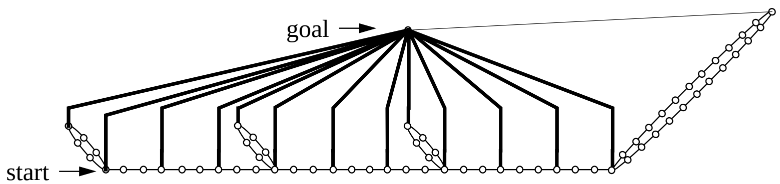
Freespace Assumption - Travel Distance

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 $\Omega(\frac{\log s}{\log\log s} s)$ 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



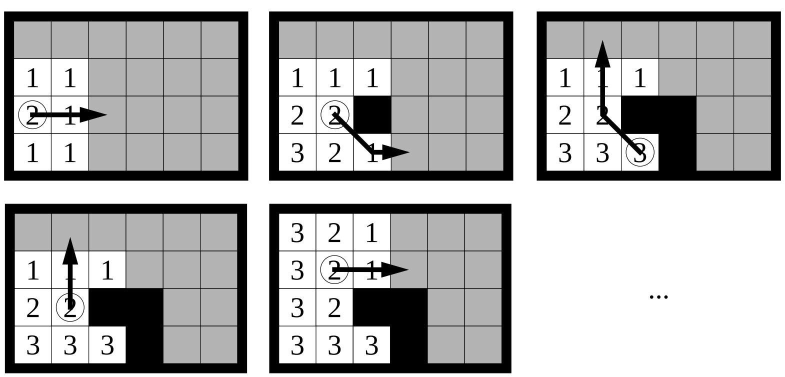
Part 3

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.

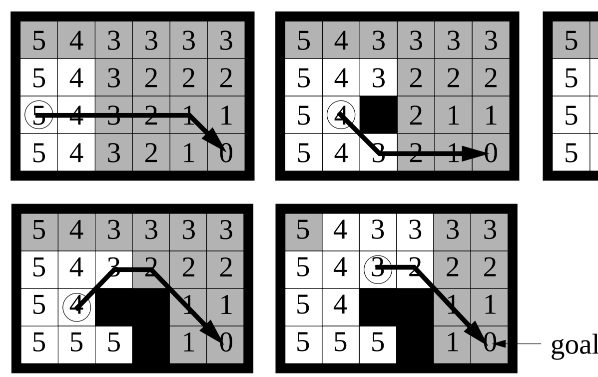


...

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.

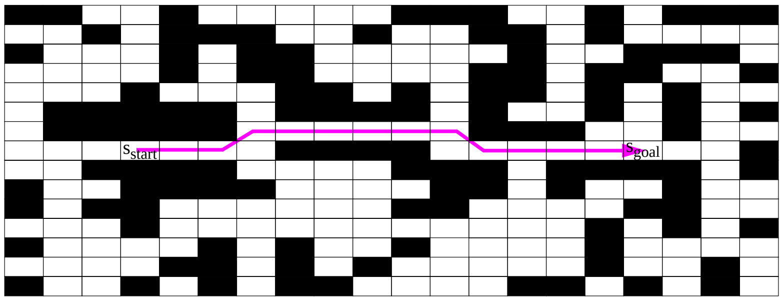


...

Path Planning - Example

we assume here that the robot can move in eight directions

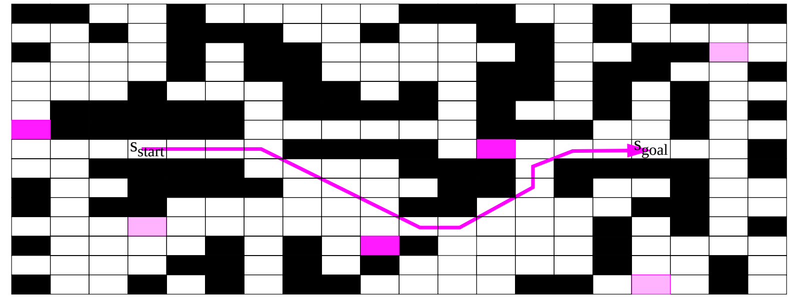
original eight-connected gridworld



Path Planning - Example

we assume here that the robot can move in eight directions

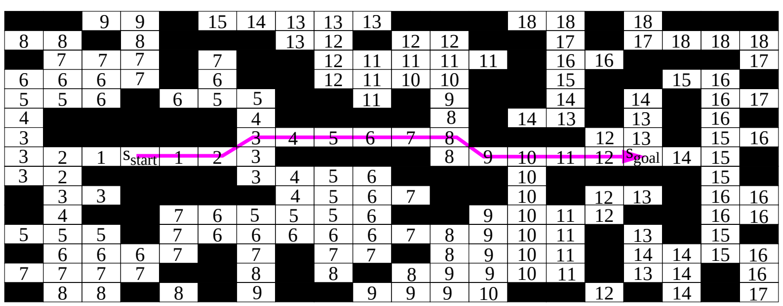
changed eight-connected gridworld



Path Planning - Example

we assume here that the robot can move in eight directions

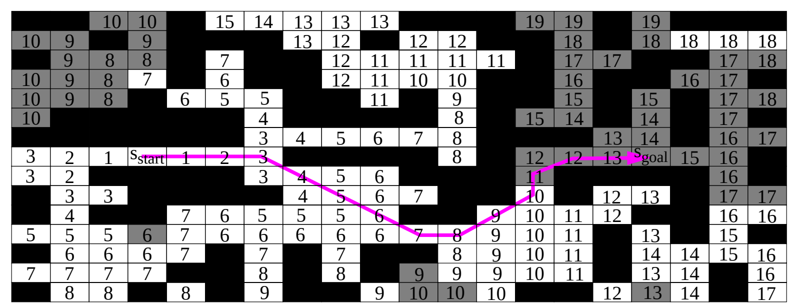
original eight-connected gridworld



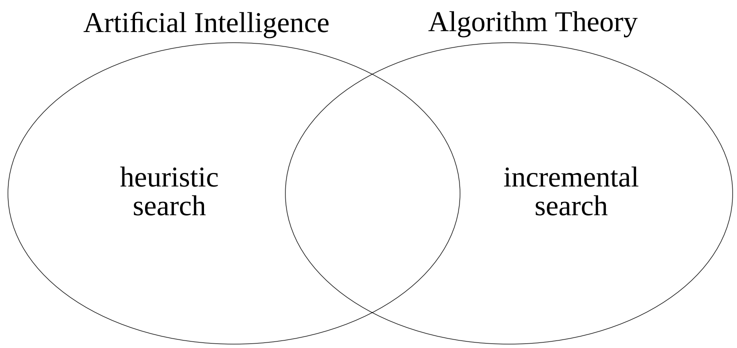
Path Planning - Example

we assume here that the robot can move in eight directions

changed eight-connected gridworld



Path Planning - Example



how to search efficiently using heuristic to guide the search

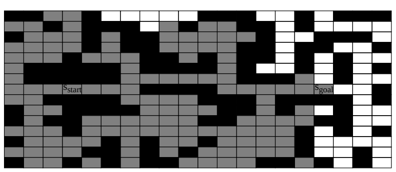
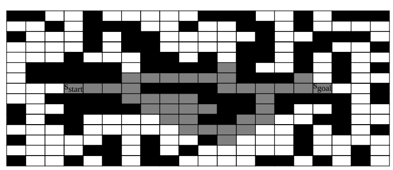
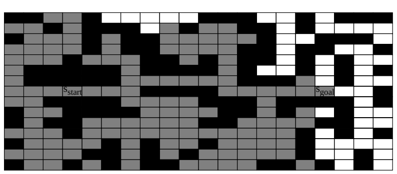
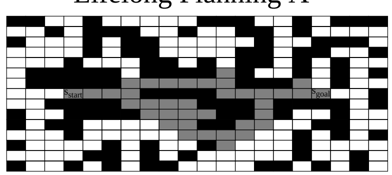
how to search efficiently by reusing information from previous searches

Path Planning - Lifelong Planning A*

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

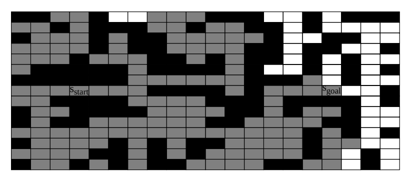
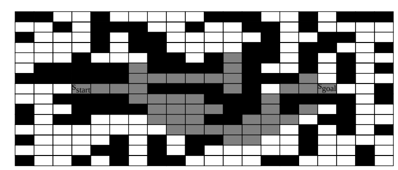
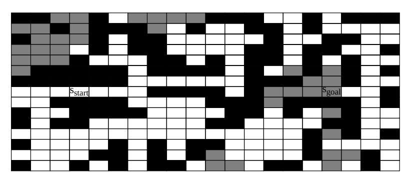
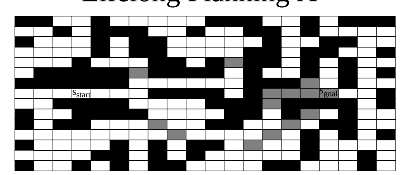
Path Planning - Experimental Evaluation

original eight-connected gridworld

	uninformed search	heuristic search
complete search		
incremental search		Lifelong Planning A* 

Path Planning - Experimental Evaluation

changed eight-connected gridworld

	uninformed search	heuristic search
complete search		
incremental search		Lifelong Planning A* 

Path Planning - Experimental Evaluation

changed eight-connected gridworld - first implementation

	uninformed search	heuristic search
complete search	ve = 1331.7 +/- 4.4 va = 26207.2 +/- 84.0 hp = 5985.3 +/- 19.7	(with the same tie-breaking as LPA*) ve= 284.0 +/- 5.9 va= 6177.3 +/- 129.3 hp= 1697.3 +/- 39.9
incremental search	ve = 173.0 +/- 4.9 va = 5697.4 +/- 167.0 hp = 956.2 +/- 26.6	Lifelong Planning A* ve= 25.6 +/- 2.0 va= 1235.9 +/- 75.0 hp= 240.1 +/- 16.9

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

Path Planning - Experimental Evaluation

changed eight-connected gridworld - second implementation

	uninformed search	heuristic search
complete search	ve = 801.76 hp = 2359.60	(with the same tie-breaking as LPA*) ve= 172.20 hp= 724.60
incremental search	ve = 115.95 hp = 561.48	Lifelong Planning A* ve= 18.80 hp= 182.15

ve = vertex expansions, hp = heap percolates

Path Planning - Experimental Evaluation

heuristic search
(with **better** tie-breaking than LPA*)

ve= 68.17
hp= 547.72
t1= 13.62
t2= 18.61

Lifelong Planning A*

ve= 18.80
hp= 182.15
t1= 6.66
t2= 13.22

A* expands nodes faster than LPA* – tie-breaking matters

time speedup = x1.5 in the long run

ve = vertex expansions, hp = heap percolates, t1 = time in main search routine,
t2= total runtime (including maze generation etc.)

after the third replanning episode,
the total planning time of LPA* over all episodes is less than that of A*

Path Planning - Lifelong Planning A*

[Koenig, Likhachev, 2001]

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

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:
$h(s_{goal},s_{goal}) = 0$
and $h(s,s_{goal}) \leq c(s,s') + h(s',s_{goal})$
for all vertices $s \in S$ and $s' \in Succ(s)$.

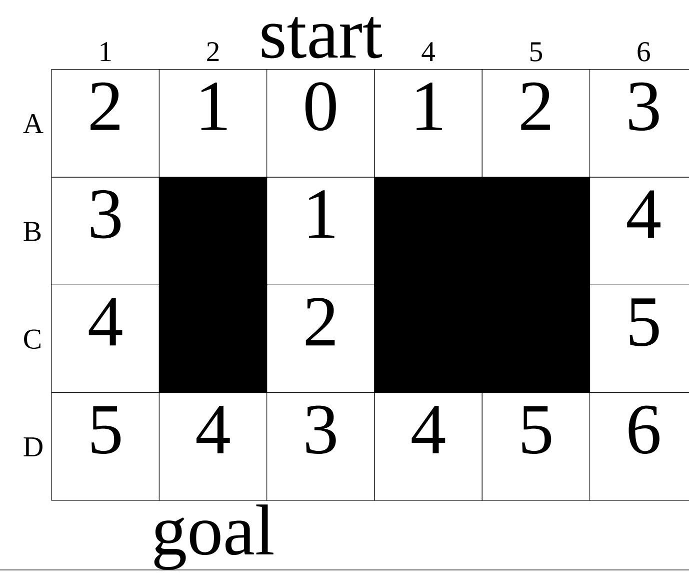
This version of LPA* can be optimized further without changing its overall operation.

We also have versions of LPA* that
- break ties differently
- work with inconsistent heuristics
- terminate earlier
- contain several runtime optimizations.

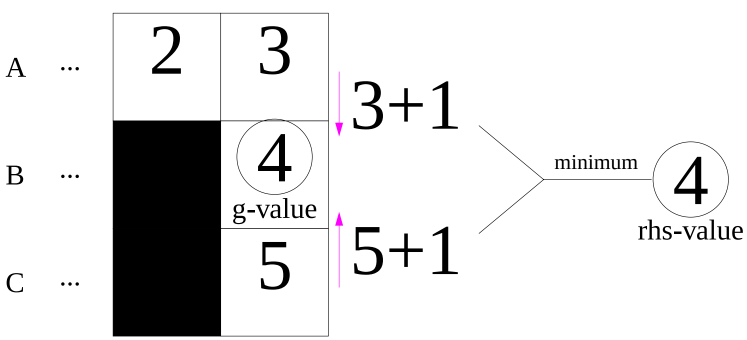
Path Planning - Lifelong Planning A*

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
  - ...

Path Planning - Lifelong Planning A*



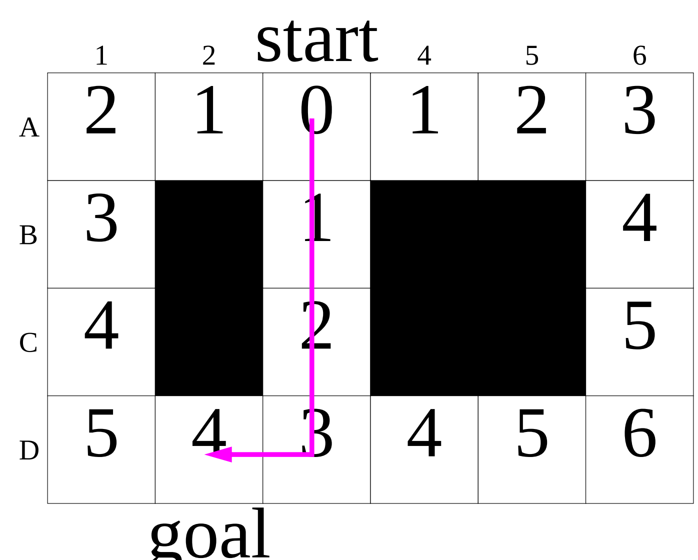
Path Planning - Lifelong Planning A*



g-value = rhs-value: cell is locally consistent
g-value ≠ rhs-value: cell is locally inconsistent
g-value > rhs-value: cell is locally overconsistent
g-value < rhs-value: cell is locally underconsistent

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

Path Planning - Lifelong Planning A*



Path Planning - Lifelong Planning A*

start

	1	2	3	4	5	6
A	2	1	0	1	2	3
B	3					4
C	4		2 min(2,4)+2 min(2,4)			5
D	5	4	3	4	5	6

goal

priority queue
C3: [4;2]

Path Planning - Lifelong Planning A*

start

	1	2	3	4	5	6
A	2	1	0	1	2	3
B	3					4
C	4		∞ min(∞,4)+2 min(∞,4)			5
D	5	4	3 min(3,5)+1 min(3,5)	4	5	6

goal

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

Path Planning - Lifelong Planning A*

start

	1	2	3	4	5	6
A	2	1	0	1	2	3
B	3					4
C	4		∞			5
D	5	4 min(4,6)+0 min(4,6)	∞ min(∞,5)+1 min(∞,5)	4 min(4,6)+2 min(4,6)	5	6

goal

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

Path Planning - Lifelong Planning A*

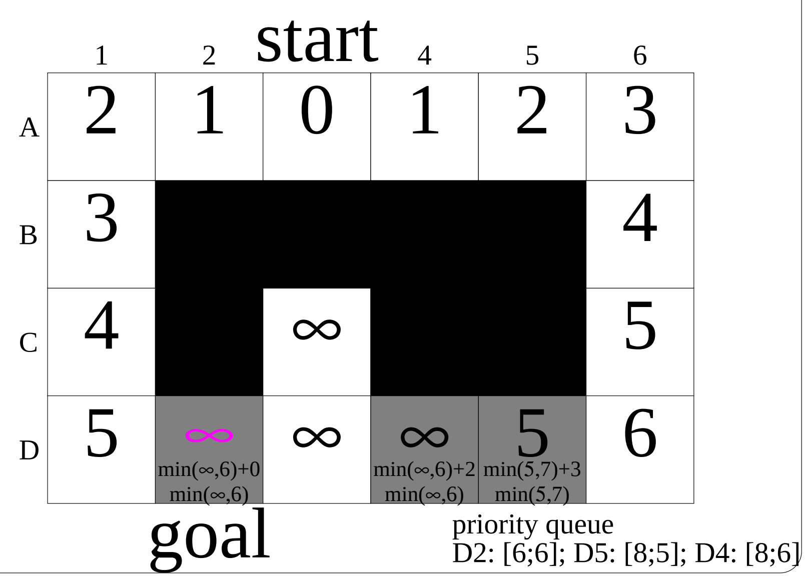
start

	1	2	3	4	5	6
A	2	1	0	1	2	3
B	3					4
C	4		∞			5
D	5	∞ min(∞,6)+0 min(∞,6)	∞ min(∞,5)+1 min(∞,5)	4 min(4,6)+2 min(4,6)	5	6

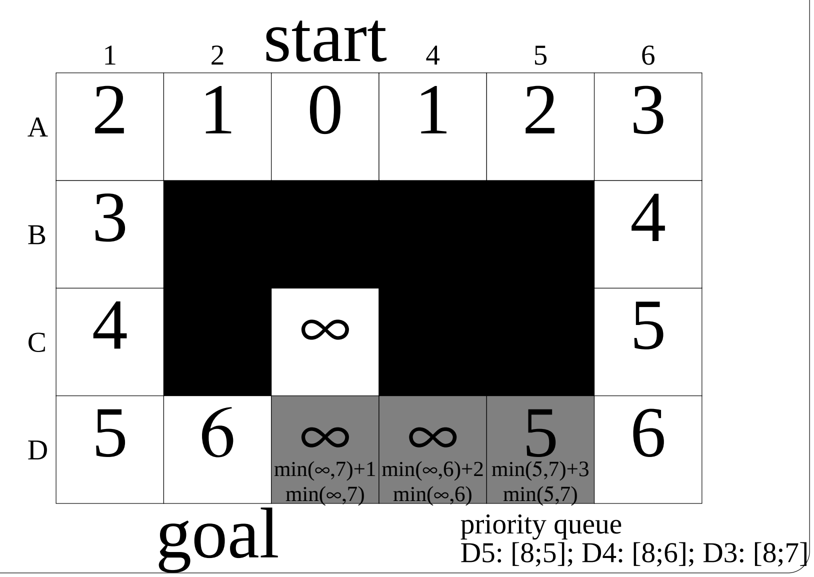
goal

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

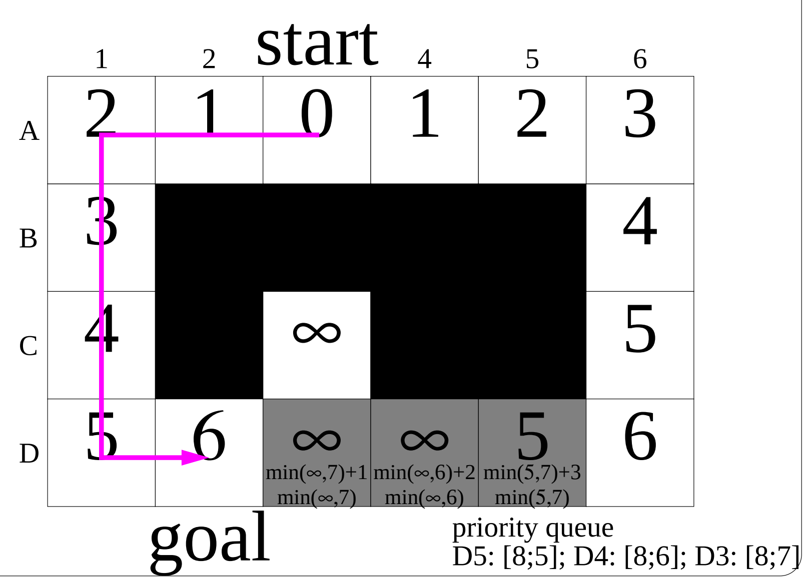
Path Planning - Lifelong Planning A*



Path Planning - Lifelong Planning A*



Path Planning - Lifelong Planning A*



Path Planning - Lifelong Planning A*

Theorem: [Likhachev and Koenig, 2001]

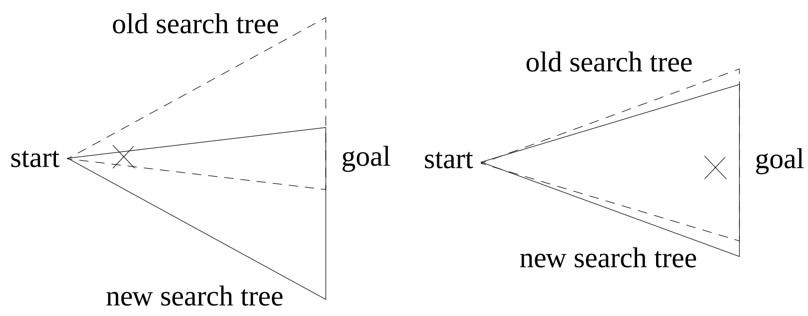
> ComputeShortestPath() expands every vertex at most twice and thus terminates.

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]



Path Planning - Lifelong Planning A*

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 $[f(s); g^*(s)] \leq [f(s_{start}); g^*(s_{start})]$ or $[g_{old}(s)+h(s); g_{old}(s)] \leq [f(s_{start}); g^*(s_{start})]$, 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*

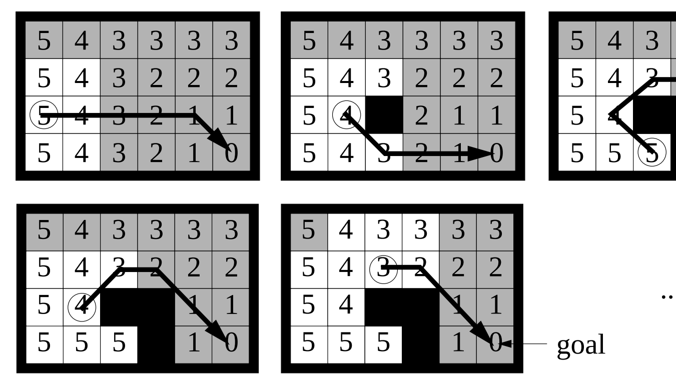
"Theorem:" [Likhachev and Koenig, 2001]

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*.)

Freespace Assumption - Implementation

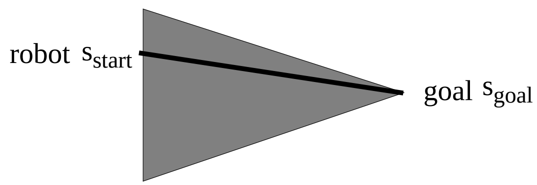
we assume here that the robot can move in eight directions

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

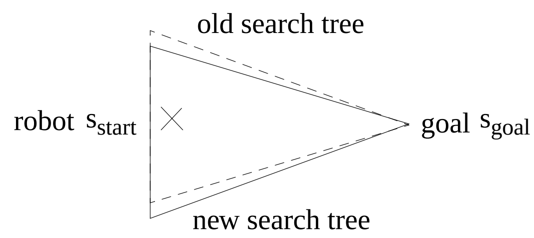
here: search from the goal location towards the robot location
- allows one to reuse parts of the search tree after the robot has moved
- allows one to use heuristics to focus the search
  (this additional argument holds for Greedy Mapping later)



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

Transforming Planning with the Freespace Assumption to Path Planning

here: search from the goal location towards the robot location
- makes incremental search efficient



Freespace Assumption - D* Lite (Basic Version)

[Koenig, Likhachev, 2002]

```
procedure CalculateKey(s)
return [min(g(s), rhs(s)) + h(s_start, s); min(g(s), rhs(s))];
procedure Initialize()
U = Ø;
for all s ∈ S rhs(s) = g(s) = ∞
rhs(s_goal) = 0;
U.Insert(s_goal, CalculateKey(s_goal));
procedure UpdateVertex(u)
if (u ≠ s_goal) rhs(u) = min_{s' ∈ Succ(u)} (c(u,s')+g(s'));
if (u ∈ U) U.Remove(u);
if (g(u) ≠ rhs(u)) U.Insert(u, CalculateKey(u));
procedure ComputeShortestPath()
while (U.TopKey < CalculateKey(s_start) OR rhs(s_start) ≠ g(s_start))
    u = U.Pop();
    if (g(u) > rhs(u))
        g(u) = rhs(u);
        for all s ∈ Pred(u) UpdateVertex(s);
    else
        g(u) = ∞;
        for all s ∈ Pred(u) ∪ {u} UpdateVertex(s);
procedure Main()
Initialize();
ComputeShortestPath();
while (s_start ≠ s_goal)
    /* if (g(s_start) = ∞) then there is no known path */
    s_start = arg min_{s' ∈ Succ(sstart)} (c(s_start,s')+g(s'))
    Move to s_start;
    Scan graph for changed edge costs;
```

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 backward consistent no matter what the start vertex is:
$h(s_{start},s_{start}) = 0$
and $h(s_{start},s) \le h(s_{start}, s')+c(s',s)$
for all vertices $s \in S$ and $s' \in Pred(s)$.

```
if any edge costs changed
    for all directed edges (u,v) with changed edge costs
        Update the edge cost c(u,v);
        UpdateVertex(u);
    for all s ∈ U
        U.Update(s, CalculateKey(s));
    ComputeShortestPath();
```


Freespace Assumption - D* Lite (Basic Version) Idea

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

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

This value changes when the robot moves from $s_{oldstart}$ to $s_{newstart}$.
Thus, one needs to reorder the priority queue. [Stentz, 1994]

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

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

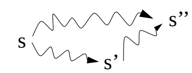
priority queue C: [7;7]; B: [8;6]; A: [9;5]

Freespace Assumption - D* Lite (Final Version)

[Koenig, Likhachev, 2002]

```
procedure CalculateKey(s)
return [min(g(s), rhs(s)) + h(s_start, s) + k_m; min(g(s), rhs(s))];
procedure Initialize()
U = Ø;
k_m = 0;
for all s ∈ S rhs(s) = g(s) = ∞
rhs(s_goal) = 0;
U.Insert(s_goal, CalculateKey(s_goal));
procedure UpdateVertex(u)
if (u ≠ s_goal) rhs(u) = min_s' in Succ(u) (c(u,s')+g(s'));
if (u ∈ U) U.Remove(u);
if (g(u) ≠ rhs(u)) U.Insert(u, CalculateKey(u));
procedure ComputeShortestPath()
while (U.TopKey < CalculateKey(s_start) OR rhs(s_start) ≠ g(s_start))
    k_old = U.TopKey();
    u = U.Pop();
    if (k_old < CalculateKey(u))
        U.Insert(u, CalculateKey(u));
    else if (g(u) > rhs(u))
        g(u) = rhs(u);
        for all s ∈ Pred(u) UpdateVertex(s);
    else
        g(u) = ∞;
        for all s ∈ Pred(u) ∪ {u} UpdateVertex(s);
procedure Main()
s_last = s_start;
Initialize();
ComputeShortestPath();
while (s_start ≠ s_goal)
    /* if (g(s_start) = ∞) then there is no known path */
    s_start = arg min s'∈Succ(sstart) (c(s_start,s')+g(s'))
    Move to s_start;
    Scan graph for changed edge costs;
    if any edge costs changed
        k_m = k_m + h(s_last,s_start);
        s_last = s_start;
        for all directed edges (u,v) with changed edge costs
            Update the edge cost c(u,v);
            UpdateVertex(u);
        ComputeShortestPath();
```

The heuristics need to be nonnegative and forward-backward consistent:
$h(s,s'') \le h(s,s')+h(s',s'')$
for all vertices $s,s',s'' \in S$.
The heuristics also need to be admissible no matter what the goal vertex is:
$h(s,s') \le$ shortest distance from s to s'
for all vertices $s,s' \in S$.



triangle inequality

Freespace Assumption - D* Lite (Final Version) Idea

[Stentz, 1995]]

Reordering the priority queue is time consuming.

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

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

$[\min(g(s),rhs(s))+h(s_{oldstart},s); \min(g(s),rhs(s))]$
$\le [\min(g(s),rhs(s))+h(s_{oldstart},s_{newstart})+h(s_{newstart},s); \min(g(s),rhs(s))]$
$[\min(g(s),rhs(s))+h(s_{oldstart},s)-h(s_{oldstart},s_{newstart}); \min(g(s),rhs(s))]$
$\le [\min(g(s),rhs(s))+h(s_{newstart},s); \min(g(s),rhs(s))]$

The term $h(s_{oldstart},s_{newstart})$ is the same across vertices in the priority queue.
Instead of deleting it from the 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.

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: [6;5]; B: [6;6]; C: [6;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]

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

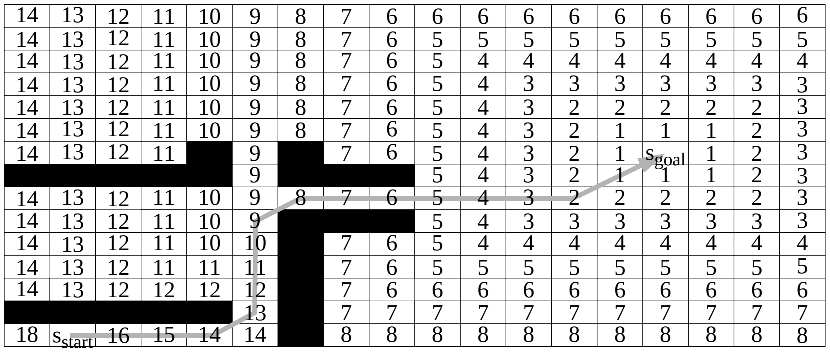
priority queue B: [8;6]; C: [8;7]; A: [9;5]
correct priority is B: [8;6]

expand B

Freespace Assumption - D* Lite

we assume here that the robot can move in eight directions

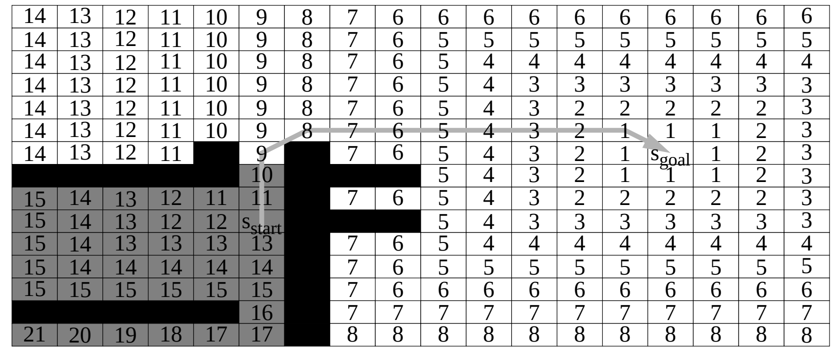
knowledge before the movement sequence of the robot



Freespace Assumption - D* Lite

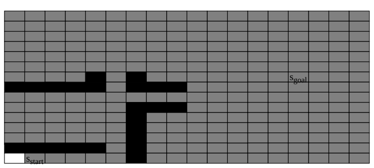
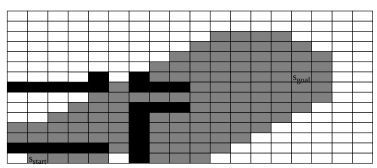
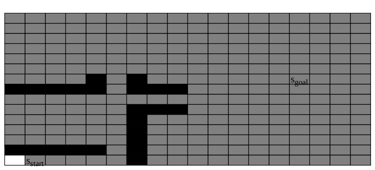
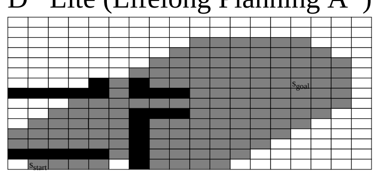
we assume here that the robot can move in eight directions

knowledge after the movement sequence of the robot



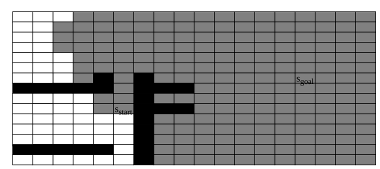
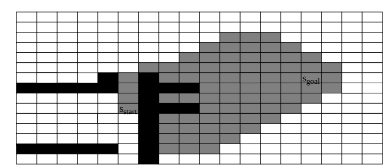
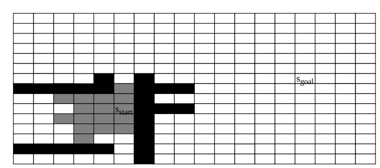
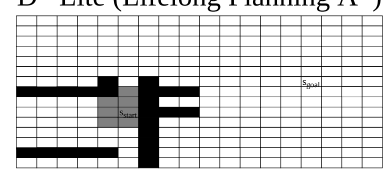
Freespace Assumption - D* Lite

before the movement sequence of the robot

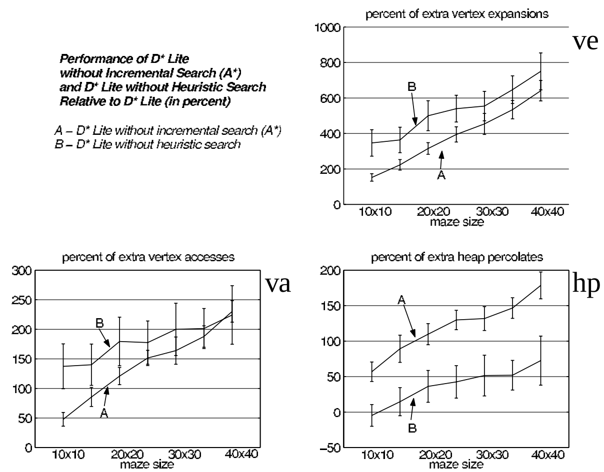
	uninformed search	heuristic search
complete search		
incremental search		D* Lite (Lifelong Planning A*) 

Freespace Assumption - D* Lite

after the movement sequence of the robot

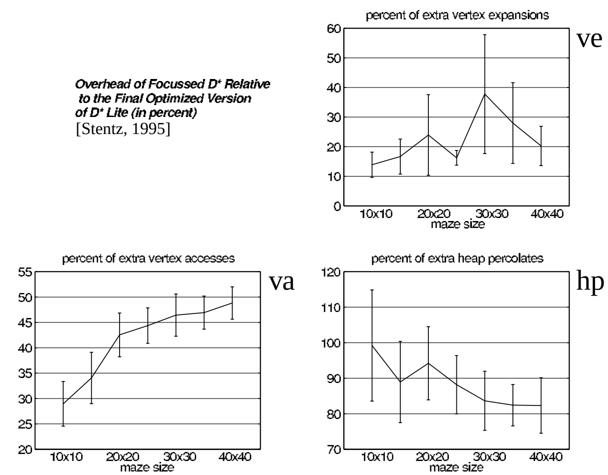
	uninformed search	heuristic search
complete search		
incremental search		D* Lite (Lifelong Planning A*) 

Freespace Assumption - D* Lite



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

Freespace Assumption - D* Lite

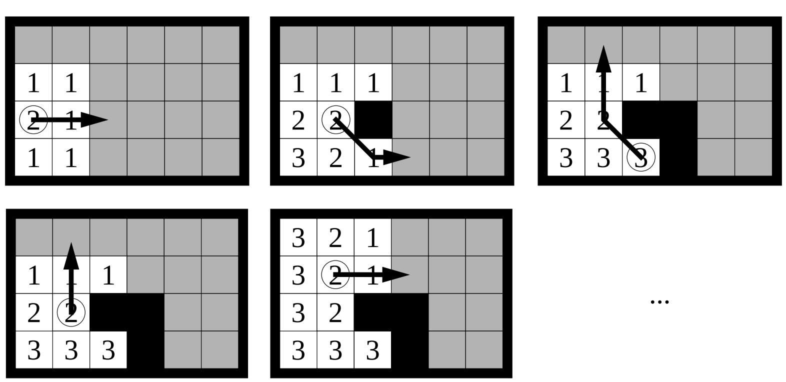


overhead of Focussed D* =
probably the first truly incremental heuristic search method
(note: Focussed D* is likely a bit faster than D* Lite per vertex expansion)

Greedy Mapping - Implementation

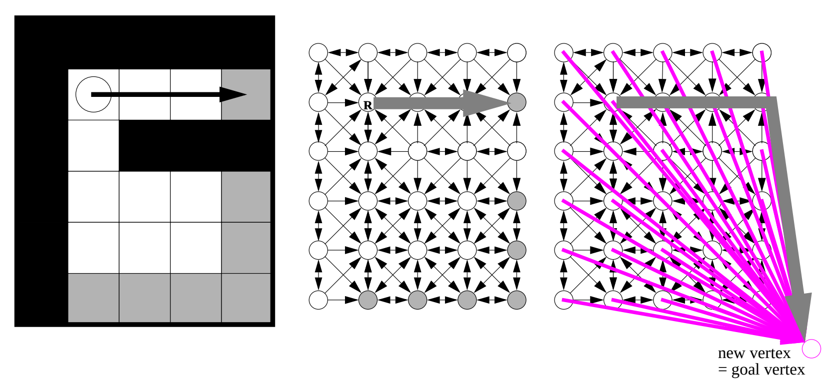
we assume here that the robot can move in eight directions

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



...

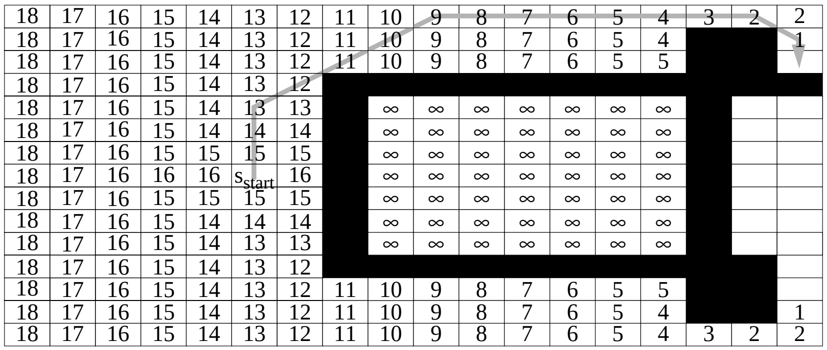
Transforming Greedy Mapping to Planning with the Freespace Assumption



Greedy Mapping - D* Lite

we assume here that the robot can move in eight directions

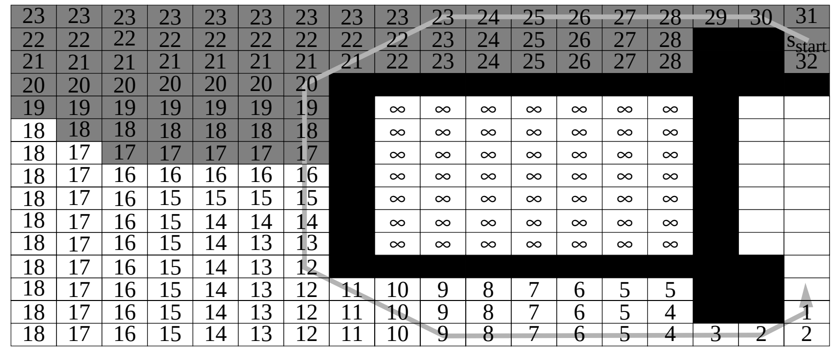
knowledge before the movement sequence of the robot



Greedy Mapping - D* Lite

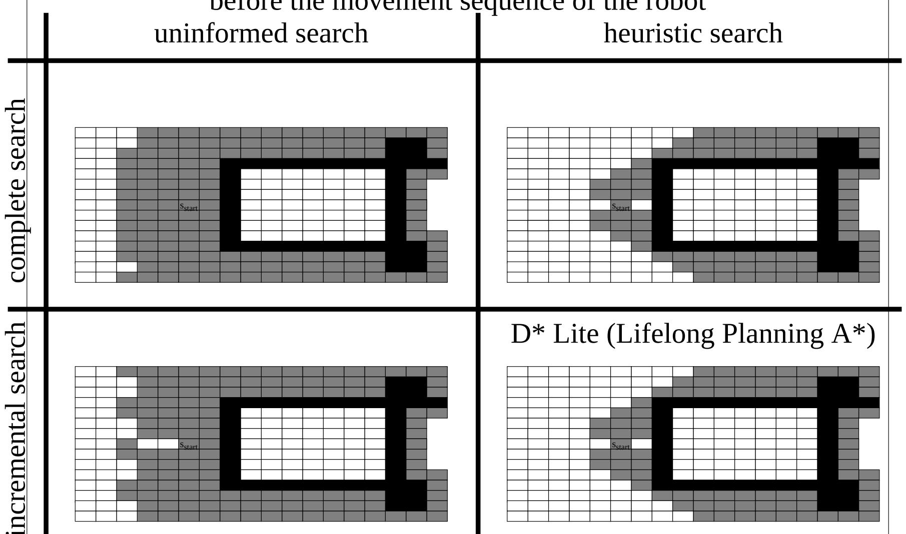
we assume here that the robot can move in eight directions

knowledge after the movement sequence of the robot



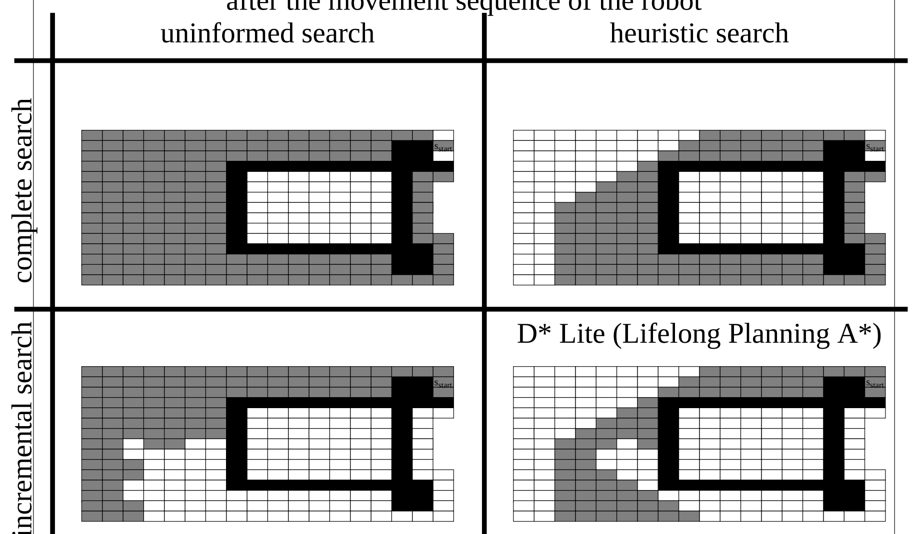
Greedy Mapping - D* Lite

before the movement sequence of the robot

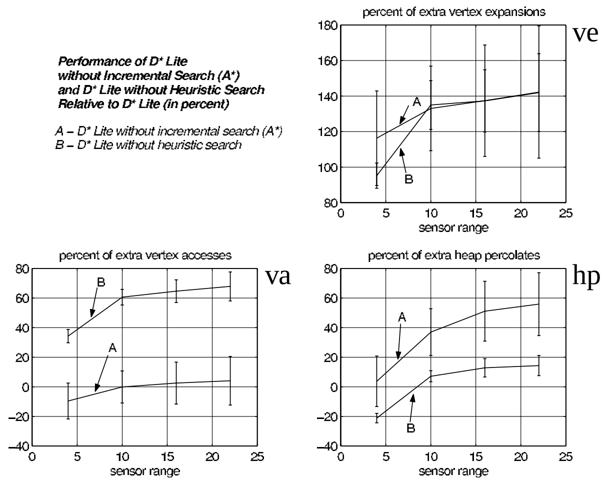


Greedy Mapping - D* Lite

after the movement sequence of the robot

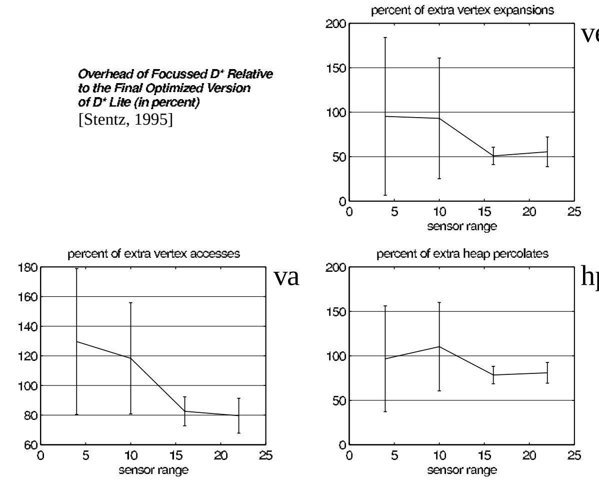


Greedy Mapping - D* Lite



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

Greedy Mapping - 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)

Other Examples of Lifelong Planning

emergency management

replanning (and plan reuse) is important!

- world changes over time
- model of the world changes over time
- what-if analyses

planning task 1	slightly different planning task 2	slightly different planning task 3	...

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)

Game Playing

Total Annihilation

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

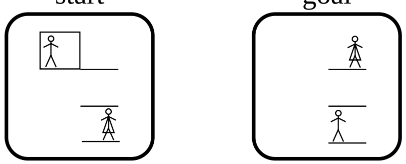
SHERPA

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

Symbolic Planning (with HSP) - Continual Planning

STRIPS-type planning in the elevator domain

start goal



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

SHERPA
Speedy HEuristic search-based RePlAnner

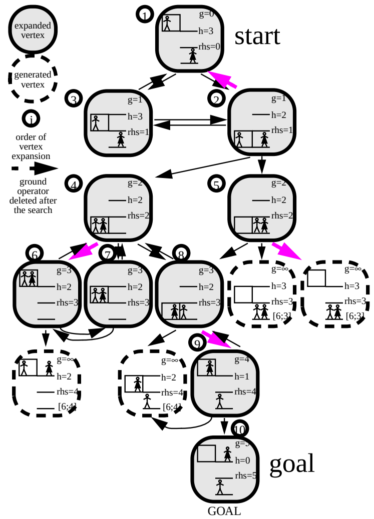
[S. Koenig, D. Furcy, C. Bauer, 2002]

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

Symbolic Planning (with HSP) - Continual Planning

first search in the
elevator domain
using SHERPA

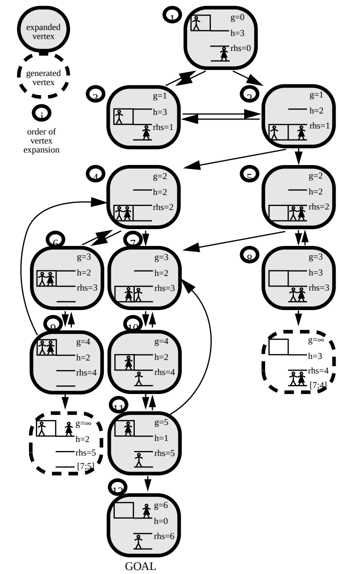
similar to HSP 2.0
with the h_{max} heuristic
[Bonet, Geffner, 2001]



Symbolic Planning (with HSP) - Continual Planning

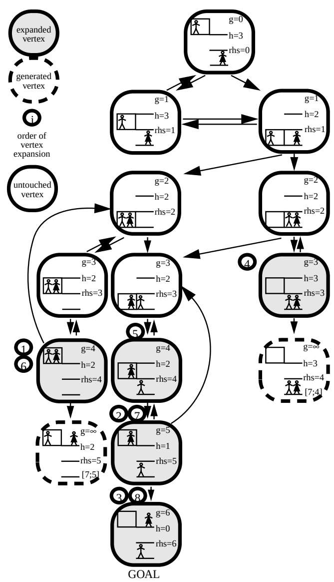
second search in the
elevator domain
using SHERPA from scratch

similar to HSP 2.0
with the h_{max} heuristic
[Bonet, Geffner, 2001]

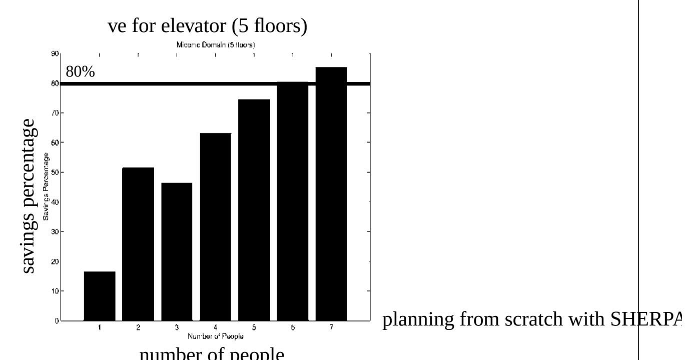


Symbolic Planning (with HSP) - Continual Planning

second search in the
elevator domain
using SHERPA

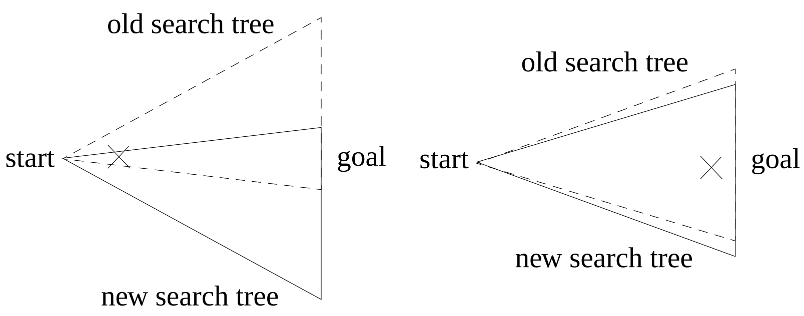


Symbolic Planning (with HSP) - Continual Planning



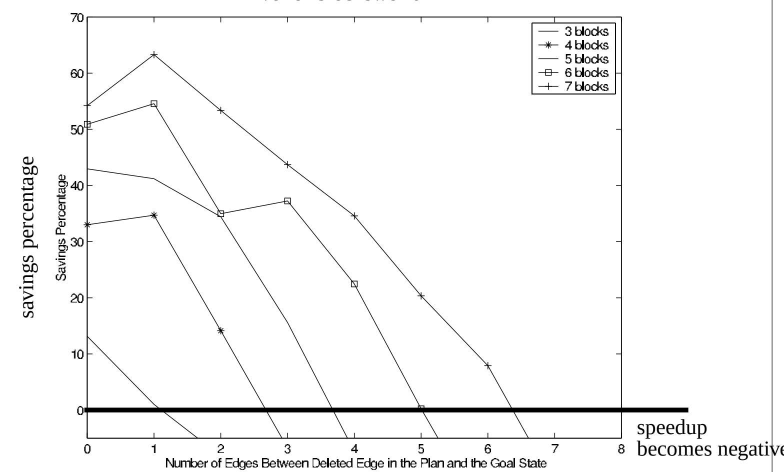
SHERPA achieves speedups up to 80 percent

Symbolic Planning (with HSP) - Continual Planning



Symbolic Planning (with HSP) - Continual Planning

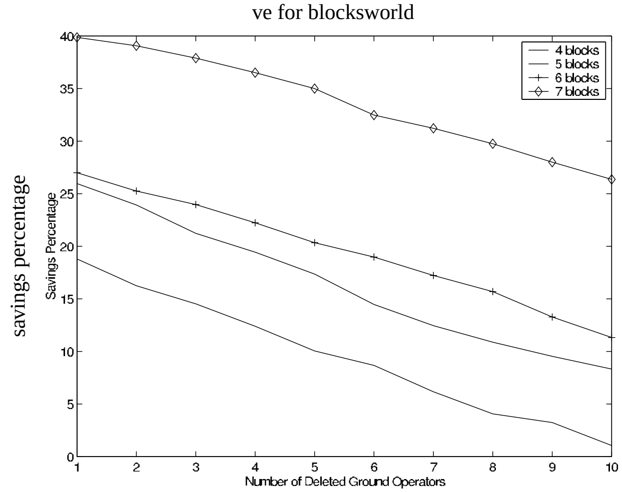
ve for blocksworld



number of edges between the deleted edge in the plan and the goal state

Symbolic Planning (with HSP) - Continual Planning

ve for blocksworld

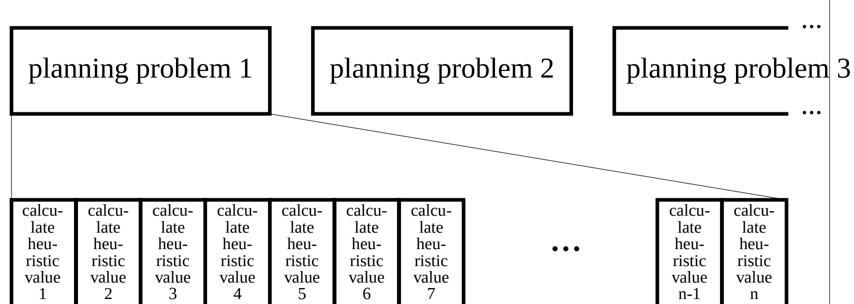


number of deleted ground operators

Symbolic Planning (with HSP) - One-Time 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

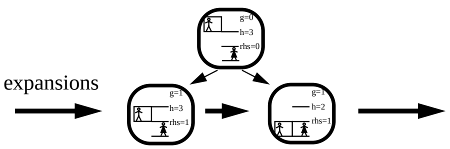
here: for HSP 2.0 with the h_{add} heuristic [Bonet, Geffner, 2001]

$h_{add}(state) = \sum_{\text{proposition in goal state}} g_{state}(proposition)$

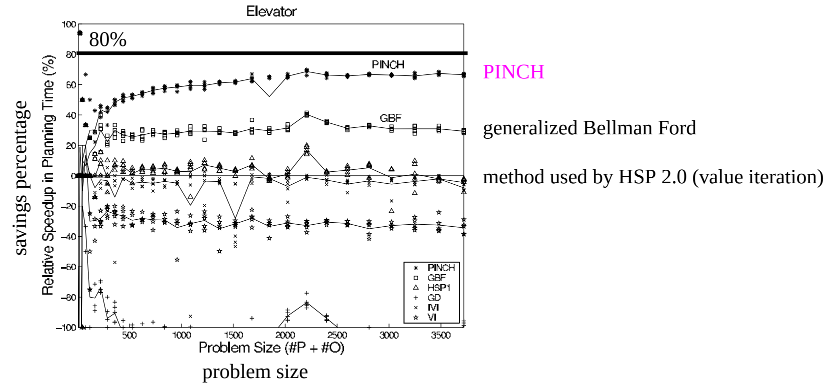
$g_{state}(proposition) = \begin{cases} 0 & \text{if proposition in state} \\ \min_{\text{operator with proposition in add list}} (1 + g_{state}(operator)) & \text{otherwise} \end{cases}$

$g_{state}(operator) = \sum_{\text{proposition on precondition list of operator}} g_{state}(proposition)$

order of state expansions



Symbolic Planning (with HSP) - One-Time Planning



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

Reinforcement Learning and On-Line DP

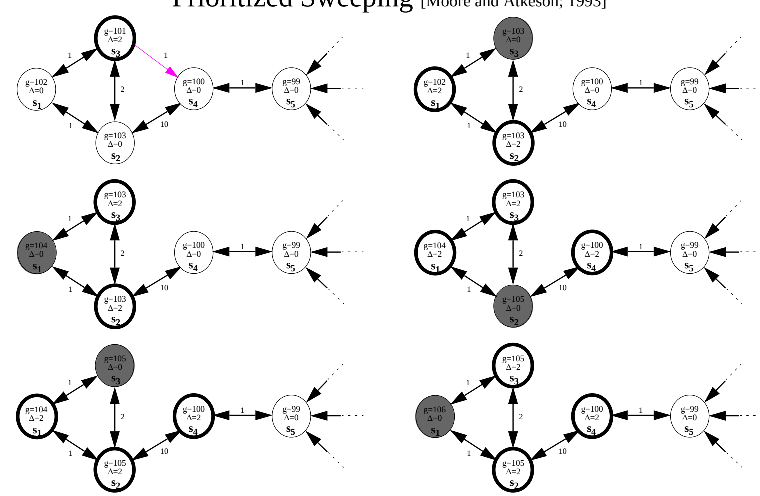
while there exists at least one state with g(s) = rhs(s)
pick a state s with g(s) = rhs(s) and then set g(s) := rhs(s)

Prioritized Sweeping [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

Reinforcement Learning and On-Line DP

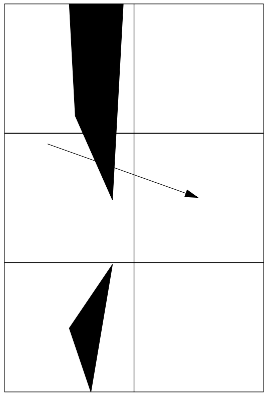
Prioritized Sweeping [Moore and Atkeson; 1993]

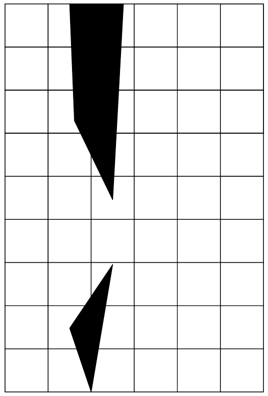


and so on, for a total of 22 g-value updates. Minimax LPA* needs only 6.
Note: Minimax LPA* expands every state at most twice.

Control (with the Parti-Game algorithm)

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



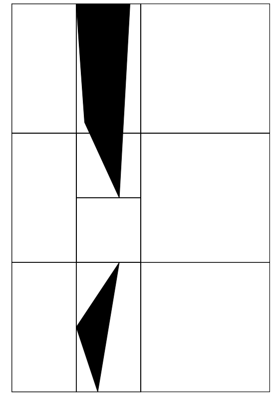


coarse-grained discretization
might not be able to find a plan

fine-grained discretization
is very inefficient

Control (with the Parti-Game algorithm)

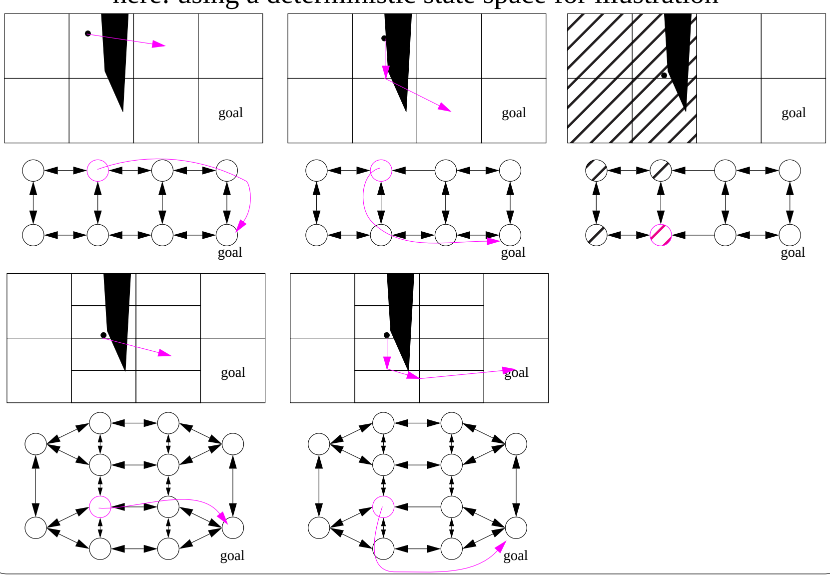
Parti-Game algorithm [Moore and Atkeson; 1995]



nonuniform discretization
avoids these problems

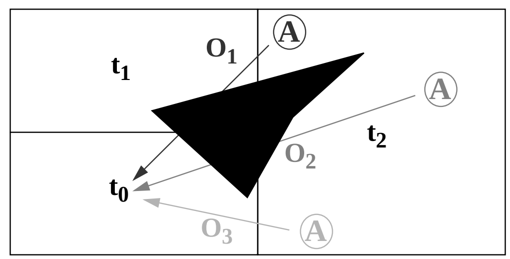
Control (with the Parti-Game algorithm)

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

Implementation	Planning Time	
Uninformed Search from Scratch	362 minutes	55 seconds
Informed Search from Scratch	135 minutes	15 seconds
Uninformed Incremental Search	14 minutes	53 seconds
Informed Incremental Search (Minimax LPA*)	13 minutes	53 seconds

References (in order of their appearance)

S. Koenig, Agent-Centered Search, Artificial Intelligence Magazine, 22(4), 2001, 109-131.
I. Nourbakhsh, Interleaving Planning and Execution for Autonomous Robots, Kluwer, 1997.
H. Choset, J. Burdick, Sensor-based planning and nonsmooth analysis. In Proceedings of the International Conference on Robotics and Automation, 1994, 3034-3041.
C. Tovey and S. Koenig, Gridworlds as Testbeds for Planning with Incomplete Information, Proceedings of the National Conference on Artificial Intelligence, 819-824, 2000.
G. Dudek, K. Romanik, S. Whitesides, Localizing a robot with minimum travel, In Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, 437-446, 1995.
C. Lund, M. Yannakakis, On the hardness of approximating minimization problems, Journal of the ACM, 41:960-981, 1994.
M. Genesereth and I. Nourbakhsh, Time-saving tips for problem solving with incomplete information, In Proceedings of the National Conference on Artificial Intlligence, 1993, 724-730.
S. Koenig and R. Simmons, Solving robot navigation problems with initial pose uncertainty using real-time heuristic search, In Proceedings of the International Conference on Artificial Intelligence Planning Systems, 1998, 145-153.
R. Simmons, S. Koenig, Probabilistic Robot Navigation in Partially Observable Environments, Proceedings of the International Joint Conference on Artificial Intelligence, 1993, 99-105.
S. Koenig and R. Simmons, Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models, In: Artificial Intelligence Based Mobile Robots: Case Studies of Successful Robot Systems, D. Kortenkamp, R. Bonasso, R. Murphy (Eds.), MIT Press, 1998.
S. Thrun, Probabilistic Algorithms in Robotics, Artificial Intelligence Magazine, 21(4), 2000, 93-109.
W. Burgard, D. Fox, S. Thrun, Active Mobile Robot Localization, Proceedings of the International Joint Conference on Artificial Intelligence, 1997.
R. Schapire, The Design and Analysis of Efficient Learning Algorithms, MIT Press, 1992.
C. Papadimitriou and J. Tsitsiklis, The complexity of Markov decision processes, Mathematics of Operations Research 12(3), 1987, 441-450.

References (in order of their appearance)

S. Koenig, C. Tovey, W. Halliburton, Greedy Mapping of Terrain, Proceedings of the International Conference on Robotics and Automation, 2001, 3594-3599.
S. Thrun, A. Buecken, W. Burgard, D. Fox, T. Froehlinghaus, D. Hennig, T. Hofmann, M. Krell, T. Schmidt, Map learning and high-speed navigation in RHINO, In : Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems, D. Kortenkamp, R. Bonasso, R. Murphy (Eds.), MIT Press, 1998, 21-52.
L. Romero, E. Morales, E. Sucar, An exploration and navigation approach for indoor mobile robots considering sensor's perceptual limitations, Proceedings of the International Conference on Robotics and Automation, 2001, 3092-3097.
D. Mackenzie, R. Arkin, J. Cameron, Multiagent mission specification and execution, Autonomous Robots, 4(1), 1997, 29-57.
S. Koenig, C. Tovey, Y. Smirnov, Performance Bounds for Planning in Unknown Terrain, 2001.
B. Brumitt, A. Stentz, GRAMMPS: a generalized mission planner for multiple mobile robots. In Proceedings of the International Conference on Robotics and Automation, 1998.
M. Hebert, R. McLachlan, P. Chang, Experiments with driving modes for urban robots, Proceedings of the SPIE Mobile Robots, 1999.
L. Matthies, Y. Xiong, R. Hogg, D. Zhu, A. Rankin, B. Kennedy, M. Hebert, R. Maclachlan, C. Won, T. Frost, G. Sukhatme, M. McHenry, S. Goldberg, A portable, autonomous, urban reconnaissance robot. Proceedings of the International Conference on Intelligent Autonomous Systems, 2000.
A. Stentz and M. Hebert, A complete navigation system for goal acquisition in unknown environments. Autonomous Robots, 2(2), 1995, 127-145.
S. Thayer, B. Digney, M. Diaz, A. Stentz, B. Nabbe, M. Hebert, Distributed robotic mapping of extreme environments. In Proceedings of the SPIE: Mobile Robots XV and Telemanipulator and Telepresence Technologies VII, Volume 4195, 2000.
P. Hart, N. Nilsson, B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths in Graphs, IEEE Transactions on Systems Science and Cybernetics, SSC-4(2), 1968, 100-107.
G. Ramalingam, T. Reps, On the computational complexity of dynamic graph problems, Theoretical Computer Science 158 (1-2), 1996, 233-277.

References (in order of their appearance)

S. Koenig, M. Likhachev, Incremental A*, Advances in Neural Information Processing Systems, 2001.
M. Likhachev, S. Koenig, Lifelong Planning A* and Dynamic A* Lite: The Proofs, 2001.
B. Nebel and J. Koehler, Plan reuse versus plan generation: A theoretical and empirical analysis, Artificial Intelligence, 76(1-2), 1995, 427-454.
A. Stentz, Optimal and Efficient Path Planning for Partially-Known Environments, Proceedings of the International Conference on Robotics and Automation, 1994, 3310-3317.
S. Koenig, M. Likhachev, D* Lite, Proceedings of the National Conference on Artificial Intelligence, 2002.
A. Stentz. The focussed D* algorithm for real-time replanning. In Proceedings of the International Joint Conference on Artificial Intelligence, 1652-1659, 1995.
S. Koenig, D. Furcy, C. Bauer, Heuristic Search-Based Replanning, Proceedings of the International Conference on Artificial Intelligence Planning Systems, 2002.
B. Bonet, H. Geffner, Heuristic Search Planner 2.0, Artificial Intelligence Magazine 22(3), 2001, 77-80.
Y. Liu, S. Koenig, D. Furcy, Speeding up the calculation of the heuristics for heuristic search-based planning, Proceedings of the National Conference on Artificial Intelligence, 2002.

Greedy On-Line Planning and Lifelong Planning Artificial Intelligence

Related Work:

K. Hammond. Explaining and repairing plans that fail, Artificial Intelligence 45, 1990, 173-228.
R. Simmons. A theory of debugging plans and interpretations, in: Proceedings of the National Conference on Artificial Intelligence, 1988, 94-99.
A. Gerevini, I. Serina, Fast plan adaptation through planning graphs: Local and systematic search techniques, in: proceedings of the International Conference on Artificial Intelligence Planning and Scheduling, 2000, 112-121.
J. Koehler, Flexible plan reuse in a formal framework, in: C. Baeckstroem, E. Sandewall (Eds.), Current Trends in AI Planning, IOS Press, 1994, 171-184.
M. Veloso, Planning and Learning by Analogical Reasoning, Springer, 1994.
R. Alterman, Adaptive Planning, Cognitive Science 12(3), 1988, 393-421.
S. Kambhampati, J. Hendler, A validation-structure-based theory of plan modification and reuse, Artificial Intelligence 55, 1992, 193-258.
S. Edelkamp, Updating Shortest Paths, Proceedings of the European Conference on Artificial Intelligence, 1998, 655-659.
S. Hanks, D. Weld, A domain-independent algorithm for plan adaptation, Journal of Artificial Intelligence Research 2, 1995, 319-360.

... and many more

Greedy On-Line Planning and Lifelong Planning Algorithm Theory

Related Work:

G. Ausiello, G. Italiano, A. Marchetti-Spaccamela, U. Nanni, Incremental algorithms for minimal length paths, Journal of Algorithms 12(4), 1991, 615-638.
S. Even, H. Gazit, Updating distance in dynamic graphs, Methods of Operations Research 49, 1985, 371-387.
E. Feuerstein, A. Marchetti-Spaccamela, Dynamic algorithms for shortest paths in planar graphs, Theoretical Computer Science 116(2), 1993, 359-371.
P. Franciosa, D. Frigioni, R. Giaccio, Semi-dynamic breadth-first search in digraphs, Theoretical Computer Science 250(1-2), 2001, 201-217.
D. Friogioni, A. Marchetti-Spaccamela, U. Nanni, Fully dynamic output bounded single source shortest path problem, in: Prodeedings of the Symposium on Discrete Algorithms, 1996, 212-221.
S. Goto, A. Sangiovanni-Vincentelli, A new shortest path updating algorithm, Networks 8(4), 1978, 341-372.
G. Italiano, Finding paths and deleting edges in directed acyclic graphs, Information Processing Letters 28(1), 1988, 5-11.
P. Klein, S. Subramanian, Fully dynamic approximation schemes for shortest path problems in planar graphs, in: Proceedings of the International Workshop on Algorithms and Data Structures, 1993, 443-451.
C. Lin, R. Chang, On the dynamic shortest path problem, Journal of Information Processing 13(4), 1990, 470-476.
H. Rohnert, A dynamization of the all pairs least cost path problem, in: Proceedings of the Symposium on Theoretical Aspects of Computer Science, 1985, 279-286.
P. Spira, A. Pan, On finding and updating spanning trees and shortest paths, SIAM Journal on Computing 4, 1975, 375-380.

... and many more

Greedy On-Line Planning and Lifelong Planning Robotics

Related Work:

V. Lumelsky and A. Stepanov. Path planning strategies for point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica, 2:403-430, 1987.
M. Barbehenn and S. Hutchinson. Efficient search and hierarchical motion planning by dynamically maintaining single-source shortest paths trees. IEEE Transactions on Robotics and Automation, 11(2):198-214, 1995.
T. Ersson and X. Hu. Path planning and navigation of mobile robots in unknown environments. In Proceedings of the International Conference on Intelligent Robots and Systems, 2001.
Y. Huiming, C. Chia-Jung, S. Tong, and B. Qiang. Hybrid evolutionary motion planning using follow boundary repair for mobile robots. Journal of Systems Architecture, 47(7):635-647, 2001.
L. Podsedkowski, J. Nowakowski, M. Idzikowski, and I. Vizvary. A new solution for path planning in partially known or unknown environments for nonholonomic mobile robots. Robotics and Autonomous Systems, 34:145-152. 2001
M. Tao, A. Elssamadisy, N. Flann, and B. Abbott. Optimal route re-planning for mobile robots: A massively parallel incremental A* algorithm. In International Conference on Robotics and Automation, pages 2727-2732, 1997.
K. Trovato. Differential A*: An adaptive search method illustrated with robot path planning for moving obstacles and goals, and an uncertain environment. Journal of Pattern Recognition and Artificial Intelligence, 4(2), 1990.

... and many more

Greedy On-Line Planning and Lifelong Planning Theoretical Results

Related Work:

S. Carlsson, H. Jonsson, Computing a shortest watchman path in a simple polygon in polynomial time, in: S. Akl, F. Dehne, J. Sack, N. Santoro (Eds.), Proceedings of the Workshop on Algorithms and Data Structures, Vol. 955 of Lecture Notes in Computer Science, Springer, 1995, 122-134.
X. Tan, T. Hirata, Constructing shortest watchman routes by divide-and-conquer, in: K. Ng, P. Raghavan, N. Balasubramanian, F. Chin (Eds.), Proceedings of the International Symposium on Algorithms and Computation, Vol. 762 of Lecture Notes in Computer Science, Springer, 1993, 68-77.
S. Ntafos, Watchman routes under limited visibility, in: Proceedings of the Canadian Conference on Computational Geometry, 1990, 89-92.
X. Deng, T. Kameda, C. Papadimitriou, How to learn an unknown environment I: the rectilinear case, Journal of the ACM 45(2), 1998, 215-245.
F. Hoffman, C. Icking, R. Klein, K. Kriegel, A competitive strategy for learning a polygon, in: Proceedings of the Symposium on Discrete Algorithms, 1997, 166-174.
V. Lumelsky, Algorithmic and complexity issues of robot motion in an uncertain environment, Journal of Complexity 3, 1987, 146-182.
A. Blum, P. Raghavan, B. Schieber, Navigating in unfamiliar geometric terrain, SIAM Journal on Computing 26(1), 1997, 110-137.
C. Icking, R. Klein, E. Langetepe, An optimal competitive strategy for walking in streets, in: C. Meinel, S. Tison (Eds.), Proceedings of the Symposium on Theoretical Aspects of Computer Science, Vol. 1563 of Lecture notes in Computer Science, Springer, 1999, 110-120.
X. Deng, C. Papadimitriou, Exploring an unknown graph, in: Proceedings of the Symposium on Foundations of Computer Science, 1990, 355-361.
S. Albers, M. Henzinger, Exploring unknown environments, in: Proceedings of the Symposium on Theory of Computing, 1997, 416-425.

... and many more

Lifelong Planning Techniques - Our Work

Please see

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

We gratefully acknowledge funding from NSF and IBM.