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)

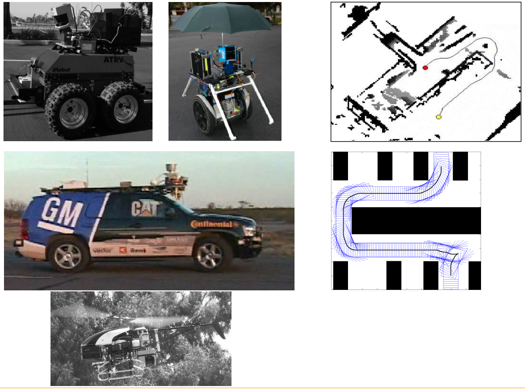
- Two types of constraints:
  environmental constraints (e.g., obstacles)
  dynamics/kinematics constraints of the robot

- Generated motion/path should (objective):
  be any feasible path
  minimize cost such as distance, time, energy, risk, …

2

Motion/Path Planning

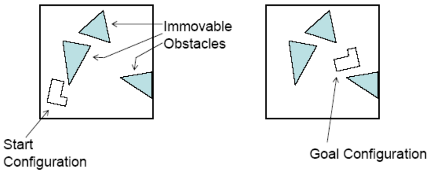
Examples (of what is usually referred to as path planning):



3

Motion/Path Planning

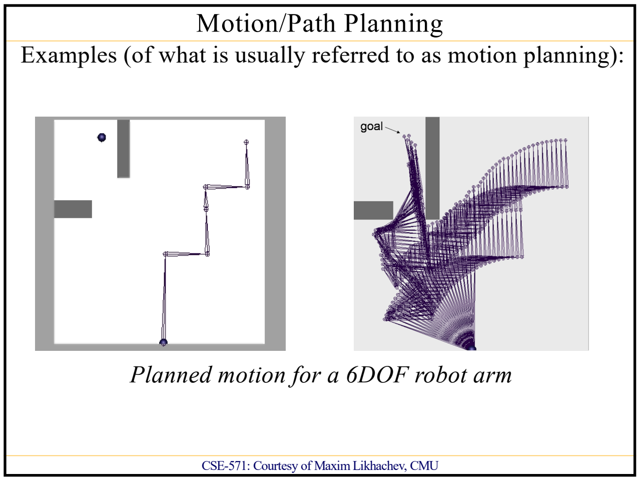
Examples (of what is usually referred to as motion planning):



Piano Movers' problem

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

4



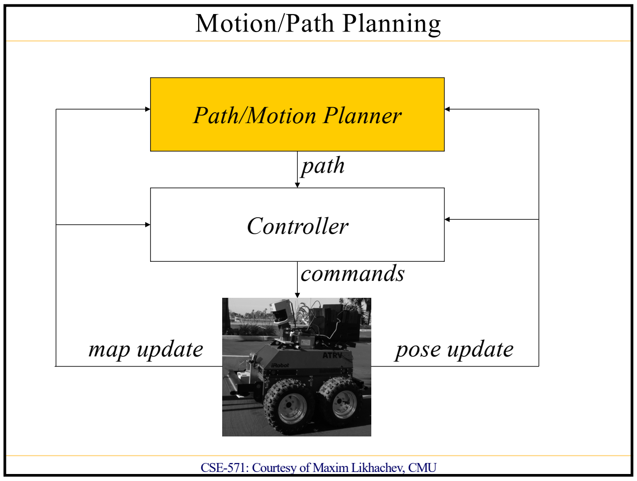
Motion/Path Planning

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



Motion/Path Planning

Path/Motion Planner

path

Controller

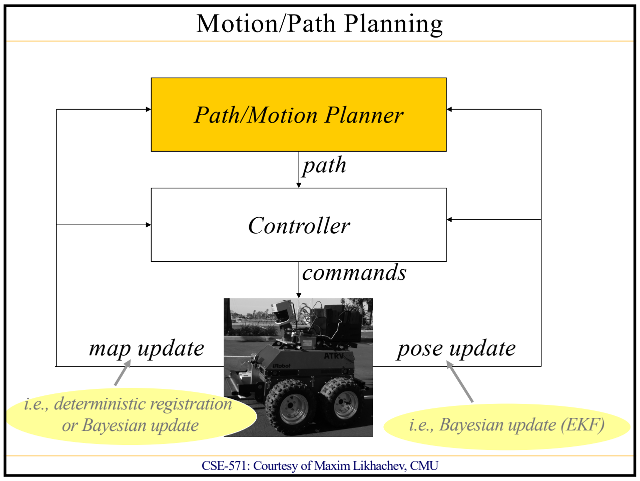
commands

map update

pose update

CSE-571: Courtesy of Maxim Likhachev, CMU

6



Motion/Path Planning

Path/Motion Planner

path

Controller

commands

map update

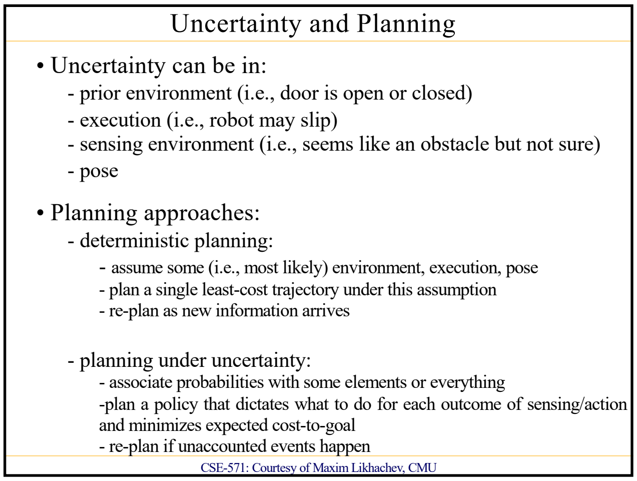
pose update

i.e., deterministic registration or Bayesian update

i.e., Bayesian update (EKF)

CSE-571: Courtesy of Maxim Likhachev, CMU

7

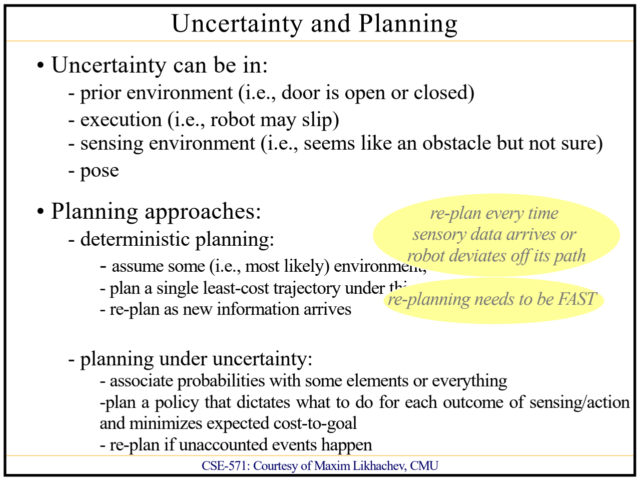


Uncertainty and Planning

- 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

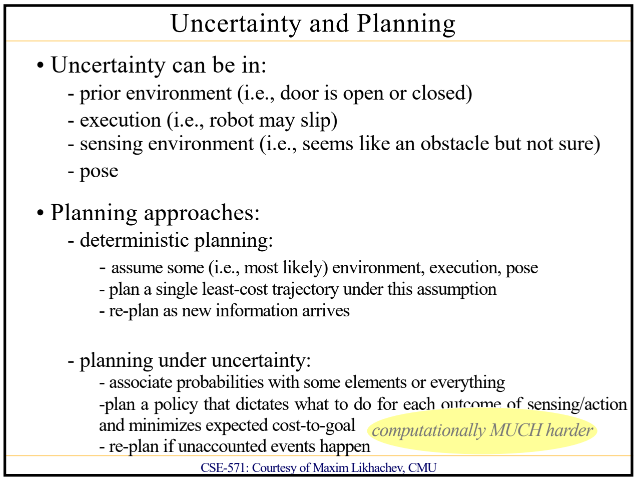
- Uncertainty can be in:
  - prior environment (i.e., door is open or closed)
  - execution (i.e., robot may slip)
  - sensing environment (i.e., seems like an obstacle but not sure)
  - pose
- Planning approaches:
  - deterministic planning:
    - assume some (i.e., most likely) environment,
    - plan a single least-cost trajectory under th
    - 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



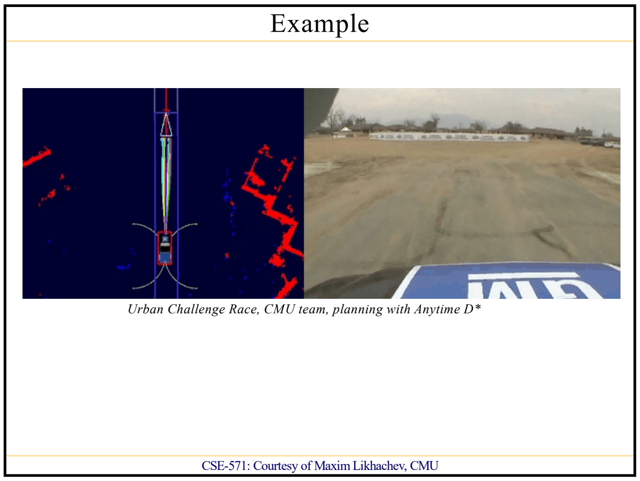
Uncertainty and Planning

- 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

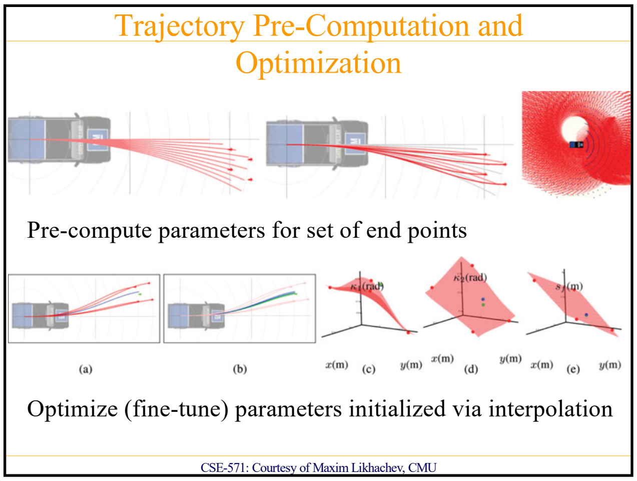


Example

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

CSE-571: Courtesy of Maxim Likhachev, CMU

11



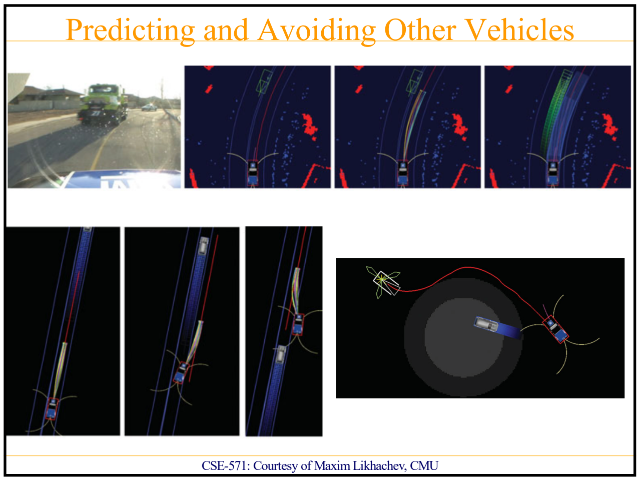
Trajectory Pre-Computation and Optimization

Pre-compute parameters for set of end points

Optimize (fine-tune) parameters initialized via interpolation

CSE-571: Courtesy of Maxim Likhachev, CMU

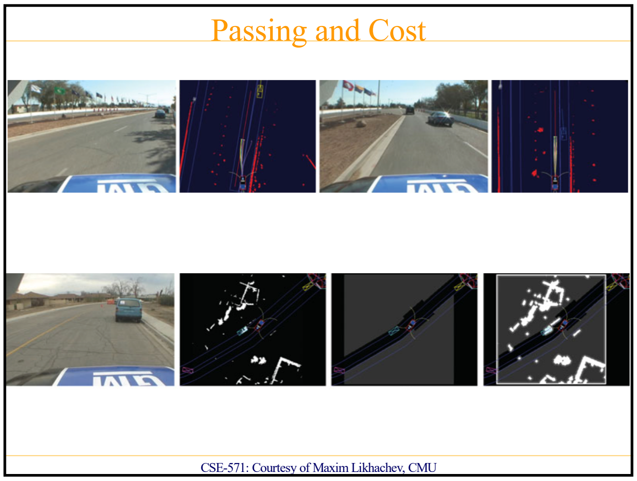
12



Predicting and Avoiding Other Vehicles

CSE-571: Courtesy of Maxim Likhachev, CMU

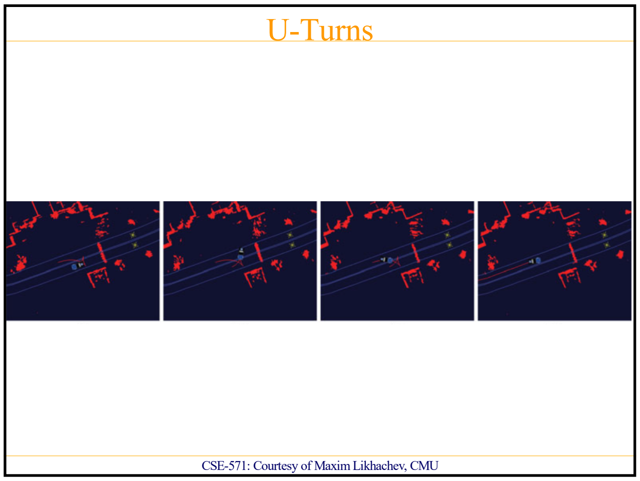
13



Passing and Cost

CSE-571: Courtesy of Maxim Likhachev, CMU

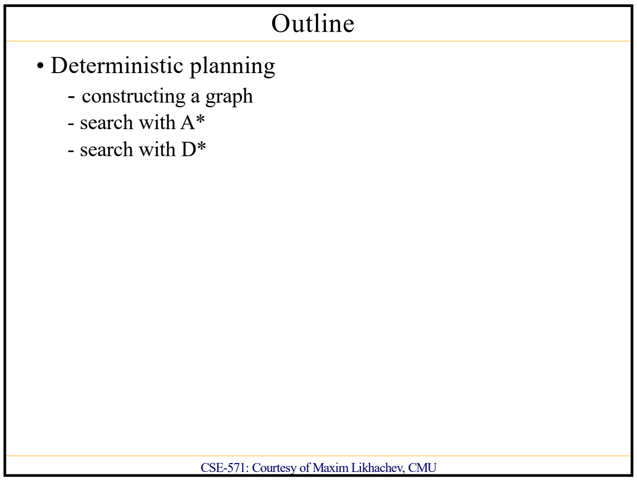
14



U-Turns

CSE-571: Courtesy of Maxim Likhachev, CMU

15

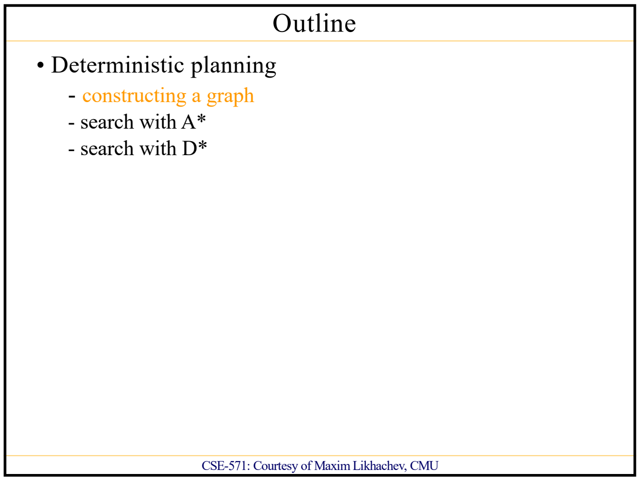


Outline

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

CSE-571: Courtesy of Maxim Likhachev, CMU

16

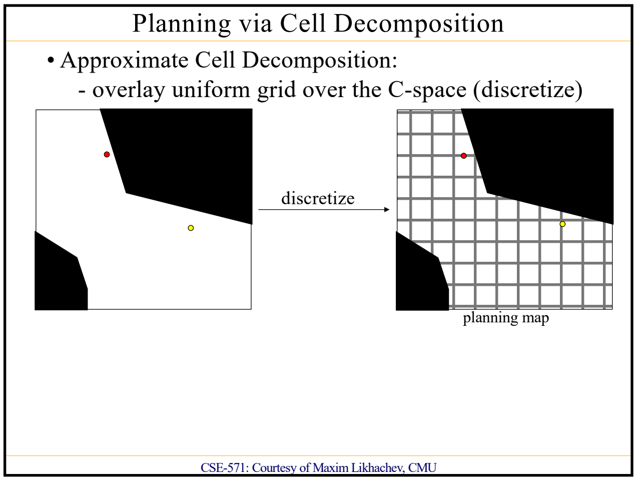


Outline

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

CSE-571: Courtesy of Maxim Likhachev, CMU

17

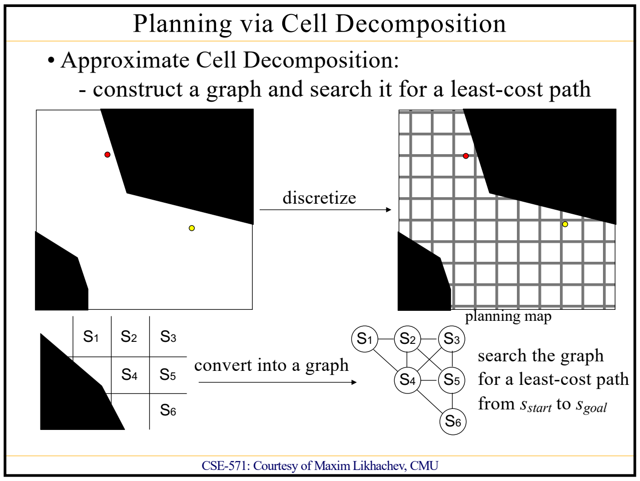


Planning via Cell Decomposition

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

CSE-571: Courtesy of Maxim Likhachev, CMU

18



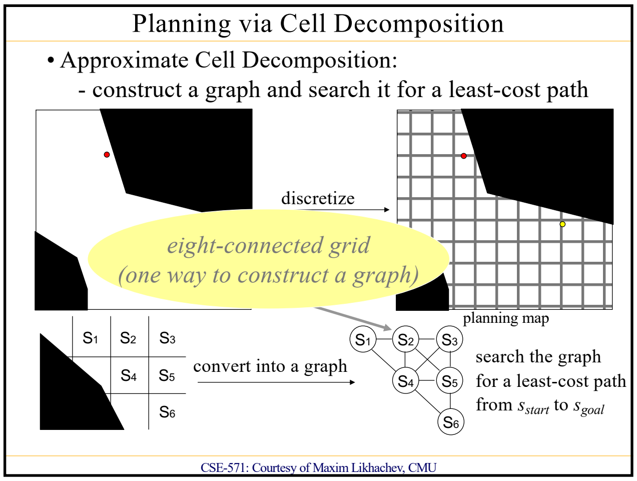
Planning via Cell Decomposition

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

search the graph for a least-cost path from s_{start} to s_{goal}

CSE-571: Courtesy of Maxim Likhachev, CMU

19



Planning via Cell Decomposition

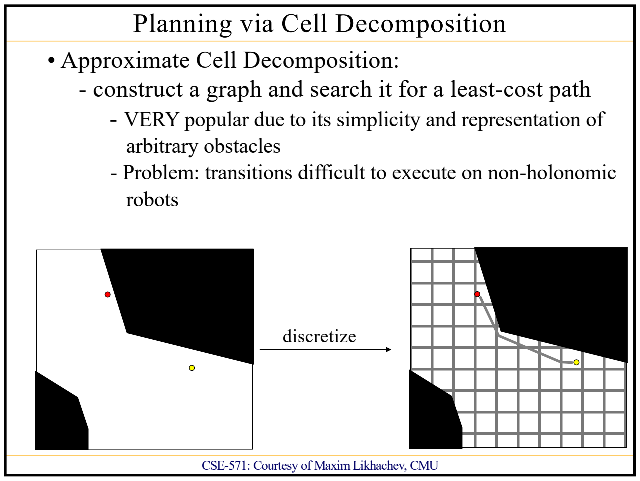
- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

eight-connected grid (one way to construct a graph)

search the graph for a least-cost path from s_{start} to s_{goal}

CSE-571: Courtesy of Maxim Likhachev, CMU

20

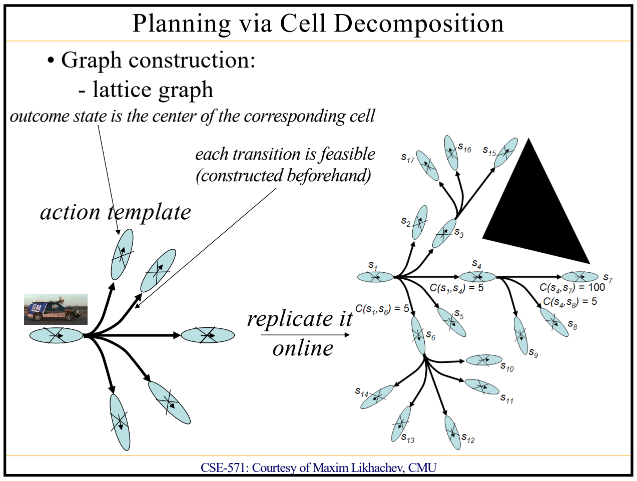


Planning via Cell Decomposition

- 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

21



Planning via Cell Decomposition

- Graph construction:
  - lattice graph

outcome state is the center of the corresponding cell

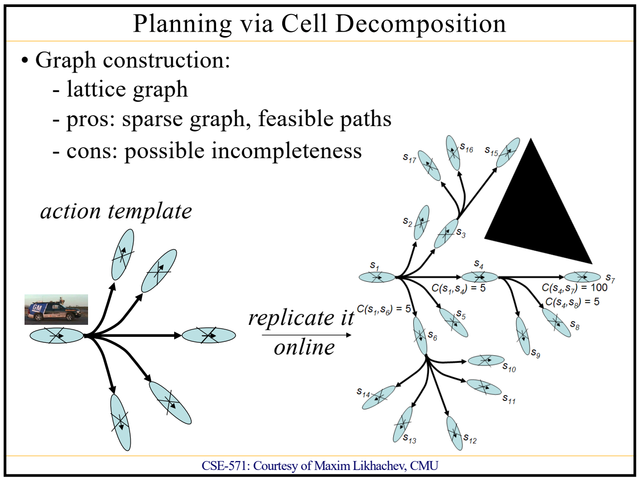
each transition is feasible (constructed beforehand)

action template

replicate it online

CSE-571: Courtesy of Maxim Likhachev, CMU

22



Planning via Cell Decomposition

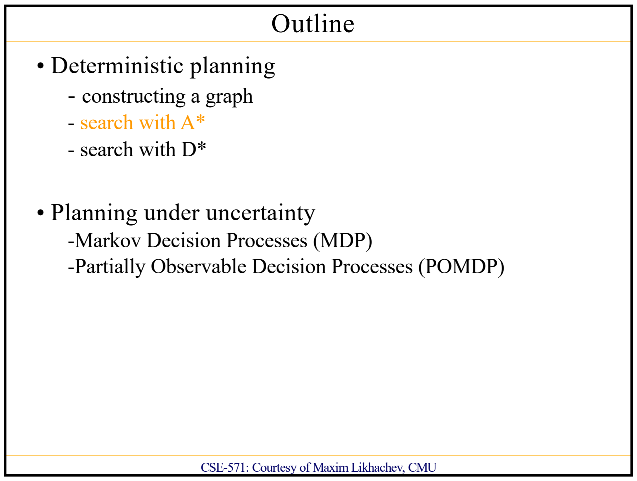
- Graph construction:
  - lattice graph
  - pros: sparse graph, feasible paths
  - cons: possible incompleteness

action template

replicate it online

CSE-571: Courtesy of Maxim Likhachev, CMU

23

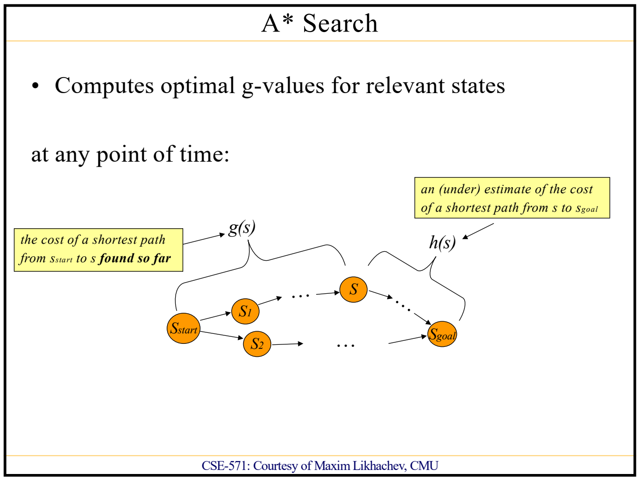


Outline

- 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

24



A* Search

- Computes optimal g-values for relevant states

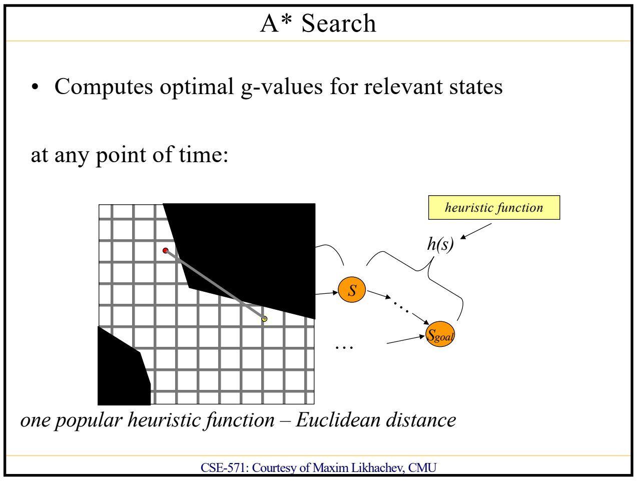
at any point of time:

the cost of a shortest path from s_{start} to s ***found so far***

an (under) estimate of the cost of a shortest path from s to s_{goal}

CSE-571: Courtesy of Maxim Likhachev, CMU

25



A* Search

- Computes optimal g-values for relevant states

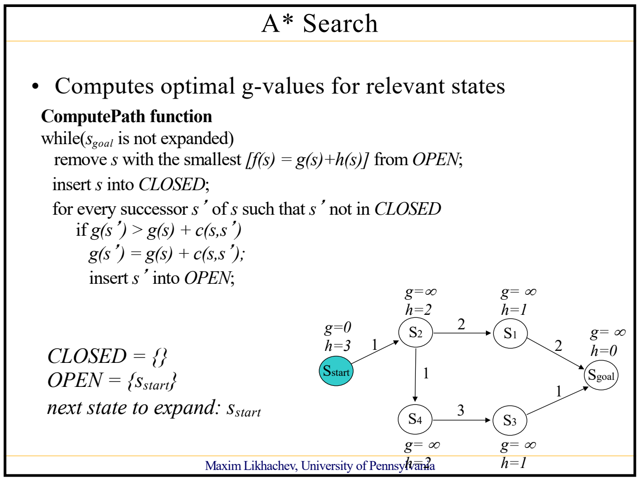
at any point of time:

heuristic function

one popular heuristic function – Euclidean distance

CSE-571: Courtesy of Maxim Likhachev, CMU

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 $OPEN$;
```

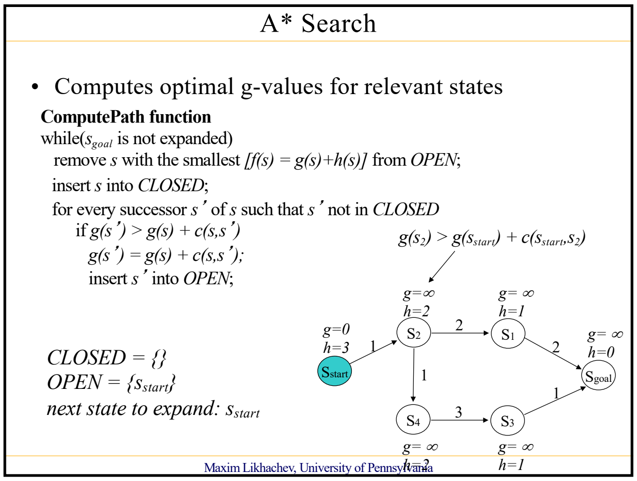
CLOSED = {}

OPEN = {s_{start}}

next state to expand: s_{start}

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 $OPEN$;
```

$g(s_2) > g(s_{start}) + c(s_{start},s_2)$

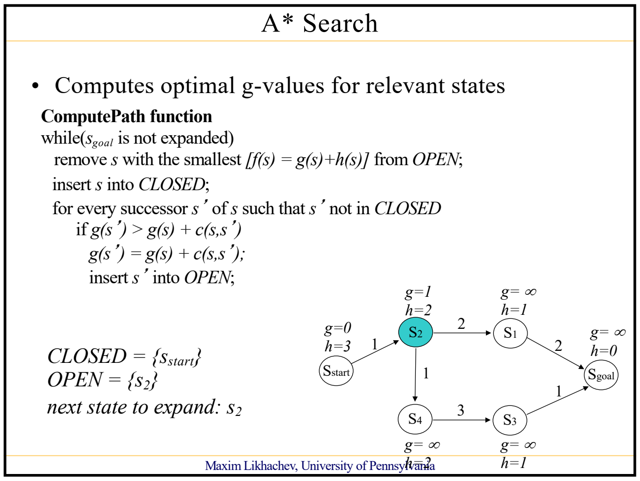
CLOSED = {}

OPEN = {s_{start}}

next state to expand: s_{start}

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 = $\{s_{start}\}$

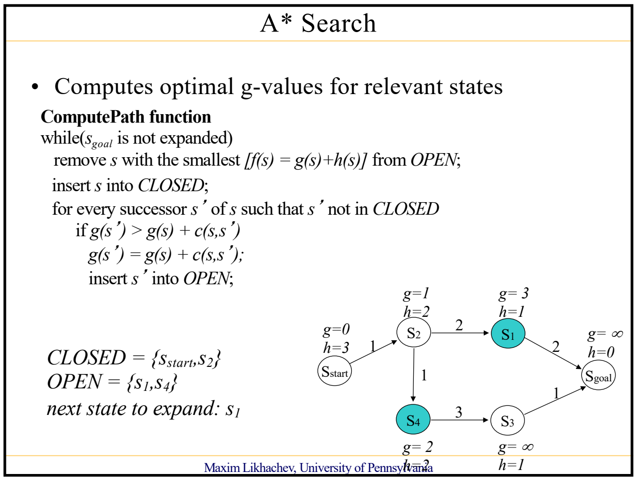
OPEN = $\{s_2\}$

next state to expand: s_2

g=0
h=3
Sstart
1
g=1
h=2
S2
2
g= ∞
h=1
S1
2
g= ∞
h=0
Sgoal
1
S4
3
S3
1
g= ∞
h=2
g= ∞
h=1

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 $OPEN$;
```

CLOSED = $\{s_{start},s_2\}$

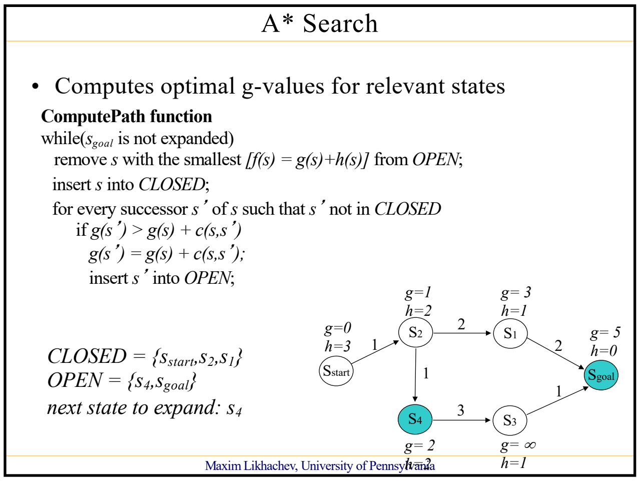
OPEN = $\{s_1,s_4\}$

next state to expand: s_1

g=0
h=3
Sstart
1
g=1
h=2
S2
2
g= 3
h=1
S1
2
g= ∞
h=0
Sgoal
1
S4
3
S3
1
g= 2
h=2
g= ∞
h=1

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 $OPEN$;
```

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

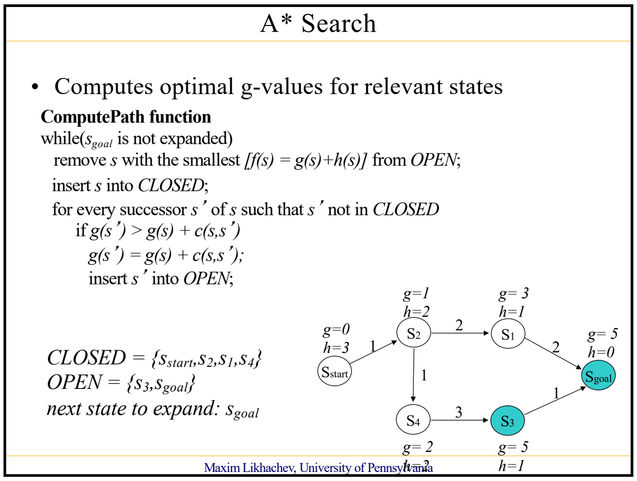
OPEN = $\{s_4,s_{goal}\}$

next state to expand: s_4

g=0
h=3
Sstart
1
g=1
h=2
S2
2
g= 3
h=1
S1
2
g= 5
h=0
Sgoal
1
S4
3
S3
1
g= 2
h=2
g= ∞
h=1

Maxim Likhachev, University of Pennsylvania

31



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

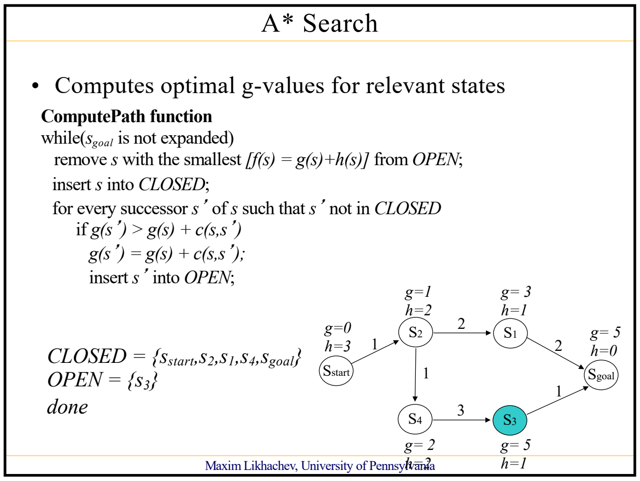
OPEN = $\{s_3,s_{goal}\}$

next state to expand: s_{goal}

g=0
h=3
Sstart
1
g=1
h=2
S2
2
g= 3
h=1
S1
2
g= 5
h=0
Sgoal
1
S4
3
S3
1
g= 2
h=2
g= 5
h=1

Maxim Likhachev, University of Pennsylvania

32



A* Search

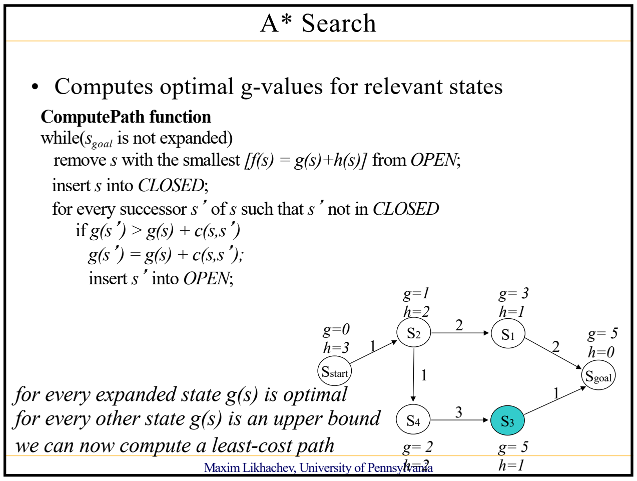
- Computes optimal g-values for relevant states

```
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

33



A* Search

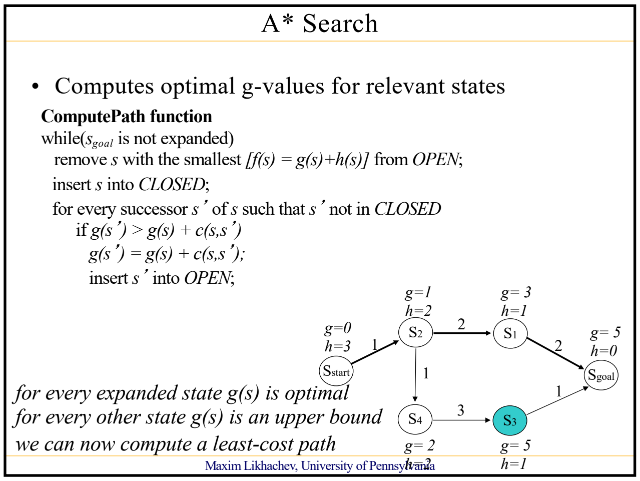
- Computes optimal g-values for relevant states

```
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

34



A* Search

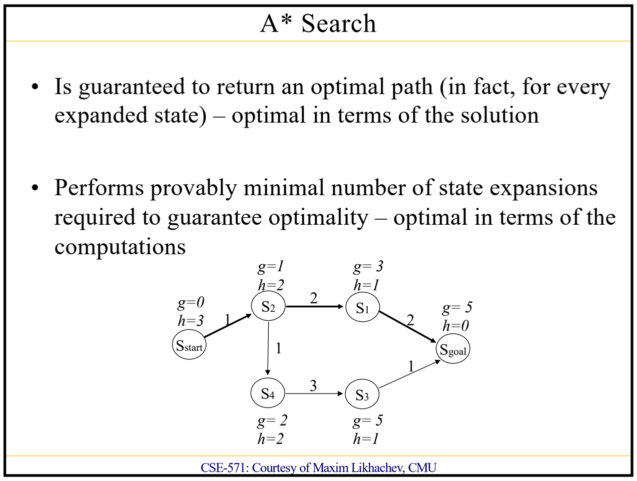
- Computes optimal g-values for relevant states

```
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

35

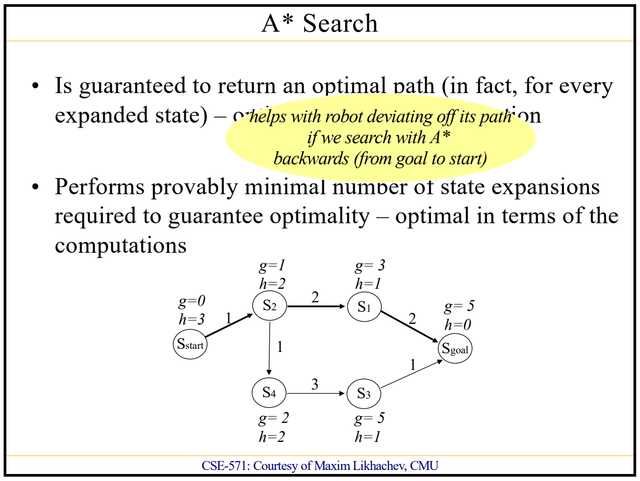


A* Search

- 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

36



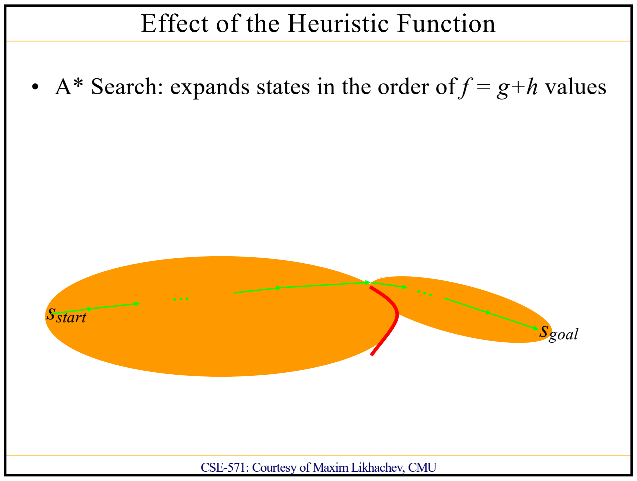
A* Search

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

helps with robot deviating off its path if we search with A backwards (from goal to start)*

CSE-571: Courtesy of Maxim Likhachev, CMU

37

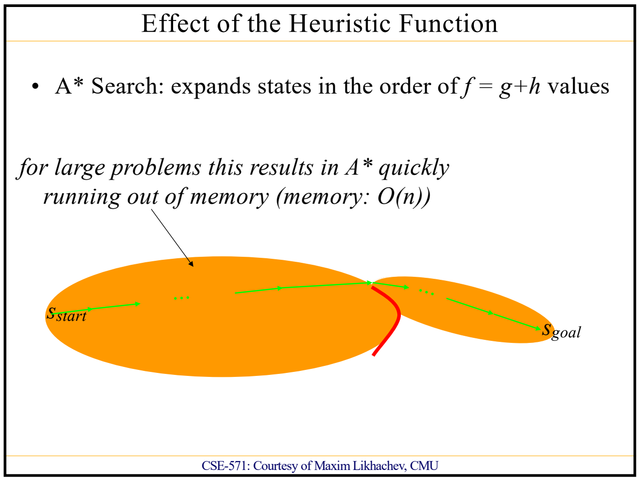


Effect of the Heuristic Function

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

CSE-571: Courtesy of Maxim Likhachev, CMU

38



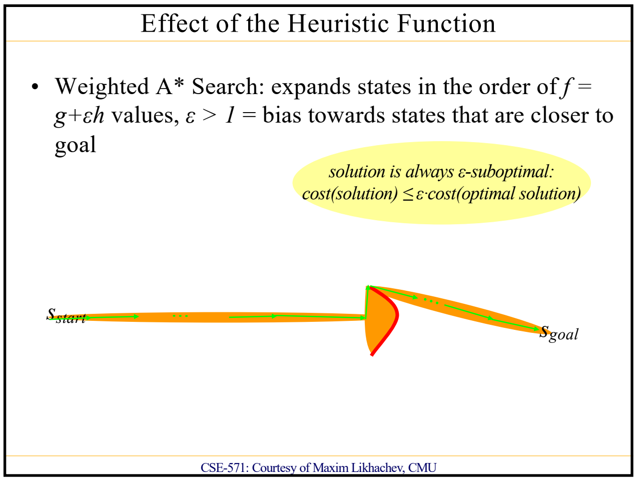
Effect of the Heuristic Function

- 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

39



Effect of the Heuristic Function

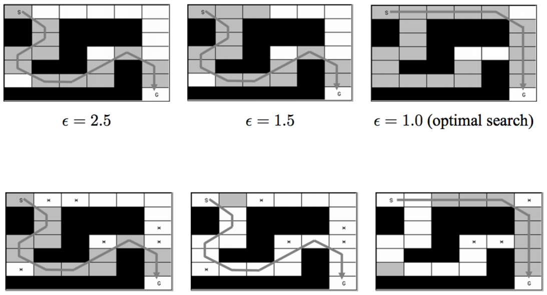
- Weighted A* Search: expands states in the order of $f = g+\varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal

solution is always ε-suboptimal: cost(solution) ≤ ε·cost(optimal solution)

CSE-571: Courtesy of Maxim Likhachev, CMU

40

Adaptive Real-Time A*



$\epsilon = 2.5$ $\epsilon = 1.5$ $\epsilon = 1.0$ (optimal search)

initial search ($\epsilon = 2.5$) second search ($\epsilon = 1.5$) third search ($\epsilon = 1.0$)

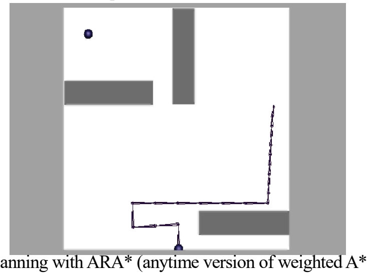
CSE-571: Courtesy of Maxim Likhachev, CMU

41

Effect of the Heuristic Function

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

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



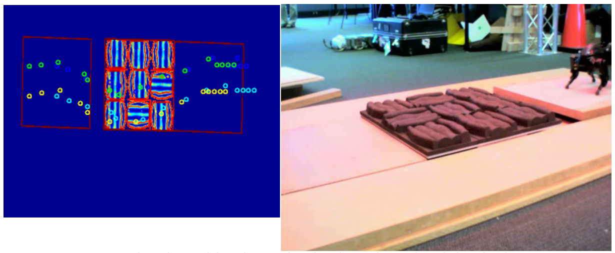
planning with ARA* (anytime version of weighted A*)

CSE-571: Courtesy of Maxim Likhachev, CMU

42

Effect of the Heuristic Function

- planning in 8D (<*x,y*> 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

43

Outline

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

CSE-571: Courtesy of Maxim Likhachev, CMU

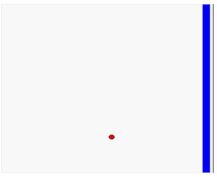
44

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

ATRV navigating initially-unknown environment

planning map and path



CSE-571: Courtesy of Maxim Likhachev, CMU

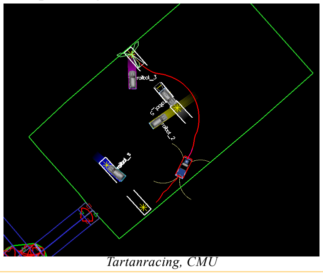
45

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

46

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	6	6
14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5	5	5
14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3	3	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11		9		7	6	5	4	3	2	1	s_{goal}	1	2	3
					9				5	4	3	2	1	1	1	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9				5	4	3	3	3	3	3	3	3
14	13	12	11	10	10		7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	11	11		7	6	5	5	5	5	5	5	5	5	5
14	13	12	12	12	12		7	6	6	6	6	6	6	6	6	6	6
					13		7	7	7	7	7	7	7	7	7	7	7
18	s_{start}	16	15	14	14		8	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	6	6
14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5	5	5
14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3	3	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11		9		7	6	5	4	3	2	1	s_{goal}	1	2	3
					10				5	4	3	2	1	1	1	2	3
15	14	13	12	11	11		7	6	5	4	3	2	2	2	2	2	3
15	14	13	12	12	s_{start}				5	4	3	3	3	3	3	3	3
15	14	13	13	13	13		7	6	5	4	4	4	4	4	4	4	4
15	14	14	14	14	14		7	6	5	5	5	5	5	5	5	5	5
15	15	15	15	15	15		7	6	6	6	6	6	6	6	6	6	6
					16		7	7	7	7	7	7	7	7	7	7	7
21	20	19	18	17	17		8	8	8	8	8	8	8	8	8	8	8

CSE-571: Courtesy of Maxim Likhachev, CMU

47

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	6	6
14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5	5	5
14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3	3	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11		9		7	6	5	4	3	2	1	s_{goal}	1	2	3
					9				5	4	3	2	1	1	1	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9				5	4	3	3	3	3	3	3	3
14	13	12	11	10	10		7	6	5	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
14	13	12	11	11	11		7	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
14	13	12	12	12	12		7	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
					13		7	7	7	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
18	s_{start}	16	15	14	14		8	8	8	8	8	8	8	8	8	8	8

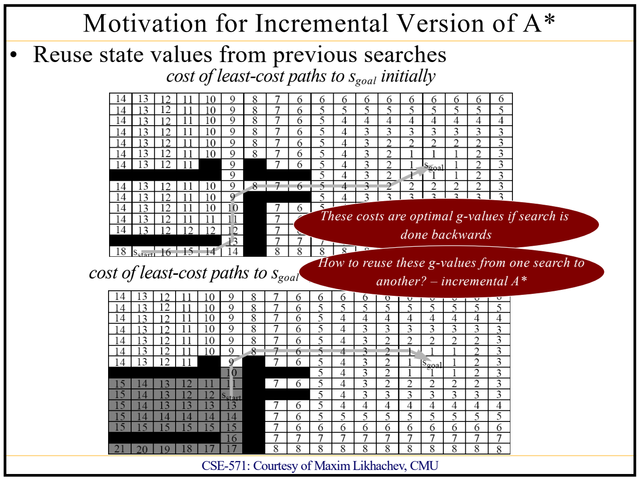
These costs are optimal g-values if search is done backwards

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	6	6
14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5	5	5
14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3	3	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11		9		7	6	5	4	3	2	1	s_{goal}	1	2	3
					10				5	4	3	2	1	1	1	2	3
15	14	13	12	11	11		7	6	5	4	3	2	2	2	2	2	3
15	14	13	12	12	s_{start}				5	4	3	3	3	3	3	3	3
15	14	13	13	13	13		7	6	5	4	4	4	4	4	4	4	4
15	14	14	14	14	14		7	6	5	5	5	5	5	5	5	5	5
15	15	15	15	15	15		7	6	6	6	6	6	6	6	6	6	6
					16		7	7	7	7	7	7	7	7	7	7	7
21	20	19	18	17	17		8	8	8	8	8	8	8	8	8	8	8

CSE-571: Courtesy of Maxim Likhachev, CMU

48



Motivation for Incremental Version of A*

- Reuse state values from previous searches

cost of least-cost paths to s_{goal} initially

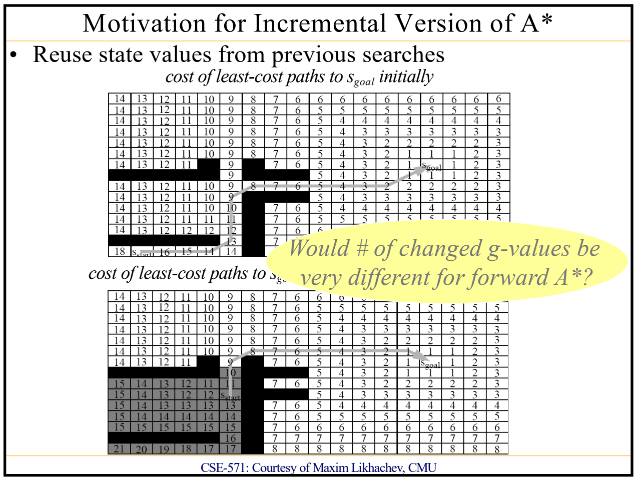
cost of least-cost paths to s_{goal} after the door turns out to be closed

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

*How to reuse these g-values from one search to another? – incremental A**

CSE-571: Courtesy of Maxim Likhachev, CMU

49



Motivation for Incremental Version of A*

- Reuse state values from previous searches

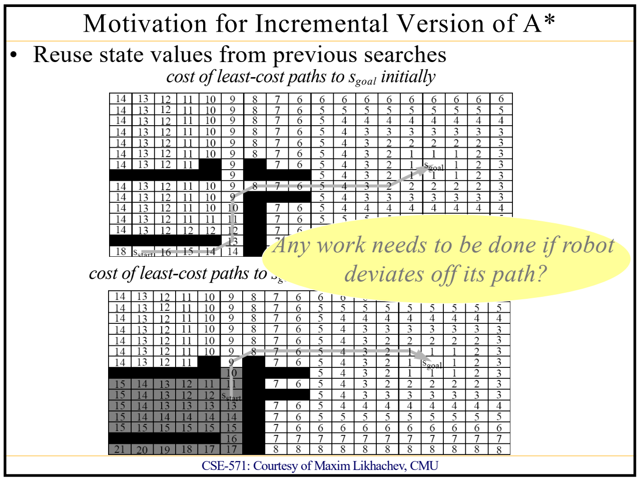
cost of least-cost paths to s_{goal} initially

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

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

CSE-571: Courtesy of Maxim Likhachev, CMU

50



Motivation for Incremental Version of A*

- Reuse state values from previous searches

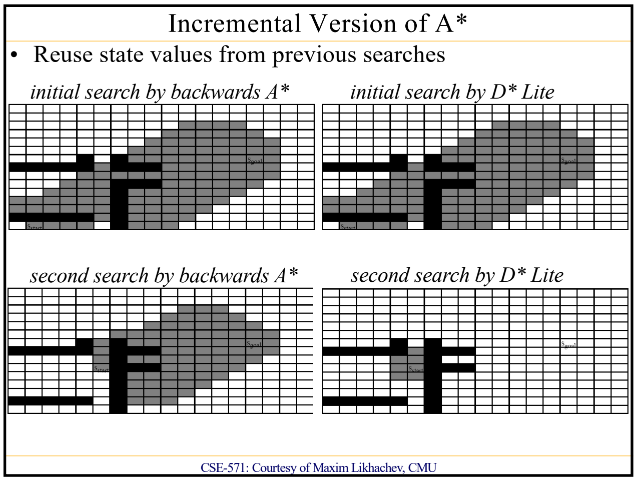
cost of least-cost paths to s_{goal} initially

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

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

CSE-571: Courtesy of Maxim Likhachev, CMU

51



Incremental Version of A*

- Reuse state values from previous searches

*initial search by backwards A**

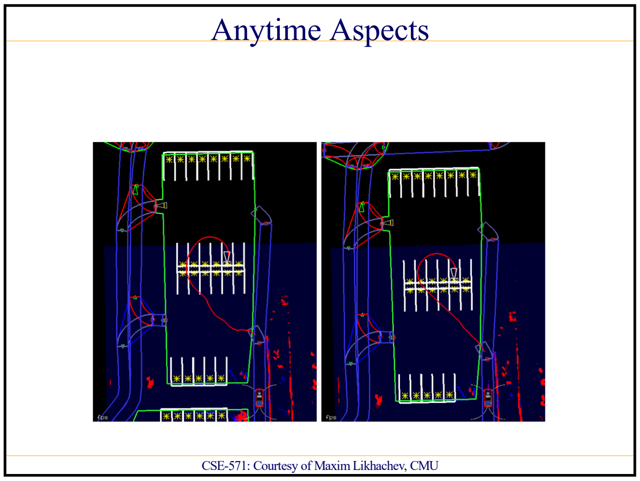
initial search by D Lite*

*second search by backwards A**

second search by D Lite*

CSE-571: Courtesy of Maxim Likhachev, CMU

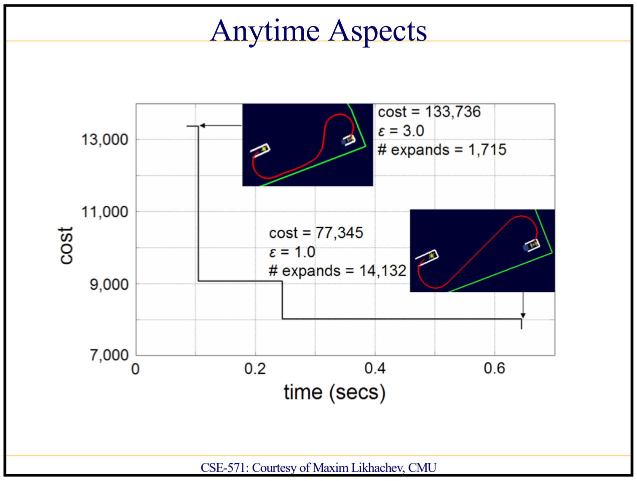
52



Anytime Aspects

CSE-571: Courtesy of Maxim Likhachev, CMU

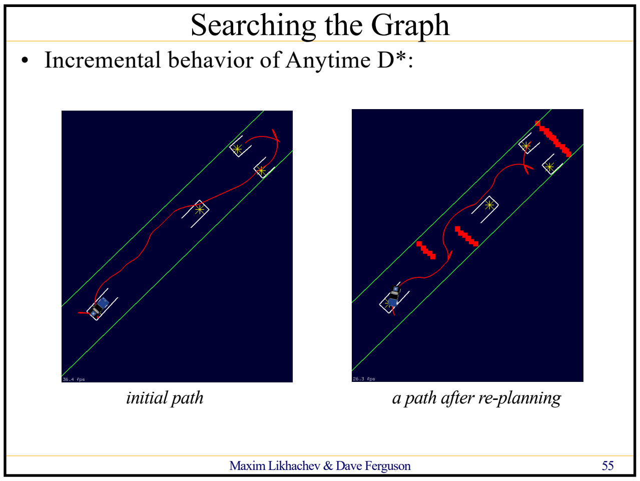
53



Anytime Aspects

CSE-571: Courtesy of Maxim Likhachev, CMU

54



Searching the Graph

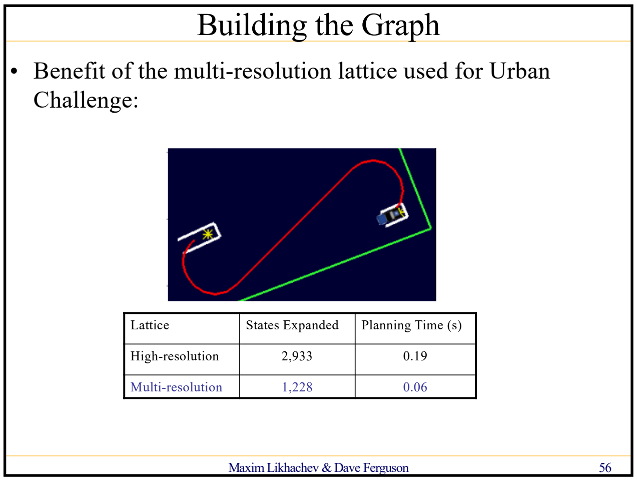
- Incremental behavior of Anytime D*:

initial path

a path after re-planning

Maxim Likhachev & Dave Ferguson 55

55



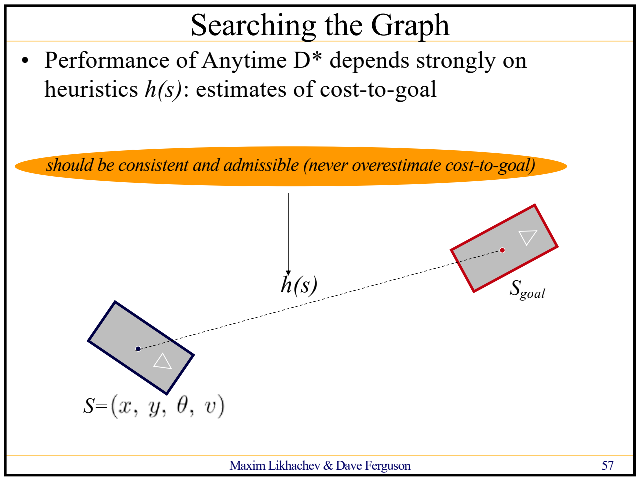
Building the Graph

- Benefit of the multi-resolution lattice used for Urban Challenge:

Lattice	States Expanded	Planning Time (s)
High-resolution	2,933	0.19
Multi-resolution	1,228	0.06

Maxim Likhachev & Dave Ferguson 56

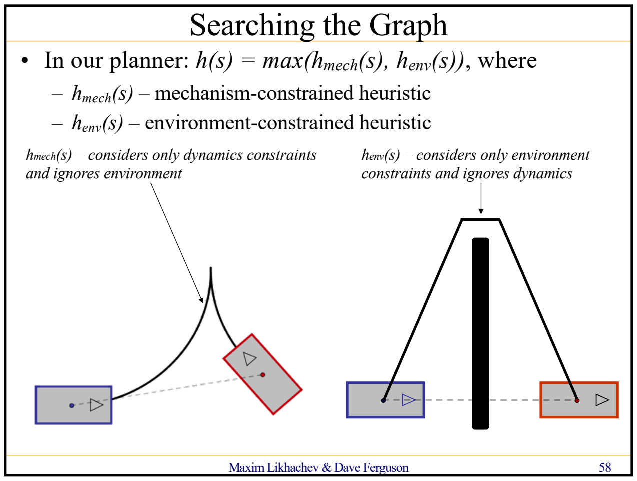
56



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)

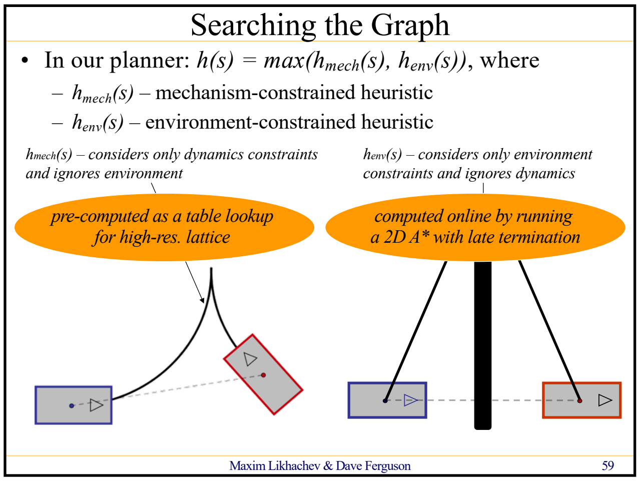


Searching the Graph

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

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

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



Searching the Graph

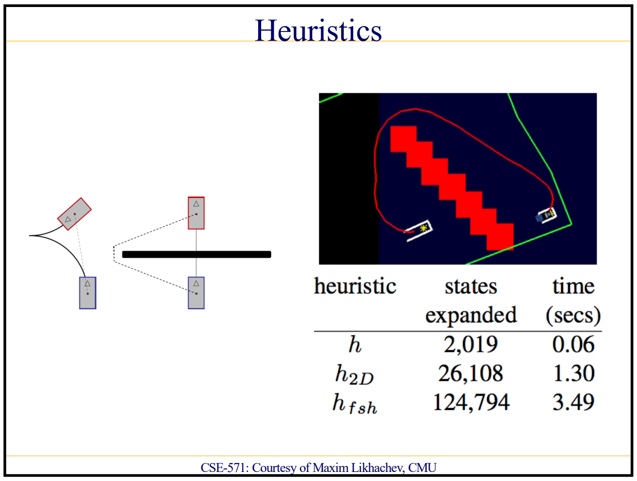
- In our planner: $h(s) = max(h_{mech}(s), h_{env}(s))$, where
  - $h_{mech}(s)$ – mechanism-constrained heuristic
  - $h_{env}(s)$ – environment-constrained heuristic

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

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

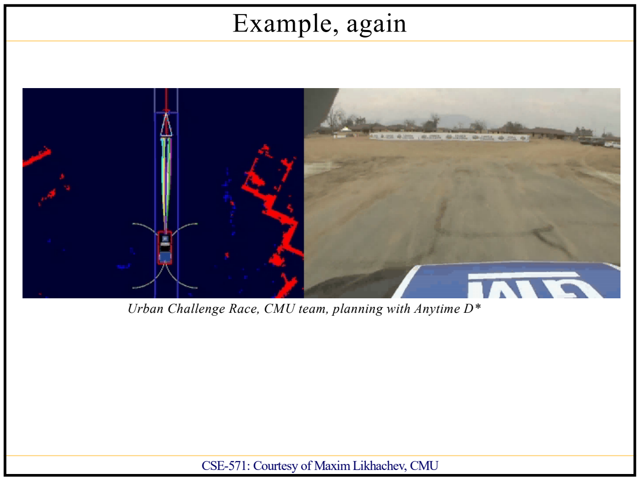
pre-computed as a table lookup for high-res. lattice

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



Heuristics

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

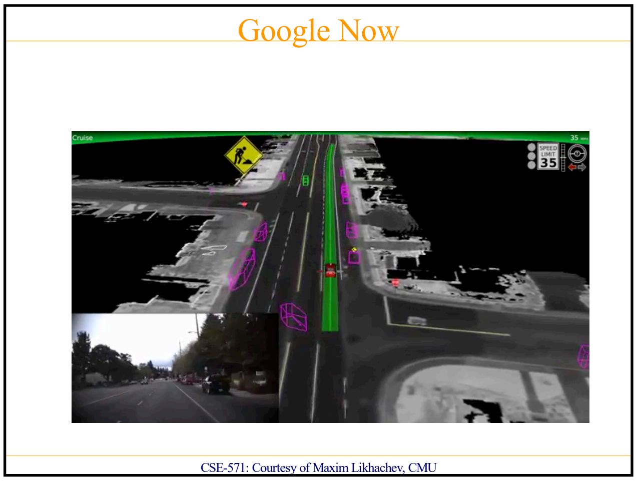


Example, again

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

CSE-571: Courtesy of Maxim Likhachev, CMU

61



Google Now

CSE-571: Courtesy of Maxim Likhachev, CMU

62

Summary

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

used a lot in real-time

think twice before trying to use it in real-time

- Planning under uncertainty
  - -Markov Decision Processes (MDP)
  - -Partially Observable Decision Processes (POMDP)

think three or four times before trying to use it in real-time

Many useful approximate solvers for MDP/POMDP exist!!

CSE-571: Courtesy of Maxim Likhachev, CMU

63