

CSE-571

Deterministic Path Planning in Robotics

*Courtesy of Maxim Likhachev
Carnegie Mellon University*

1

Motion/Path Planning

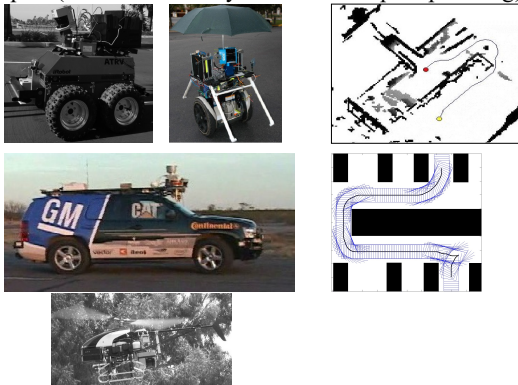
- Task:
find a feasible (and cost-minimal) path/motion from the current configuration of the robot to its goal configuration (or one of its goal configurations)
- Two types of constraints:
environmental constraints (e.g., obstacles)
dynamics/kinematics constraints of the robot
- Generated motion/path should (objective):
be any feasible path
minimize cost such as distance, time, energy, risk, ...

CSE-571: Courtesy of Maxim Likhachev, CMU

2

Motion/Path Planning

Examples (of what is usually referred to as path planning):

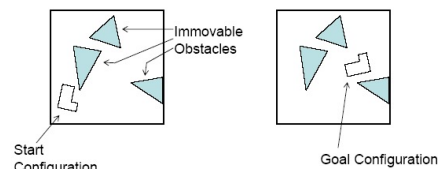


CSE-571: Courtesy of Maxim Likhachev, CMU

3

Motion/Path Planning

Examples (of what is usually referred to as motion planning):



Piano Movers' problem

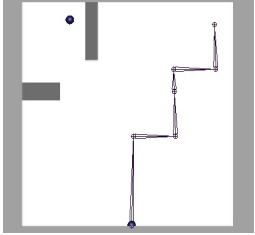
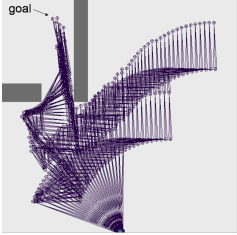
the example above is borrowed from www.cs.cmu.edu/~awm/tutorials

CSE-571: Courtesy of Maxim Likhachev, CMU

4

Motion/Path Planning

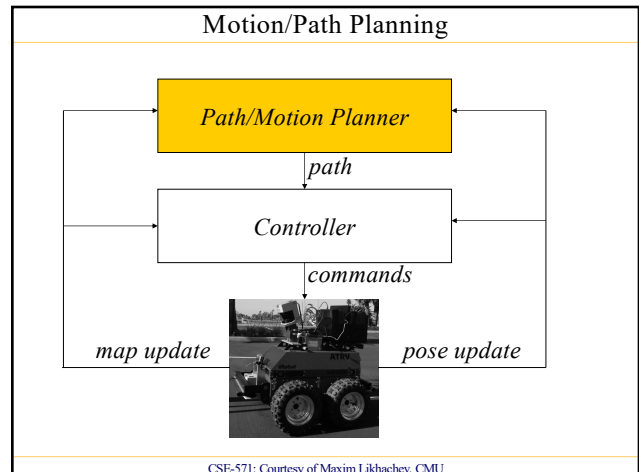
Examples (of what is usually referred to as motion planning):

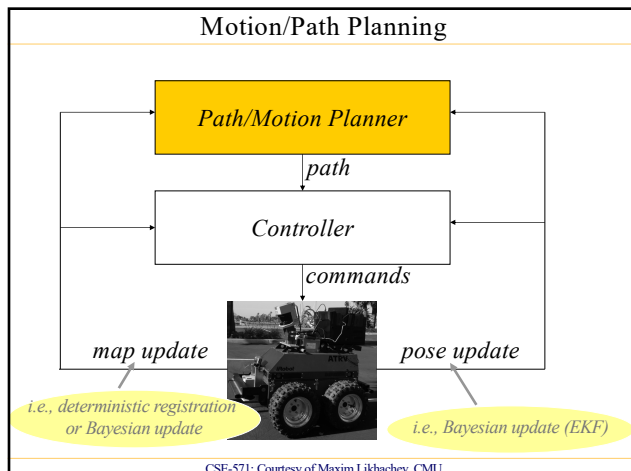
Planned motion for a 6DOF robot arm

CSE-571: Courtesy of Maxim Likhachev, CMU

5



6



7

Uncertainty and Planning

- Uncertainty can be in:
 - prior environment (i.e., door is open or closed)
 - execution (i.e., robot may slip)
 - sensing environment (i.e., seems like an obstacle but not sure)
 - pose
- Planning approaches:
 - deterministic planning:
 - assume some (i.e., most likely) environment, execution, pose
 - plan a single least-cost trajectory under this assumption
 - re-plan as new information arrives
 - planning under uncertainty:
 - associate probabilities with some elements or everything
 - plan a policy that dictates what to do for each outcome of sensing/action and minimizes expected cost-to-goal
 - re-plan if unaccounted events happen

CSE-571: Courtesy of Maxim Likhachev, CMU

8

Uncertainty and Planning

- Uncertainty can be in:
 - prior environment (i.e., door is open or closed)
 - execution (i.e., robot may slip)
 - sensing environment (i.e., seems like an obstacle but not sure)
 - pose
- Planning approaches:
 - deterministic planning:
 - assume some (i.e., most likely) environment
 - plan a single least-cost trajectory under this assumption
 - re-plan as new information arrives
 - planning under uncertainty:
 - associate probabilities with some elements or everything
 - plan a policy that dictates what to do for each outcome of sensing/action and minimizes expected cost-to-goal
 - re-plan if unaccounted events happen

*re-plan every time
sensory data arrives or
robot deviates off its path
re-planning needs to be FAST*

CSE-571: Courtesy of Maxim Likhachev, CMU

9

Uncertainty and Planning

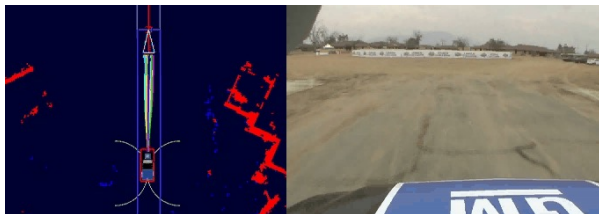
- Uncertainty can be in:
 - prior environment (i.e., door is open or closed)
 - execution (i.e., robot may slip)
 - sensing environment (i.e., seems like an obstacle but not sure)
 - pose
- Planning approaches:
 - deterministic planning:
 - assume some (i.e., most likely) environment, execution, pose
 - plan a single least-cost trajectory under this assumption
 - re-plan as new information arrives
 - planning under uncertainty:
 - associate probabilities with some elements or everything
 - plan a policy that dictates what to do for each outcome of sensing/action and minimizes expected cost-to-goal
 - re-plan if unaccounted events happen

computationally MUCH harder

CSE-571: Courtesy of Maxim Likhachev, CMU

10

Example



Urban Challenge Race, CMU team, planning with Anytime D*

CSE-571: Courtesy of Maxim Likhachev, CMU

11

Outline

- Deterministic planning
 - constructing a graph
 - search with A*
 - search with D*

CSE-571: Courtesy of Maxim Likhachev, CMU

12

Outline

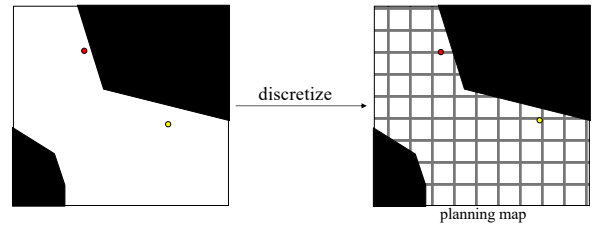
- Deterministic planning
 - constructing a graph
 - search with A*
 - search with D*

CSE-571: Courtesy of Maxim Likhachev, CMU

13

Planning via Cell Decomposition

- Approximate Cell Decomposition:
 - overlay uniform grid over the C-space (discretize)

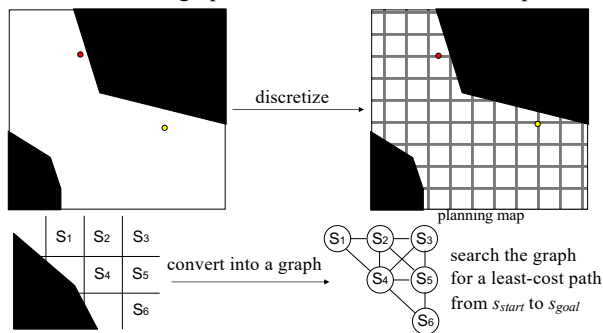


CSE-571: Courtesy of Maxim Likhachev, CMU

14

Planning via Cell Decomposition

- Approximate Cell Decomposition:
 - construct a graph and search it for a least-cost path

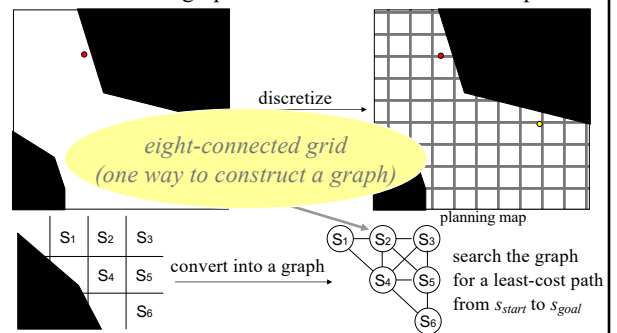


CSE-571: Courtesy of Maxim Likhachev, CMU

15

Planning via Cell Decomposition

- Approximate Cell Decomposition:
 - construct a graph and search it for a least-cost path

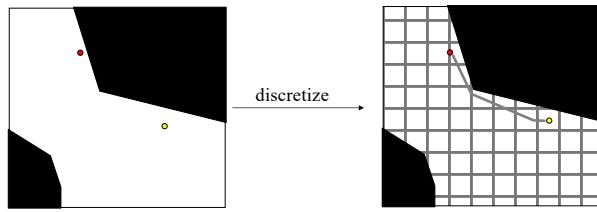


CSE-571: Courtesy of Maxim Likhachev, CMU

16

Planning via Cell Decomposition

- Approximate Cell Decomposition:
 - construct a graph and search it for a least-cost path
 - VERY popular due to its simplicity and representation of arbitrary obstacles
 - Problem: transitions difficult to execute on non-holonomic robots



CSE-571: Courtesy of Maxim Likhachev, CMU

17

Planning via Cell Decomposition

- Graph construction:
 - lattice graph

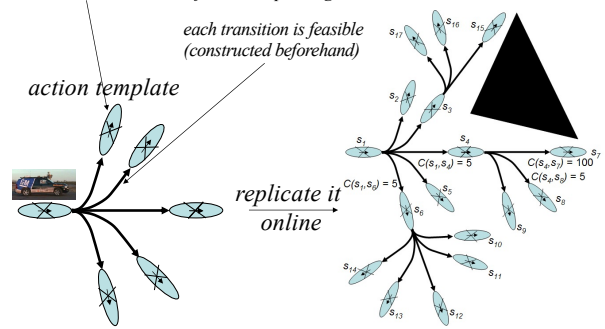
outcome state is the center of the corresponding cell

each transition is feasible
(constructed beforehand)

action template



replicate it
online



CSE-571: Courtesy of Maxim Likhachev, CMU

18

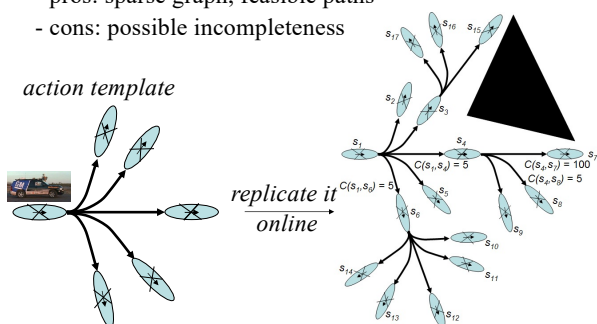
Planning via Cell Decomposition

- Graph construction:
 - lattice graph
 - pros: sparse graph, feasible paths
 - cons: possible incompleteness

action template



replicate it
online



CSE-571: Courtesy of Maxim Likhachev, CMU

19

Outline

- Deterministic planning
 - constructing a graph
 - search with A*
 - search with D*
- Planning under uncertainty
 - Markov Decision Processes (MDP)
 - Partially Observable Decision Processes (POMDP)

CSE-571: Courtesy of Maxim Likhachev, CMU

20

A* Search

- Computes optimal g-values for relevant states

at any point of time:

CSE-571: Courtesy of Maxim Likhachev, CMU

21

A* Search

- Computes optimal g-values for relevant states

at any point of time:

one popular heuristic function – Euclidean distance

CSE-571: Courtesy of Maxim Likhachev, CMU

22

A* Search

- Computes optimal g-values for relevant states

ComputePath function
while(s_{goal} is not expanded)
 remove s with the smallest $[f(s) = g(s) + h(s)]$ from $OPEN$;
 insert s into $CLOSED$;
 for every successor s' of s such that s' not in $CLOSED$
 if $g(s') > g(s) + c(s, s')$
 $g(s') = g(s) + c(s, s')$;
 insert s' into $OPEN$;

$CLOSED = \{\}$
 $OPEN = \{s_{start}\}$
next state to expand: s_{start}

Maxim Likhachev, University of Pennsylvania

23

A* Search

- Computes optimal g-values for relevant states

ComputePath function
while(s_{goal} is not expanded)
 remove s with the smallest $[f(s) = g(s) + h(s)]$ from $OPEN$;
 insert s into $CLOSED$;
 for every successor s' of s such that s' not in $CLOSED$
 if $g(s') > g(s) + c(s, s')$
 $g(s') = g(s) + c(s, s')$;
 insert s' into $OPEN$;

$CLOSED = \{\}$
 $OPEN = \{s_{start}\}$
next state to expand: s_{start}

Maxim Likhachev, University of Pennsylvania

24

A* Search

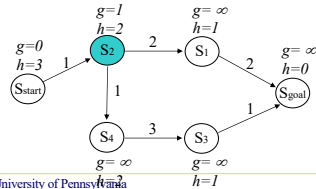
- Computes optimal g-values for relevant states

ComputePath function

```

while( $s_{goal}$  is not expanded)
  remove  $s$  with the smallest  $[f(s) = g(s) + h(s)]$  from  $OPEN$ ;
  insert  $s$  into  $CLOSED$ ;
  for every successor  $s'$  of  $s$  such that  $s'$  not in  $CLOSED$ 
    if  $g(s') > g(s) + c(s, s')$ 
       $g(s') = g(s) + c(s, s')$ ;
      insert  $s'$  into  $OPEN$ ;
  
```

$CLOSED = \{s_{start}\}$
 $OPEN = \{s_2\}$
 next state to expand: s_2



Maxim Likhachev, University of Pennsylvania

25

A* Search

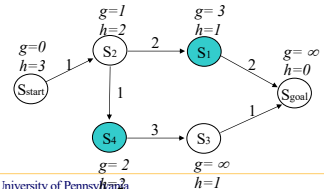
- Computes optimal g-values for relevant states

ComputePath function

```

while( $s_{goal}$  is not expanded)
  remove  $s$  with the smallest  $[f(s) = g(s) + h(s)]$  from  $OPEN$ ;
  insert  $s$  into  $CLOSED$ ;
  for every successor  $s'$  of  $s$  such that  $s'$  not in  $CLOSED$ 
    if  $g(s') > g(s) + c(s, s')$ 
       $g(s') = g(s) + c(s, s')$ ;
      insert  $s'$  into  $OPEN$ ;
  
```

$CLOSED = \{s_{start}, s_2\}$
 $OPEN = \{s_1, s_4\}$
 next state to expand: s_1



Maxim Likhachev, University of Pennsylvania

26

A* Search

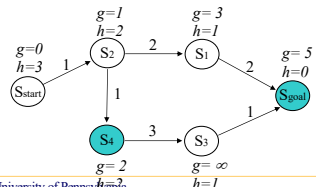
- Computes optimal g-values for relevant states

ComputePath function

```

while( $s_{goal}$  is not expanded)
  remove  $s$  with the smallest  $[f(s) = g(s) + h(s)]$  from  $OPEN$ ;
  insert  $s$  into  $CLOSED$ ;
  for every successor  $s'$  of  $s$  such that  $s'$  not in  $CLOSED$ 
    if  $g(s') > g(s) + c(s, s')$ 
       $g(s') = g(s) + c(s, s')$ ;
      insert  $s'$  into  $OPEN$ ;
  
```

$CLOSED = \{s_{start}, s_2, s_1\}$
 $OPEN = \{s_4, s_{goal}\}$
 next state to expand: s_4



Maxim Likhachev, University of Pennsylvania

27

A* Search

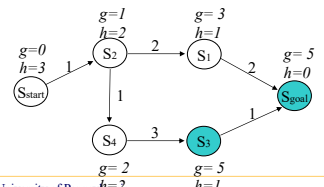
- Computes optimal g-values for relevant states

ComputePath function

```

while( $s_{goal}$  is not expanded)
  remove  $s$  with the smallest  $[f(s) = g(s) + h(s)]$  from  $OPEN$ ;
  insert  $s$  into  $CLOSED$ ;
  for every successor  $s'$  of  $s$  such that  $s'$  not in  $CLOSED$ 
    if  $g(s') > g(s) + c(s, s')$ 
       $g(s') = g(s) + c(s, s')$ ;
      insert  $s'$  into  $OPEN$ ;
  
```

$CLOSED = \{s_{start}, s_2, s_1, s_4\}$
 $OPEN = \{s_3, s_{goal}\}$
 next state to expand: s_{goal}



Maxim Likhachev, University of Pennsylvania

28

A* Search

- Computes optimal g-values for relevant states

ComputePath function

```

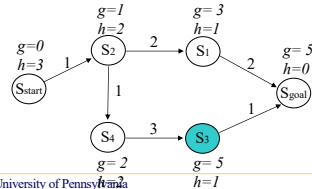
while( $s_{goal}$  is not expanded)
  remove  $s$  with the smallest  $[f(s) = g(s) + h(s)]$  from  $OPEN$ ;
  insert  $s$  into  $CLOSED$ ;
  for every successor  $s'$  of  $s$  such that  $s'$  not in  $CLOSED$ 
    if  $g(s') > g(s) + c(s, s')$ 
       $g(s') = g(s) + c(s, s')$ ;
      insert  $s'$  into  $OPEN$ ;

```

$CLOSED = \{s_{start}, s_2, s_1, s_4, s_{goal}\}$

$OPEN = \{s_3\}$

done



Maxim Likhachev, University of Pennsylvania

29

A* Search

- Computes optimal g-values for relevant states

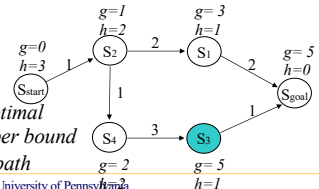
ComputePath function

```

while( $s_{goal}$  is not expanded)
  remove  $s$  with the smallest  $[f(s) = g(s) + h(s)]$  from  $OPEN$ ;
  insert  $s$  into  $CLOSED$ ;
  for every successor  $s'$  of  $s$  such that  $s'$  not in  $CLOSED$ 
    if  $g(s') > g(s) + c(s, s')$ 
       $g(s') = g(s) + c(s, s')$ ;
      insert  $s'$  into  $OPEN$ ;

```

for every expanded state $g(s)$ is optimal
 for every other state $g(s)$ is an upper bound
 we can now compute a least-cost path



Maxim Likhachev, University of Pennsylvania

30

A* Search

- Computes optimal g-values for relevant states

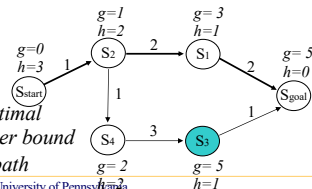
ComputePath function

```

while( $s_{goal}$  is not expanded)
  remove  $s$  with the smallest  $[f(s) = g(s) + h(s)]$  from  $OPEN$ ;
  insert  $s$  into  $CLOSED$ ;
  for every successor  $s'$  of  $s$  such that  $s'$  not in  $CLOSED$ 
    if  $g(s') > g(s) + c(s, s')$ 
       $g(s') = g(s) + c(s, s')$ ;
      insert  $s'$  into  $OPEN$ ;

```

for every expanded state $g(s)$ is optimal
 for every other state $g(s)$ is an upper bound
 we can now compute a least-cost path

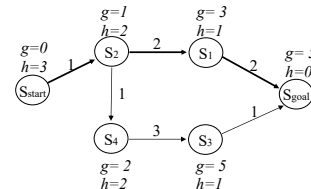


Maxim Likhachev, University of Pennsylvania

31

A* Search

- Is guaranteed to return an optimal path (in fact, for every expanded state) – optimal in terms of the solution
- Performs provably minimal number of state expansions required to guarantee optimality – optimal in terms of the computations

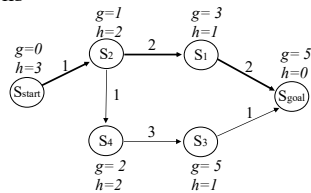


CSE-571: Courtesy of Maxim Likhachev, CMU

32

A* Search

- Is guaranteed to return an optimal path (in fact, for every expanded state) – *helps with robot deviating off its path: on if we search with A* backwards (from goal to start)*
- Performs provably minimal number of state expansions required to guarantee optimality – optimal in terms of the computations

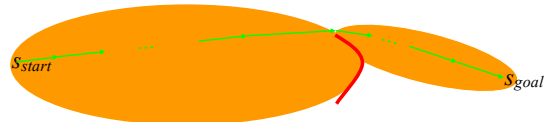


CSE-571: Courtesy of Maxim Likhachev, CMU

33

Effect of the Heuristic Function

- A* Search: expands states in the order of $f = g + h$ values



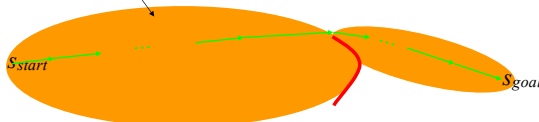
CSE-571: Courtesy of Maxim Likhachev, CMU

34

Effect of the Heuristic Function

- A* Search: expands states in the order of $f = g + h$ values

for large problems this results in A quickly running out of memory (memory: $O(n)$)*



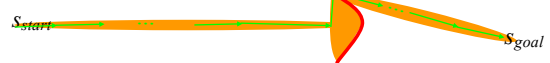
CSE-571: Courtesy of Maxim Likhachev, CMU

35

Effect of the Heuristic Function

- Weighted A* Search: expands states in the order of $f = g + \epsilon h$ values, $\epsilon > 1$ = bias towards states that are closer to goal

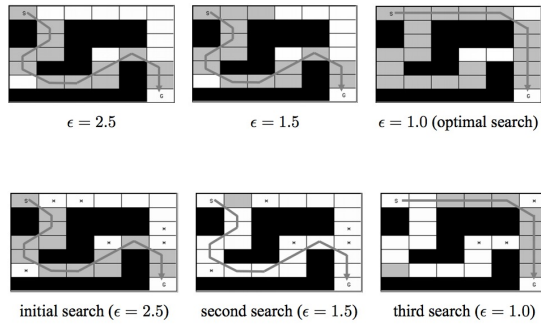
*solution is always ϵ -suboptimal:
 $\text{cost}(\text{solution}) \leq \epsilon \cdot \text{cost}(\text{optimal solution})$*



CSE-571: Courtesy of Maxim Likhachev, CMU

36

Adaptive Real-Time A*



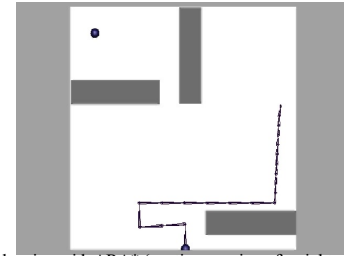
CSE-571: Courtesy of Maxim Likhachev, CMU

37

Effect of the Heuristic Function

- Weighted A* Search: expands states in the order of $f = g + \epsilon h$ values, $\epsilon > 1$ = bias towards states that are closer to goal

20DOF simulated robotic arm
state-space size: over 10^{26} states



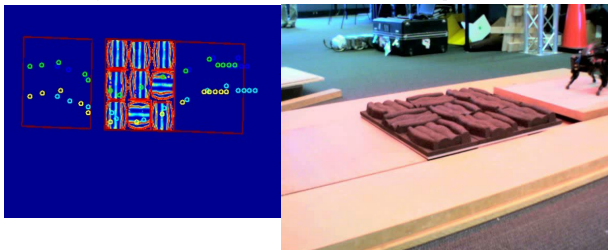
planning with ARA* (anytime version of weighted A*)

CSE-571: Courtesy of Maxim Likhachev, CMU

38

Effect of the Heuristic Function

- planning in 8D ($\langle x, y \rangle$ for each foothold)
- heuristic is Euclidean distance from the center of the body to the goal location
- cost of edges based on kinematic stability of the robot and quality of footholds



planning with R* (randomized version of weighted A*)

joint work with Subhrajit Bhattacharya, Jon Bohren, Sachin Chitta, Daniel D. Lee, Aleksandr Kushleyev, Paul Vernaza

CSE-571: Courtesy of Maxim Likhachev, CMU

39

Outline

- Deterministic planning
 - constructing a graph
 - search with A*
 - search with D*

CSE-571: Courtesy of Maxim Likhachev, CMU

40

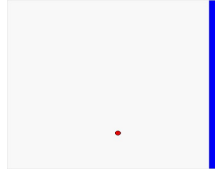
Incremental version of A* (D*/D* Lite)

- Robot needs to re-plan whenever
 - new information arrives (partially-known environments or/and dynamic environments)
 - robot deviates off its path

ATR V navigating
initially-unknown environment



planning map and path

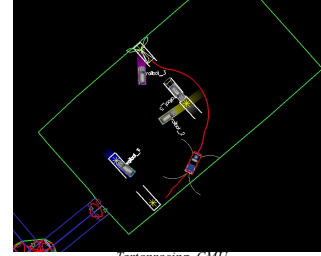


CSE-571: Courtesy of Maxim Likhachev, CMU

41

Incremental version of A* (D*/D* Lite)

- Robot needs to re-plan whenever
 - new information arrives (partially-known environments or/and dynamic environments)
 - robot deviates off its path
- incremental planning (re-planning):
reuse of previous planning efforts
planning in dynamic environments*



Tartanracing, CMU

CSE-571: Courtesy of Maxim Likhachev, CMU

42

Motivation for Incremental Version of A*

- Reuse state values from previous searches
cost of least-cost paths to s_{goal} initially

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|------|---|
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 4 | 4 | 4 | 4 | 4 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 3 | 3 | 3 | 3 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 2 | 2 | 2 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 1 | 2 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | Goal | 2 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 2 | 2 | 3 |
| 14 | 13 | 12 | 11 | 10 | 10 | 7 | 6 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 14 | 13 | 12 | 11 | 11 | 11 | 7 | 6 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 14 | 13 | 12 | 12 | 12 | 12 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 18 | 16 | 16 | 15 | 14 | 14 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |

cost of least-cost paths to s_{goal} after the door turns out to be closed

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|------|---|
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 4 | 4 | 4 | 4 | 4 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 3 | 3 | 3 | 3 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 2 | 2 | 2 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 1 | 2 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | Goal | 2 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 |
| 14 | 13 | 12 | 11 | 10 | 10 | 7 | 6 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 14 | 13 | 12 | 12 | 12 | 12 | 7 | 6 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 14 | 13 | 12 | 13 | 13 | 13 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 14 | 13 | 12 | 14 | 14 | 14 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 15 | 15 | 15 | 15 | 15 | 15 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 16 | 16 | 16 | 16 | 16 | 16 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 18 | 16 | 16 | 15 | 14 | 14 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |

CSE-571: Courtesy of Maxim Likhachev, CMU

43

Motivation for Incremental Version of A*

- Reuse state values from previous searches
cost of least-cost paths to s_{goal} initially

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|------|---|
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 4 | 4 | 4 | 4 | 4 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 3 | 3 | 3 | 3 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 2 | 2 | 2 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 1 | 2 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | Goal | 2 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 |
| 14 | 13 | 12 | 11 | 10 | 10 | 7 | 6 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 14 | 13 | 12 | 11 | 11 | 11 | 7 | 6 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 14 | 13 | 12 | 12 | 12 | 12 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 14 | 13 | 12 | 13 | 13 | 13 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 14 | 13 | 12 | 14 | 14 | 14 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 15 | 15 | 15 | 15 | 15 | 15 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 16 | 16 | 16 | 16 | 16 | 16 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 18 | 16 | 16 | 15 | 14 | 14 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |

These costs are optimal g-values if search is done backwards

cost of least-cost paths to s_{goal} after the door turns out to be closed

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|------|---|
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 4 | 4 | 4 | 4 | 4 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 3 | 3 | 3 | 3 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 2 | 2 | 2 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 1 | 2 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | Goal | 2 |
| 14 | 13 | 12 | 11 | 10 | 10 | 7 | 6 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 14 | 13 | 12 | 12 | 12 | 12 | 7 | 6 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 14 | 13 | 12 | 13 | 13 | 13 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 14 | 13 | 12 | 14 | 14 | 14 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 15 | 15 | 15 | 15 | 15 | 15 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 16 | 16 | 16 | 16 | 16 | 16 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 18 | 16 | 16 | 15 | 14 | 14 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |

CSE-571: Courtesy of Maxim Likhachev, CMU

44

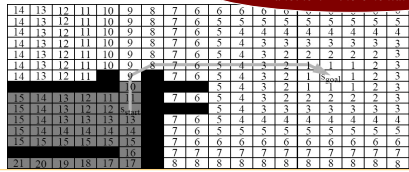
Motivation for Incremental Version of A*

- Reuse state values from previous searches
cost of least-cost paths to s_{goal} initially



These costs are optimal g-values if search is done backwards

cost of least-cost paths to s_{goal} How to reuse these g-values from one search to another? - incremental A*

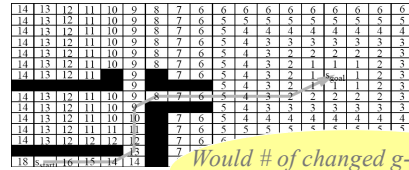


CSE-571: Courtesy of Maxim Likhachev, CMU

45

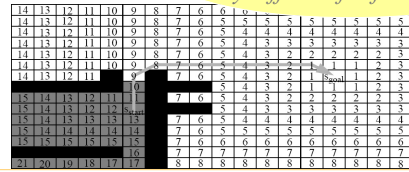
Motivation for Incremental Version of A*

- Reuse state values from previous searches
cost of least-cost paths to s_{goal} initially



Would # of changed g-values be very different for forward A*?

cost of least-cost paths to s_{start}

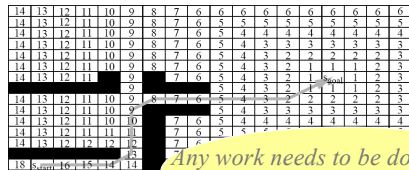


CSE-571: Courtesy of Maxim Likhachev, CMU

46

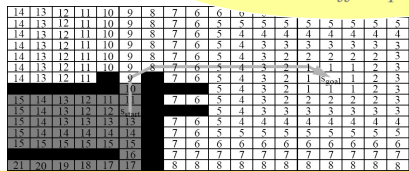
Motivation for Incremental Version of A*

- Reuse state values from previous searches
cost of least-cost paths to s_{goal} initially



Any work needs to be done if robot deviates off its path?

cost of least-cost paths to s_{start}



CSE-571: Courtesy of Maxim Likhachev, CMU

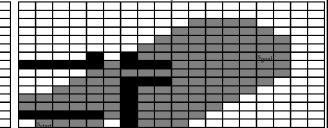
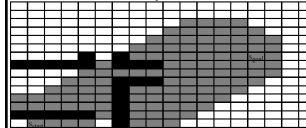
47

Incremental Version of A*

- Reuse state values from previous searches

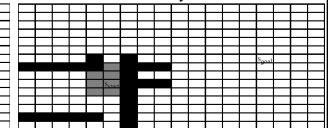
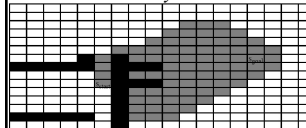
initial search by backwards A*

initial search by D* Lite



second search by backwards A*

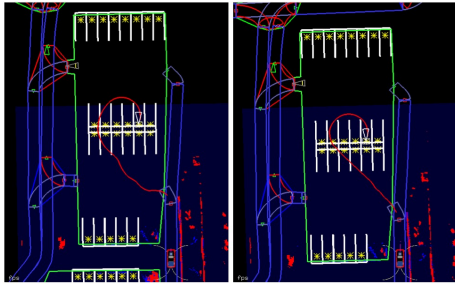
second search by D* Lite



CSE-571: Courtesy of Maxim Likhachev, CMU

48

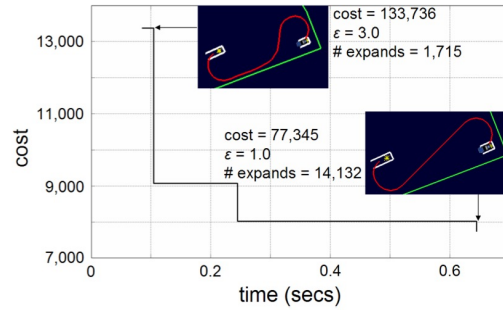
Anytime Aspects



CSE-571: Courtesy of Maxim Likhachev, CMU

49

Anytime Aspects

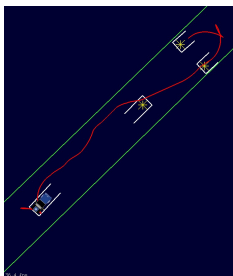


CSE-571: Courtesy of Maxim Likhachev, CMU

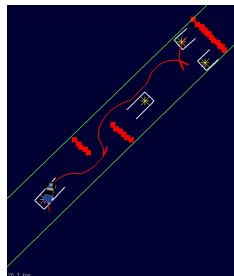
50

Searching the Graph

- Incremental behavior of Anytime D*:



initial path



a path after re-planning

Maxim Likhachev & Dave Ferguson

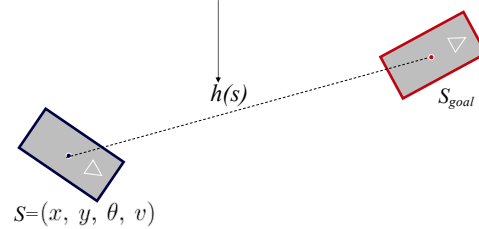
51

51

Searching the Graph

- Performance of Anytime D* depends strongly on heuristics $h(s)$: estimates of cost-to-goal

should be consistent and admissible (never overestimate cost-to-goal)



Maxim Likhachev & Dave Ferguson

53

53

Searching the Graph

- In our planner: $h(s) = \max(h_{\text{mech}}(s), h_{\text{env}}(s))$, where
 - $h_{\text{mech}}(s)$ – mechanism-constrained heuristic
 - $h_{\text{env}}(s)$ – environment-constrained heuristic

$h_{\text{mech}}(s)$ – considers only dynamics constraints and ignores environment

$h_{\text{env}}(s)$ – considers only environment constraints and ignores dynamics

Maxim Likhachev & Dave Ferguson

54

Searching the Graph

- In our planner: $h(s) = \max(h_{\text{mech}}(s), h_{\text{env}}(s))$, where
 - $h_{\text{mech}}(s)$ – mechanism-constrained heuristic
 - $h_{\text{env}}(s)$ – environment-constrained heuristic

$h_{\text{mech}}(s)$ – considers only dynamics constraints and ignores environment

$h_{\text{env}}(s)$ – considers only environment constraints and ignores dynamics

pre-computed as a table lookup for high-res. lattice

computed online by running a 2D A* with late termination

Maxim Likhachev & Dave Ferguson

55

Heuristics

| heuristic | states expanded | time (secs) |
|-----------|-----------------|-------------|
| h | 2,019 | 0.06 |
| h_{2D} | 26,108 | 1.30 |
| h_{fsh} | 124,794 | 3.49 |

CSE-571: Courtesy of Maxim Likhachev, CMU

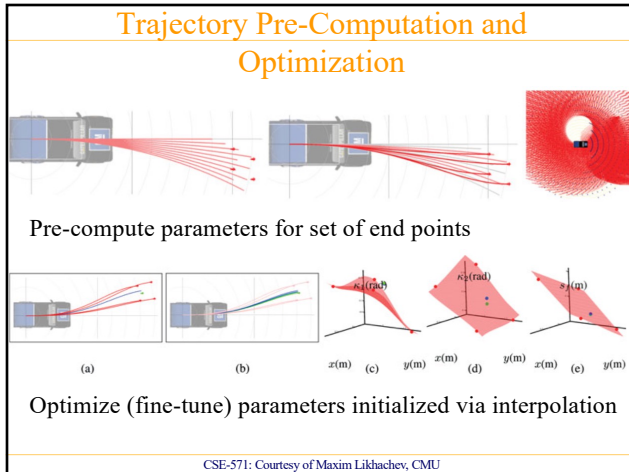
56

Example, again

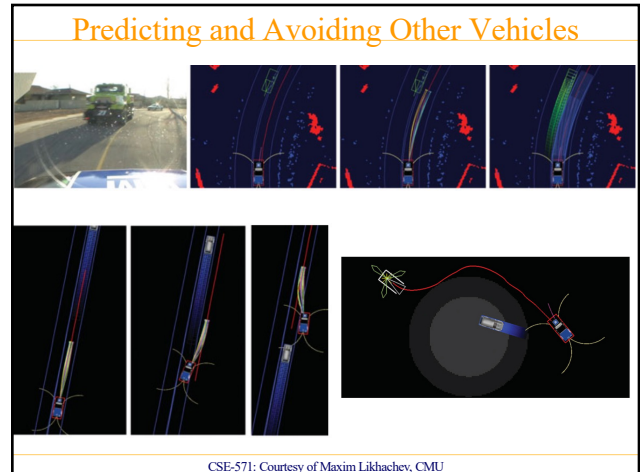
Urban Challenge Race, CMU team, planning with Anytime D*

CSE-571: Courtesy of Maxim Likhachev, CMU

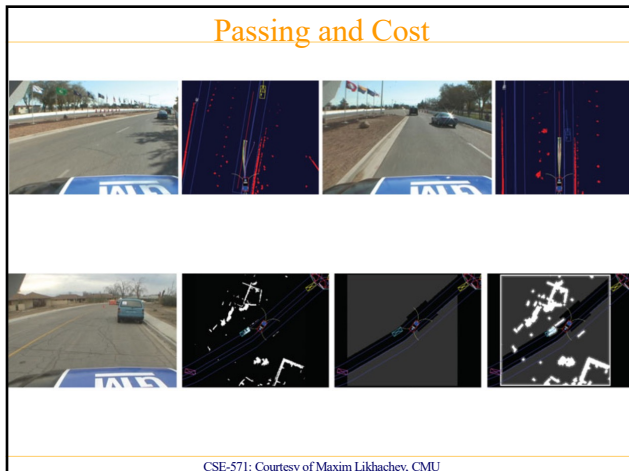
57



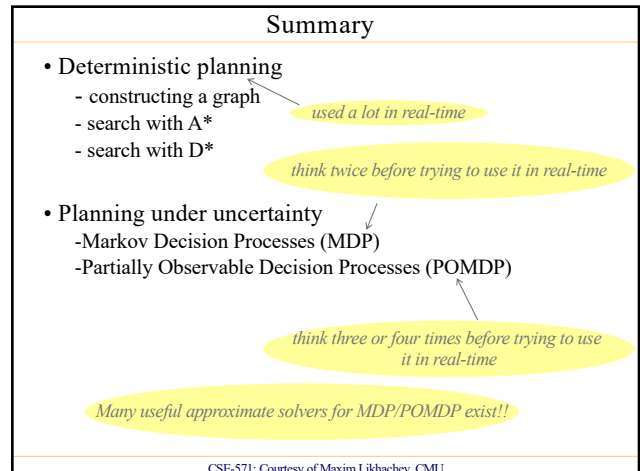
58



59

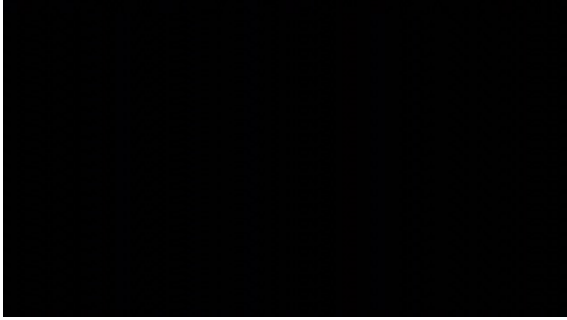


60



63

Manipulation Planning Examples



64