

# Fast Optimal Replanning

Sven Koenig



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

Collaborators:

Maxim Likhachev,

David Furcy, Yaxin Liu

(Additional Programming: Colin Bauer, William Halliburton)

## Motivation



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

...

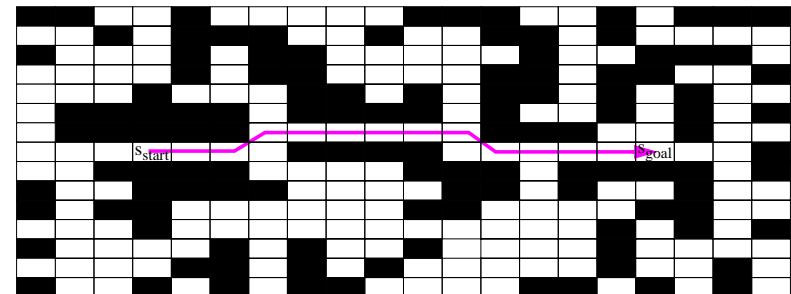
## Structure of the Talk

- overview of incremental heuristic search
- Lifelong Planning A\* (LPA\*) and its properties
- applications of incremental heuristic search
  - symbolic planning (with HSP)
    - continual planning
    - one-time planning
  - mobile robotics
    - mapping
    - goal-directed navigation in unknown terrain
  - computer games
  - reinforcement learning and on-line dynamic programming
  - control (with the Parti-Game algorithm)

## Path Planning - Example

we assume here that the robot can move in eight directions

original eight-connected gridworld





## Path Planning - Lifelong Planning A\*

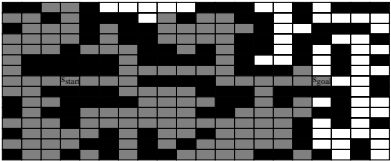
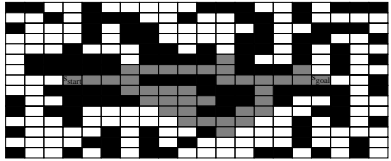
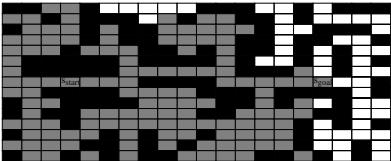
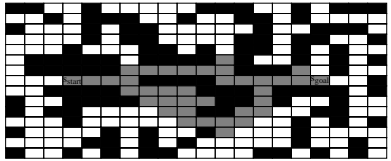
	uninformed search	heuristic search
complete search	Breadth-First Search	A* [Hart, Nilsson, Raphael, 1968]
incremental search	DynamicSWSF-FP with early termination (our addition) [Ramalingam, Reps, 1996]	Lifelong Planning A*

Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

9 of 62

## Path Planning - Experimental Evaluation

original eight-connected gridworld

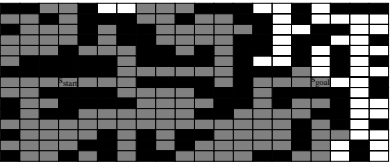
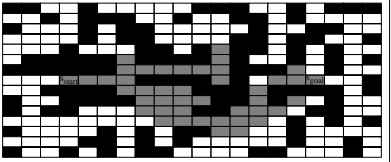
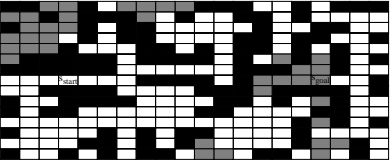
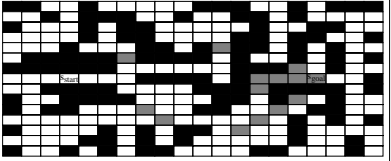
	uninformed search	heuristic search
complete search		
incremental search		Lifelong Planning A* 

Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

10 of 62

## Path Planning - Experimental Evaluation

changed eight-connected gridworld

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

Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

11 of 62

## Path Planning - Experimental Evaluation

changed eight-connected gridworld

	uninformed search	heuristic search
complete search	ve = 1331.7 +/- 4.4 va = 26207.2 +/- 84.0 hp = 5985.3 +/- 19.7	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

Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

12 of 62

# Path Planning - Lifelong Planning A\*

[Koenig, Likhachev, 2001]

```

procedure CalculateKey(s)
  return [min(g(s), rhs(s)) + h(s,sgoal); min(g(s), rhs(s))];
procedure Initialize()
  U = ∅;
  for all s ∈ S rhs(s) = g(s) = ∞
  rhs(sstart) = 0;
  U.Insert(sstart, CalculateKey(sstart));
procedure UpdateVertex(u)
  if (u ≠ sstart) rhs(u) = minv' ∈ 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(sgoal) OR rhs(sgoal) ≠ g(sgoal))
    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);
  
```

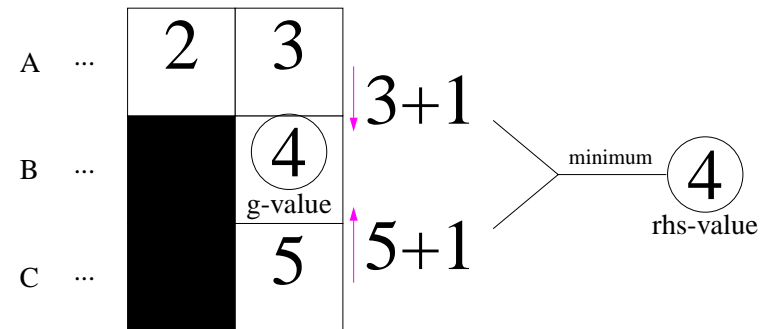
# Path Planning - Lifelong Planning A\*

- 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

# Path Planning - Lifelong Planning A\*

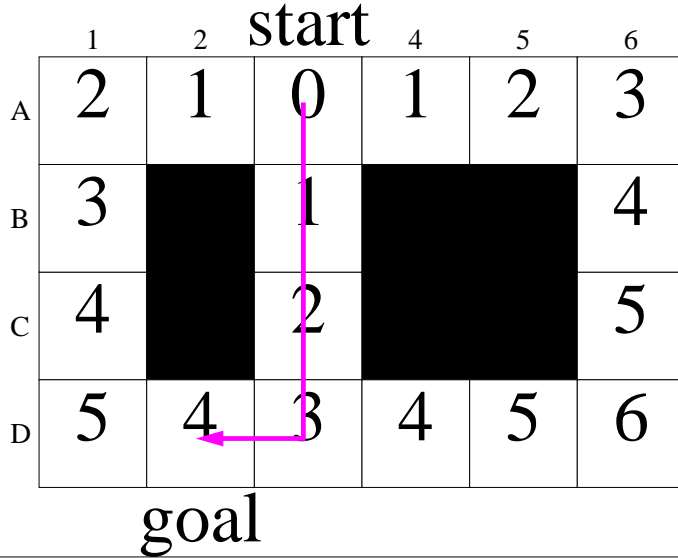
	1	2	start	4	5	6
A	2	1	0	1	2	3
B	3		1			4
C	4		2			5
D	5	4	3	4	5	6
			goal			

# 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<sub>goal</sub>); min(g(s),rhs(s))]  
 smaller priorities first, according to a lexicographic ordering

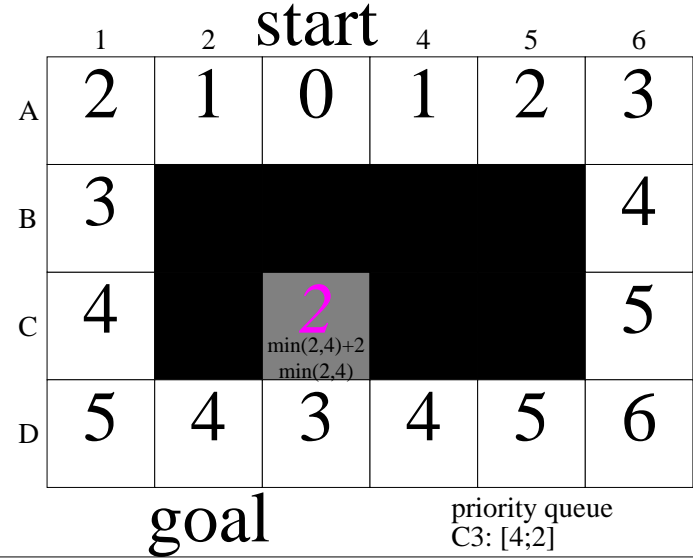
### Path Planning - Lifelong Planning A\*



Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

17 of 62

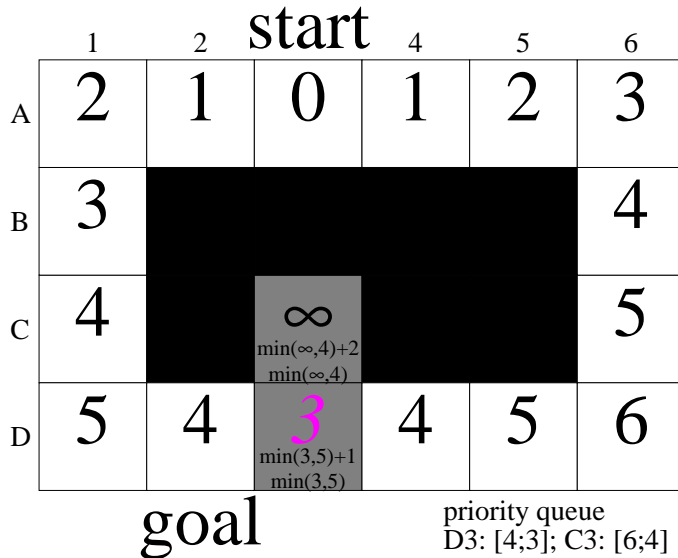
### Path Planning - Lifelong Planning A\*



Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

18 of 62

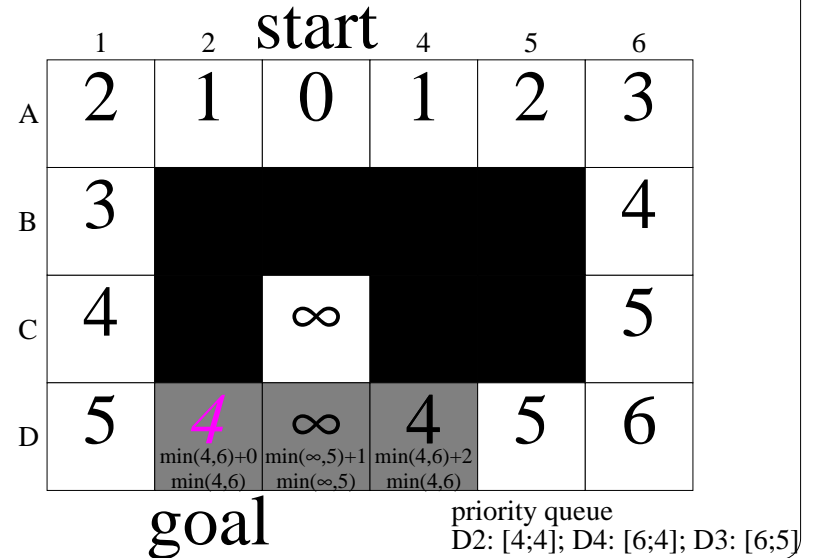
### Path Planning - Lifelong Planning A\*



Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

19 of 62

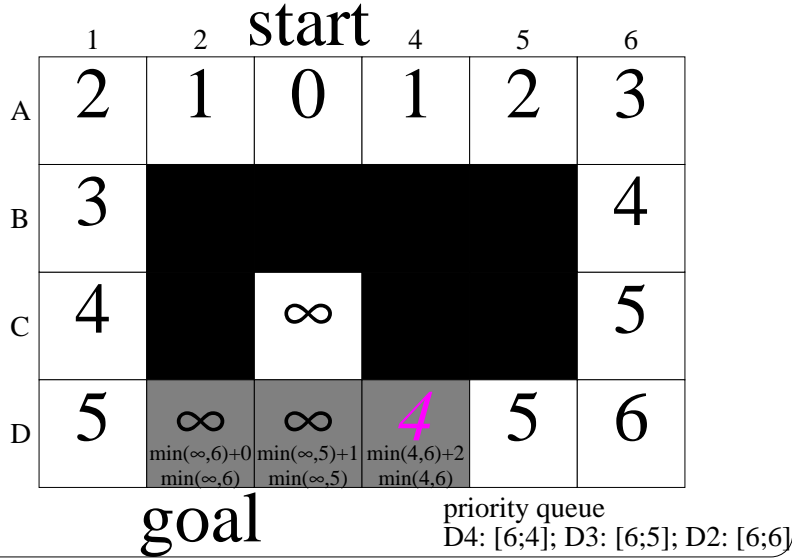
### Path Planning - Lifelong Planning A\*



Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

20 of 62

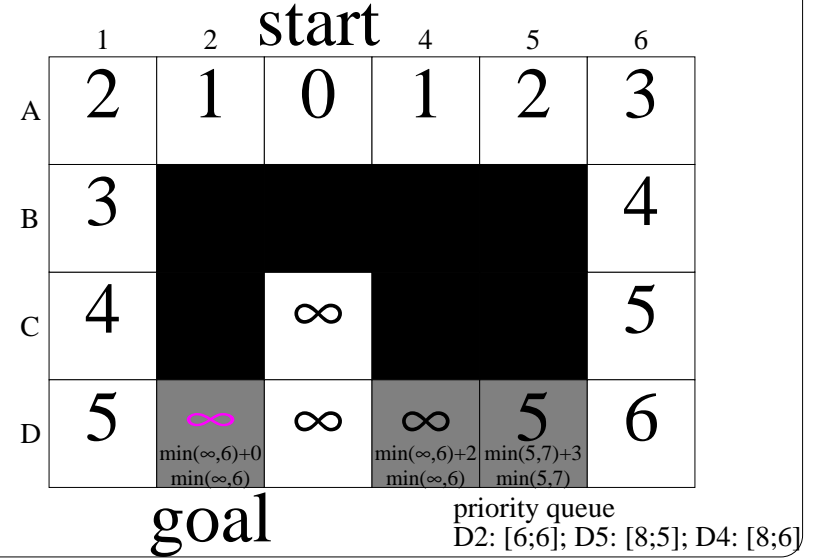
### Path Planning - Lifelong Planning A\*



Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

21 of 62

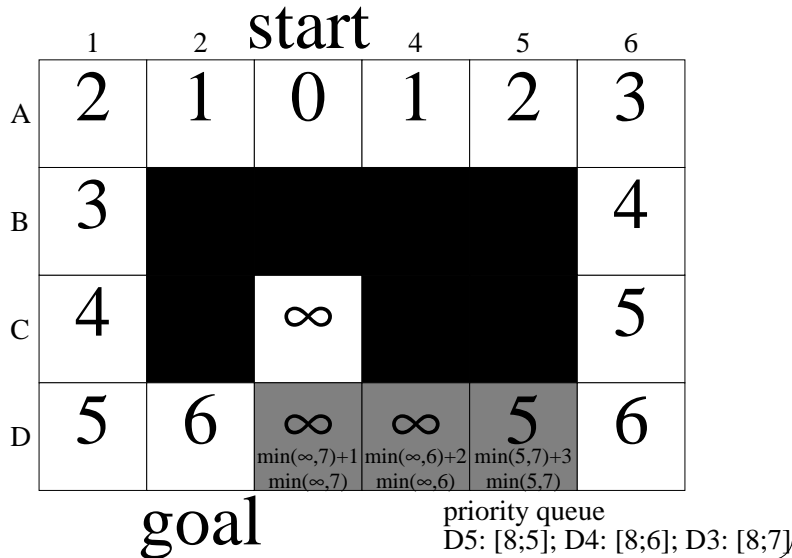
### Path Planning - Lifelong Planning A\*



Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

22 of 62

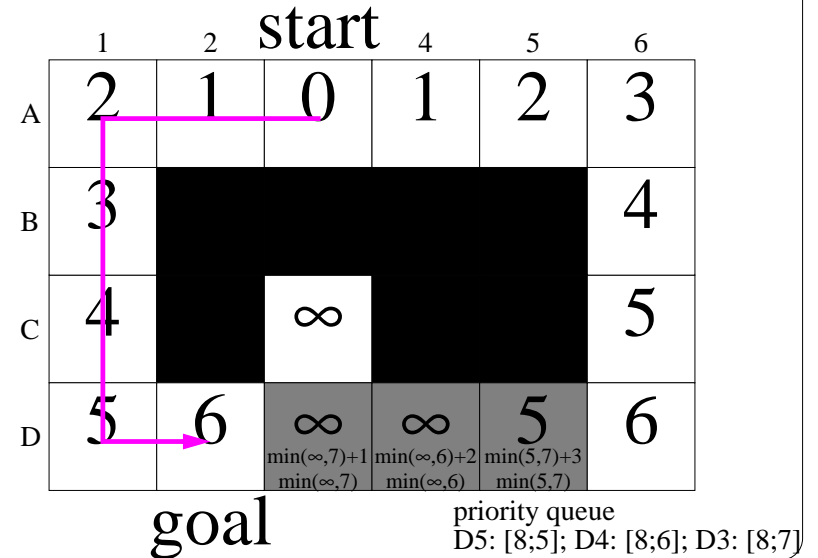
### Path Planning - Lifelong Planning A\*



Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

23 of 62

### Path Planning - Lifelong Planning A\*



Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

24 of 62

## Path Planning - Lifelong Planning A\*

Theorem: [Likhachev and Koenig, 2001]

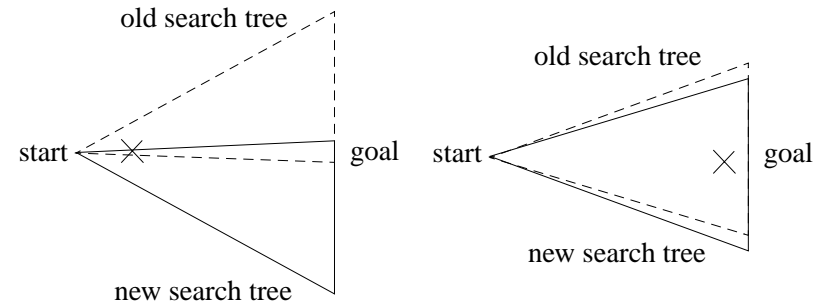
ComputeShortestPath() expands every vertex at most twice and thus terminates.

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]  
However, the overhead of LPA\* is bounded.



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

## Applications

- route planning
    - in traffic networks
    - in computer networks
- 
- symbolic planning (with HSP)
    - continual planning
    - one-time planning
  - mobile robotics
    - mapping
    - goal-directed navigation in unknown terrain
  - computer games
  - reinforcement learning and on-line dynamic programming
  - control (with the Parti-Game algorithm)

## 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

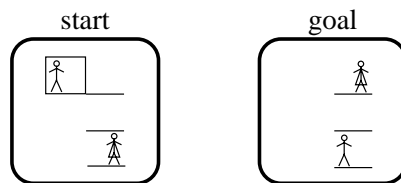
- 
- lifelong planning

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



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  $f_i$  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]

planning problem 1

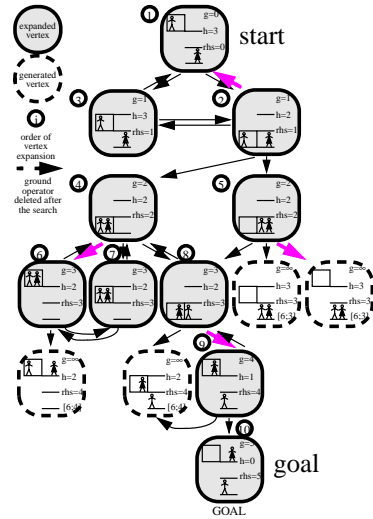
planning problem 2

planning problem 3

## Symbolic Planning (with HSP) - Continual Planning

first search in the elevator domain

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



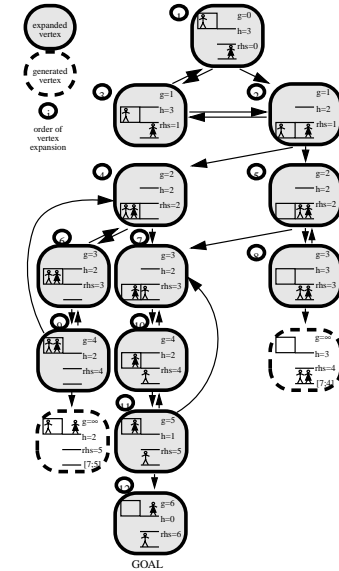
Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

33 of 62

## Symbolic Planning (with HSP) - Continual Planning

second search in the elevator domain (from scratch)

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

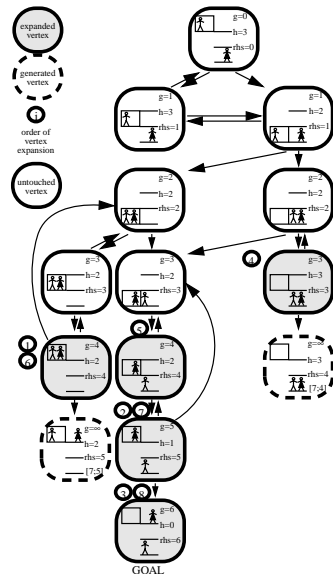


Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

34 of 62

## Symbolic Planning (with HSP) - Continual Planning

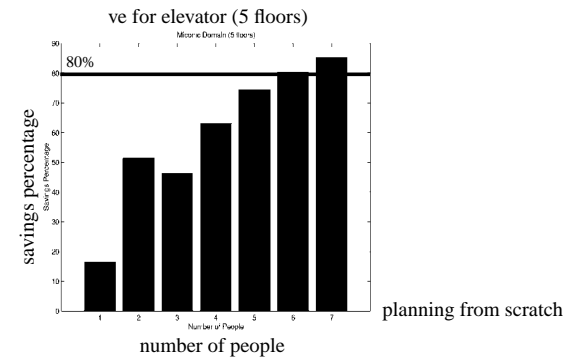
second search in the elevator domain (with SHERPA)



Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

35 of 62

## Symbolic Planning (with HSP) - Continual Planning

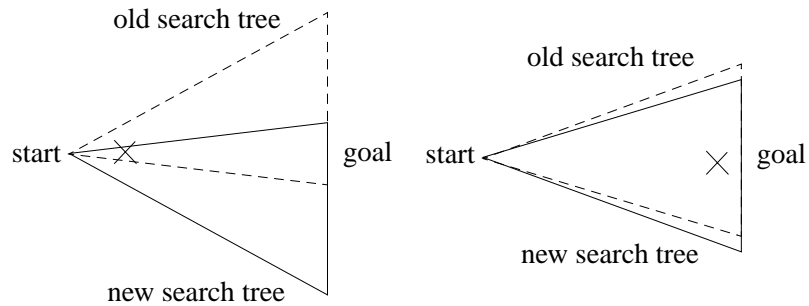


SHERPA achieves speedups up to 80 percent

Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

36 of 62

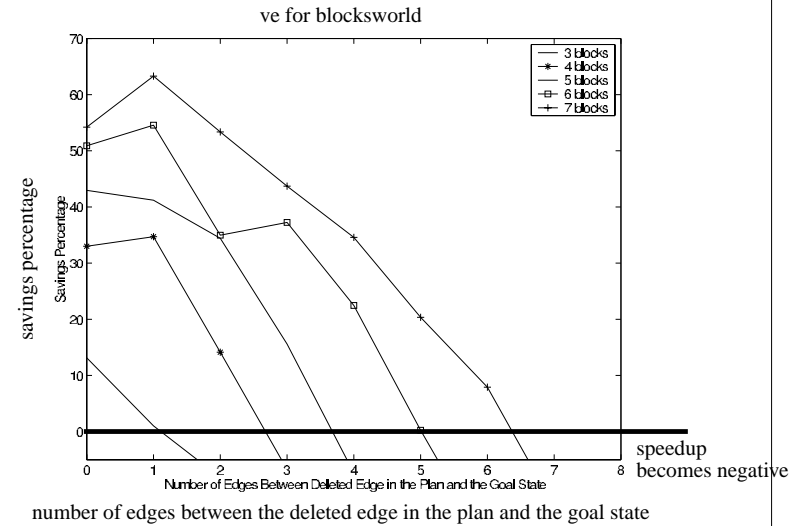
## Symbolic Planning (with HSP) - Continual Planning



Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

37 of 62

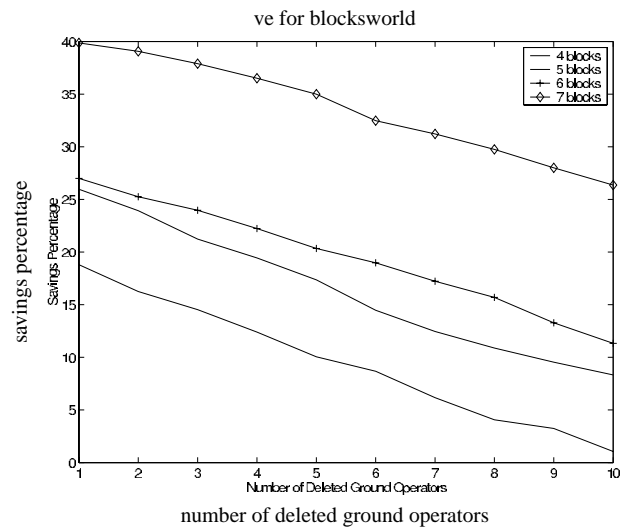
## Symbolic Planning (with HSP) - Continual Planning



Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

38 of 62

## Symbolic Planning (with HSP) - Continual Planning



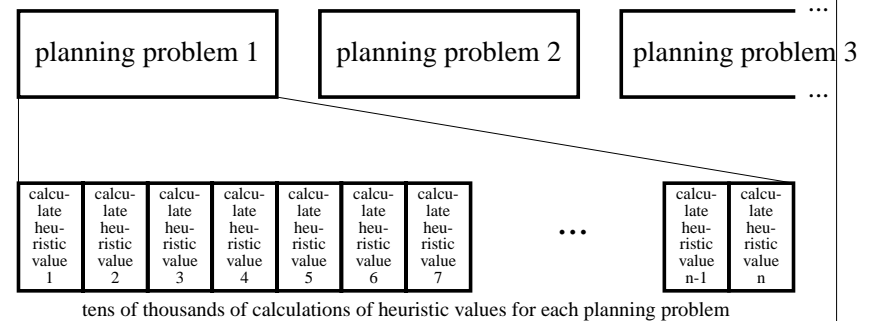
Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

39 of 62

## 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

Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

40 of 62

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

## PINCH

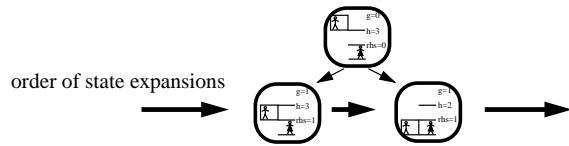
### Prioritized, INCremental Heuristics calculation

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

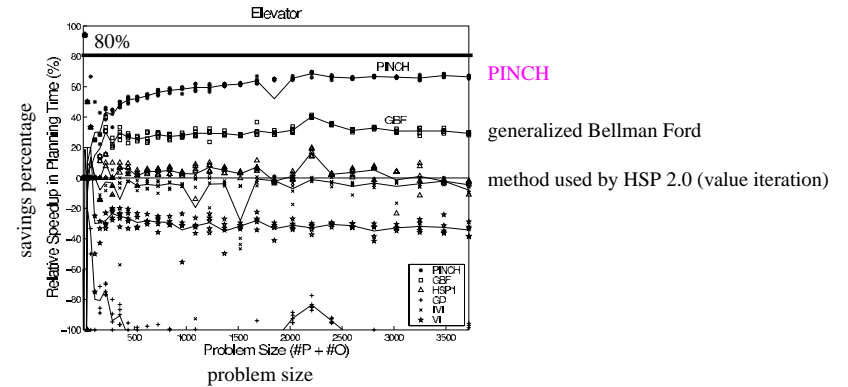
$$h_{add}(state) = \sum_{\text{proposition in goal state}} g_{state}(\text{proposition})$$

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

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



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



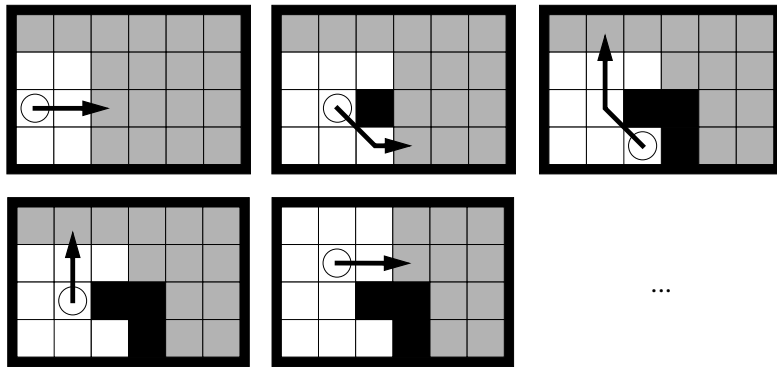
PINCH achieves speedups up to (another!) 80 percent.

## Mobile Robotics - Mapping

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.

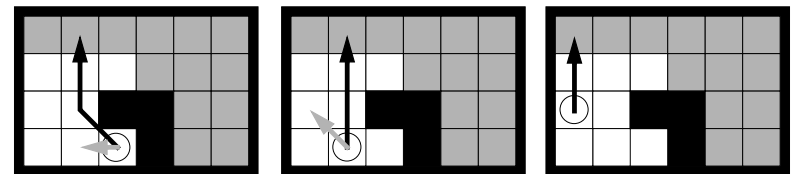
[Koenig, Tovey, Halliburton, 2001] [Thrun et al. 1998] [Romero, Morales, Sucar, 2001]



## Mobile Robotics - Mapping

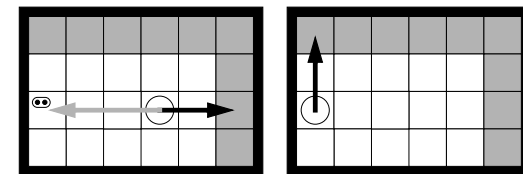
we assume here that the robot can move in eight directions

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]

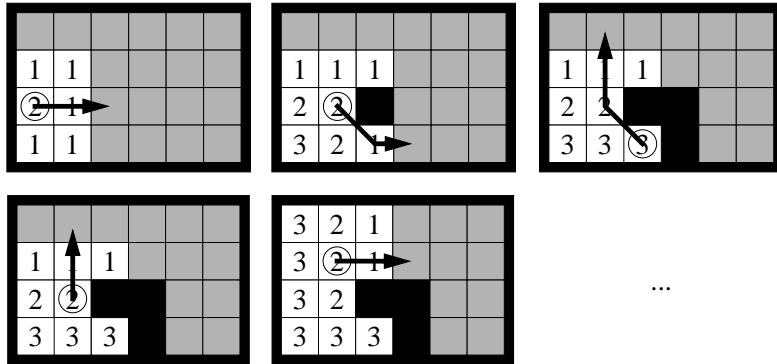
does not need to be in control of the robot at all times (“reactive planning”)



# Mobile Robotics - Mapping

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.



Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

# Mobile Robotics - Mapping

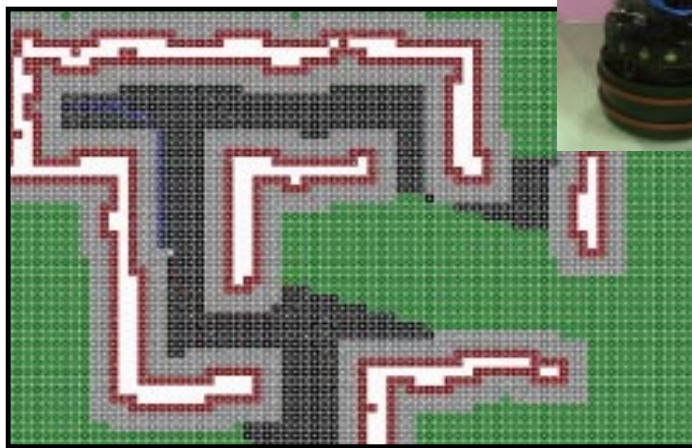


20 feet

28 feet

Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

# Mobile Robotics - Mapping

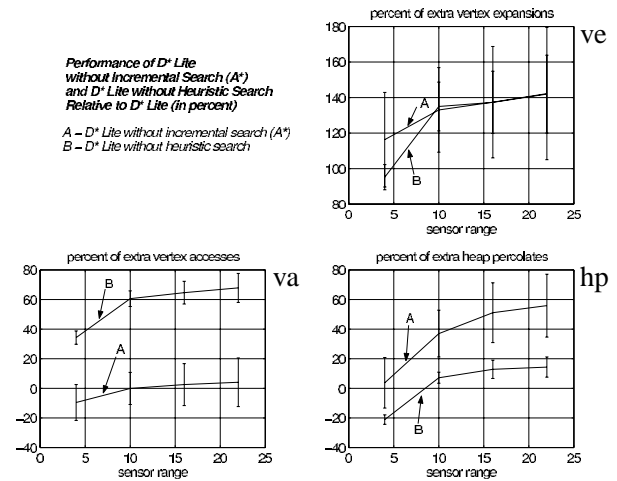


Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

# Mobile Robotics - Mapping

Performance of D\* Lite without Incremental Search (A\*) and D\* Lite without Heuristic Search Relative to D\* Lite (in percent)

A = D\* Lite without incremental search (A\*)  
 B = D\* Lite without heuristic search



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

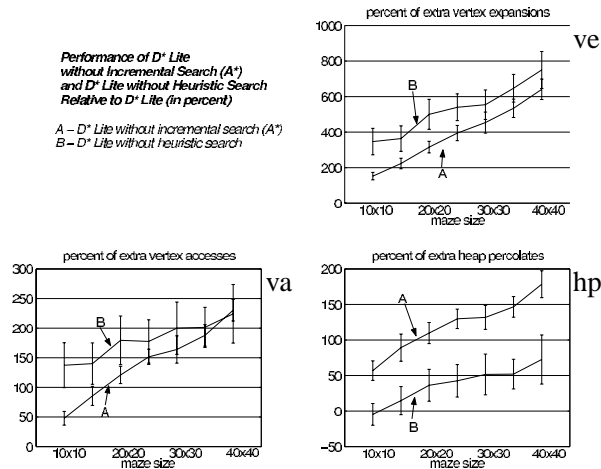
Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.



## Mobile Robotics - Navigation in Unknown Terrain

Performance of D\* Lite without Incremental Search (A\*) and D\* Lite without Heuristic Search Relative to D\* Lite (in percent)

A - D\* Lite without incremental search (A\*)  
B - D\* Lite without heuristic search



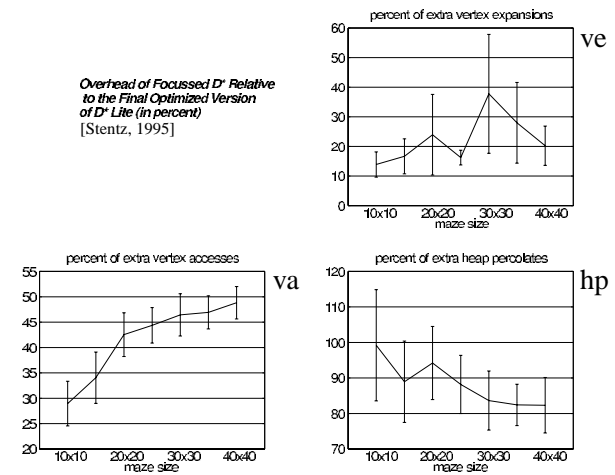
A = overhead of LPA\* (D\* Lite) without incremental Search (A\*)  
B = overhead of LPA\* (D\* Lite) without heuristic search

Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

53 of 62

## Mobile Robotics - Navigation in Unknown Terrain

Overhead of Focussed D\* Relative to the Final Optimized Version of D\* Lite (in percent) [Stentz, 1995]



Focussed Dynamic A\* (D\*) by Anthony Stentz, probably the first incremental heuristic search algorithm (way ahead of its time) and our original motivation for developing LPA\*

Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

54 of 62

## Game Playing



"Total Annihilation" by Cavedog Entertainment

Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

55 of 62

## Reinforcement Learning and On-Line DP

while there exists at least one state with  $g(s) \neq rhs(s)$   
pick a state  $s$  with  $g(s) \neq rhs(s)$  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\*

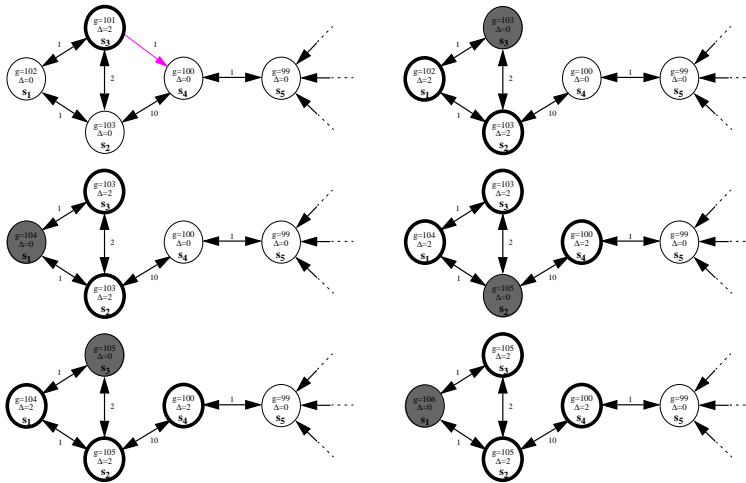
- chooses the  $g$ -value of which state to update
- updates the  $g$ -value of the chosen state in a particular way
- terminates immediate once a shortest path is found
- uses heuristics to focus the search
- minimizes the worst-case plan-execution cost for MDPs

Fast Optimal Replanning; (c) Sven Koenig; Georgia Tech; January 2002.

56 of 62

# Reinforcement Learning and On-Line DP

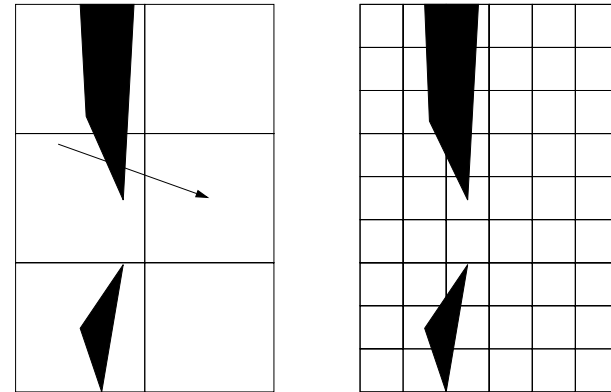
Prioritized Sweeping [Moore and Atkeson; 1993]



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

# Control (with the Parti-Game algorithm)

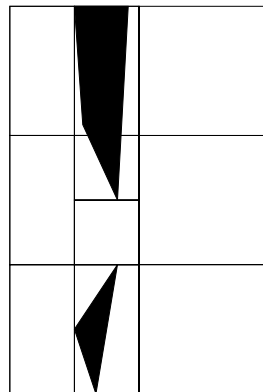
state spaces of control problems are often continuous and sometimes high-dimensional



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

# Control (with the Parti-Game algorithm)

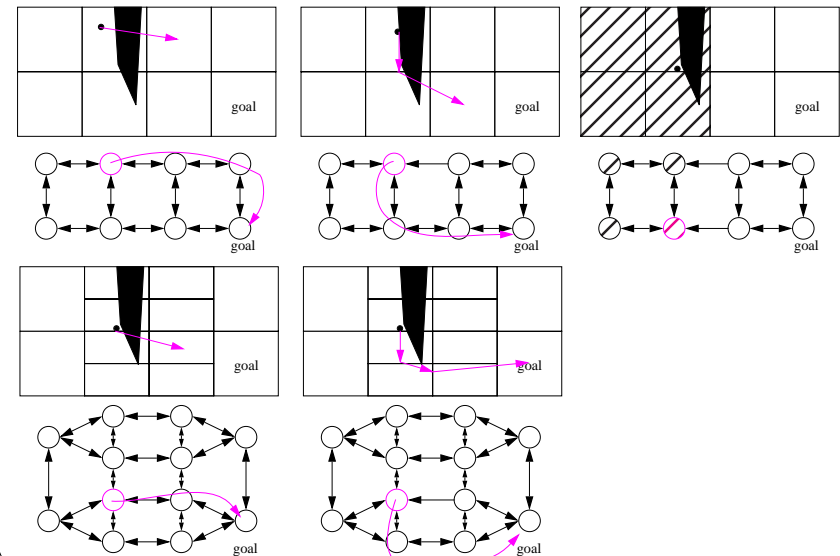
Parti-Game algorithm [Moore and Atkeson; 1995]



nonuniform discretization avoids these problems

# Control (with the Parti-Game algorithm)

here: we use a deterministic version for illustration - we actually use Minimax 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

## More Information on Fast Optimal Replanning

Please see  
<http://www.cc.gatech.edu/fac/Sven.Koenig/fastreplanning.html>

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.