15-887
Planning, Execution and Learning

Low-level Planning Representations

Maxim Likhachev
Robotics Institute
Carnegie Mellon University

Example of Low*er*-level Planning

- Opening and moving through a door

Example of Lower-level Planning

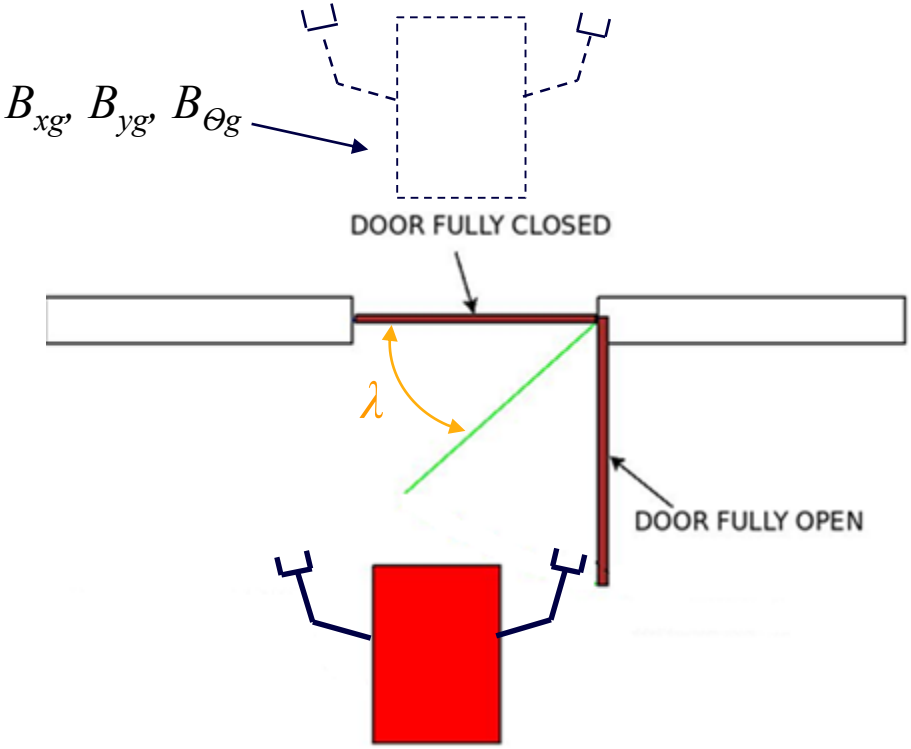
- Opening and moving through a door

– *State:* $<B_x, B_y, B_\Theta, LArm_{q1}, \ldots, LArm_{q7}, RArm_{q1}, \ldots, RArm_{q7}, \lambda, Gripper=\{open,close\}>$

– *Actions:* $<dB_x, dB_y, dB_\Theta, dLArm_{q1}, \ldots, dLArm_{q7}, dRArm_{q1}, \ldots, dRArm_{q7}, dGripper>$

– *Goal:* $B_x=B_{xg}$, $B_y=B_{yg}$, $B_\Theta=B_{\Theta g}$

– *Constraints:*
  - *Environmental (e.g., obstacles)*
  - *Kinematics of the robot*



Planning as Graph Search Problem

1. Construct a graph representing the planning problem

2. Search the graph for a (hopefully, close-to-optimal) path

The two steps above are often interleaved

Planning as Graph Search Problem

1. Construct a graph representing the planning problem

2. Search the graph for a (hopefully, close-to-optimal) path

The two steps above are often interleaved

Interleaving Search and Graph Construction

Graph Search using an **Implicit Graph** (allocated as needed by the search):

1. *Instantiate Start state*

2. *Start searching with the Start state using functions*

   a) *Succs = GetSuccessors (State s, Action)*
   b) *ComputeEdgeCost (State s, Action a, State s')*

   and allocating memory for the generated states

Using Implicit Graphs is critical for most (>2D) problems in Robotics

Planning as Graph Search Problem

1. Construct a graph representing the planning problem

2. Search the graph for a (hopefully, close-to-optimal) path

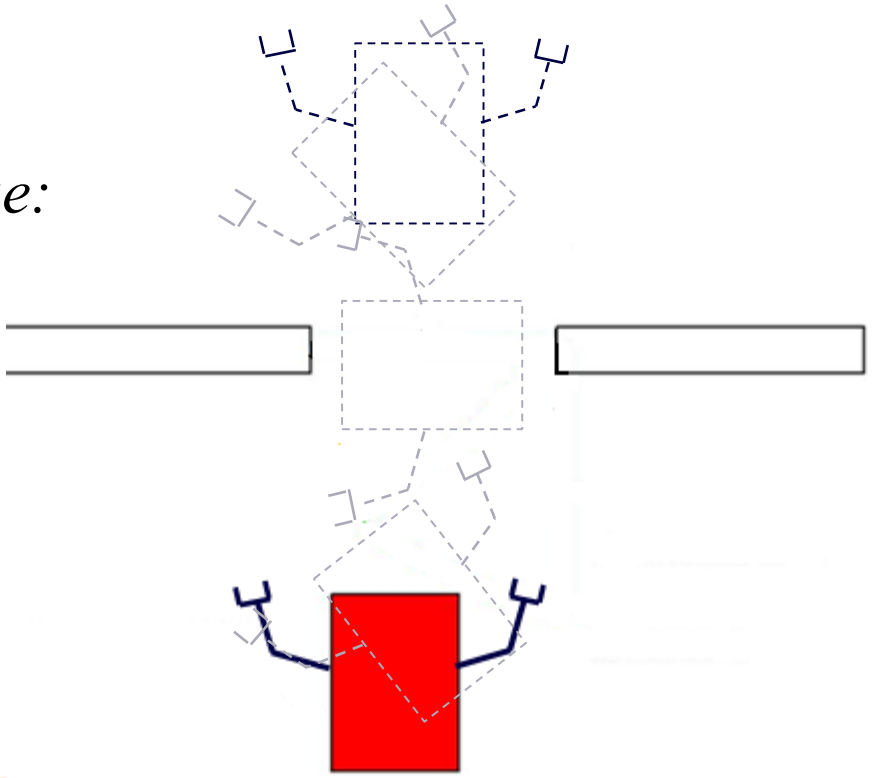
The two steps above are often interleaved

Configuration Space

- Configuration is legal if it does not intersect any obstacles and is valid

- Configuration Space is the set of legal configurations

Simple problem of planning for the base:

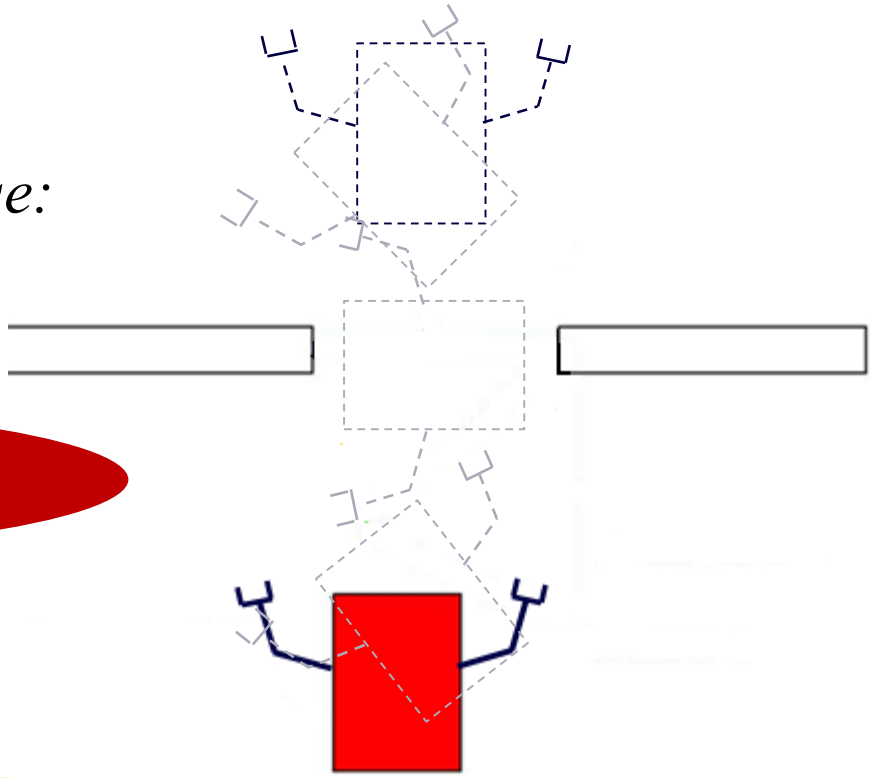


Configuration Space

- Configuration is legal if it does not intersect any obstacles and is valid

- Configuration Space is the set of legal configurations

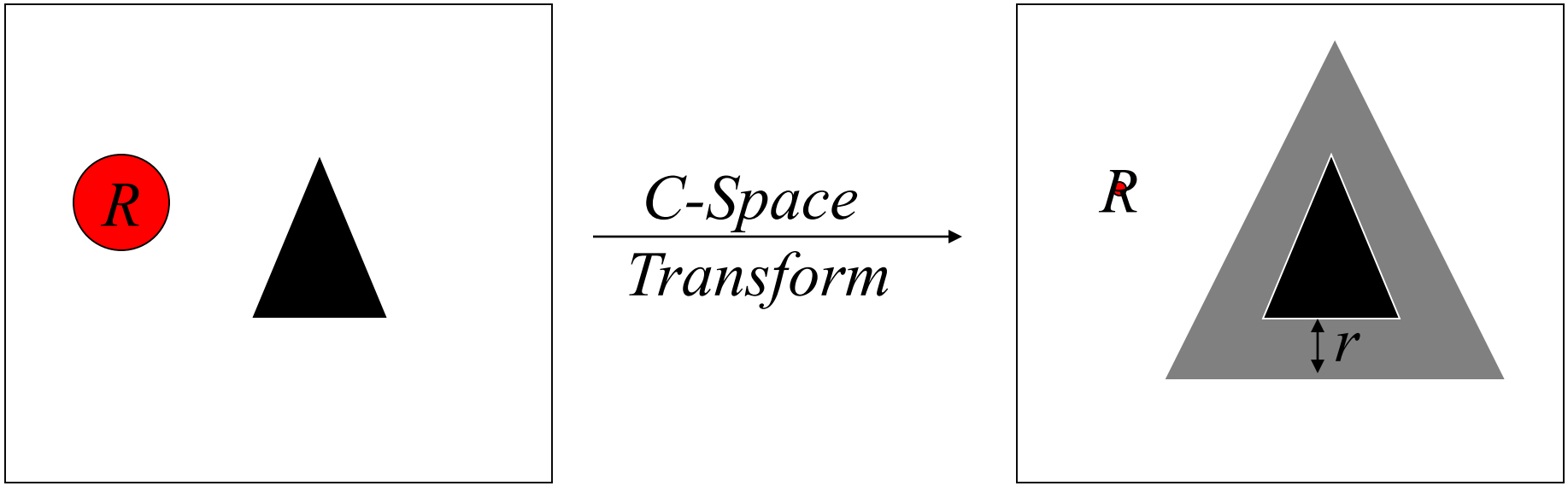
Simple problem of planning for the base:



What is the dimensionality of this configuration space?

C-Space Transform

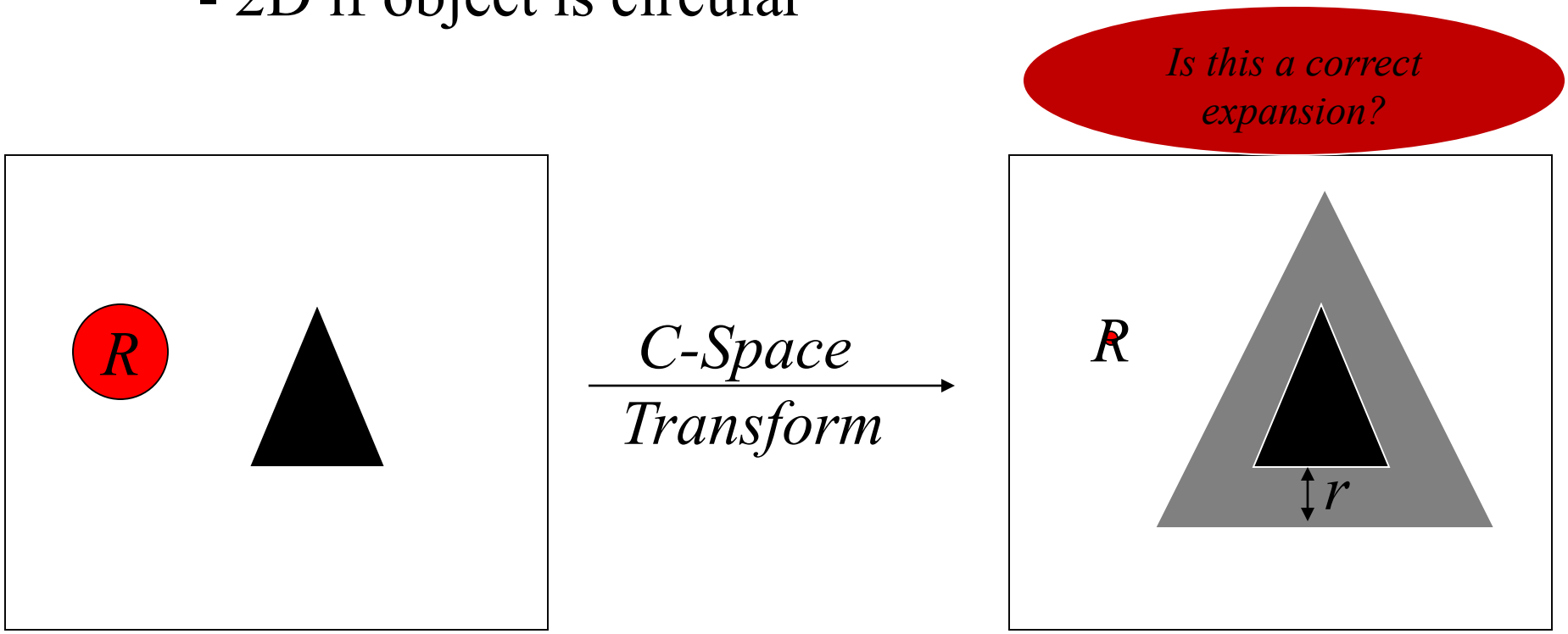
- Configuration space for rigid-body objects in 2D world is:
  - 2D if object is circular



- expand all obstacles by the radius of the object r
- planning can be done for a point R (and not a circle anymore)

C-Space Transform

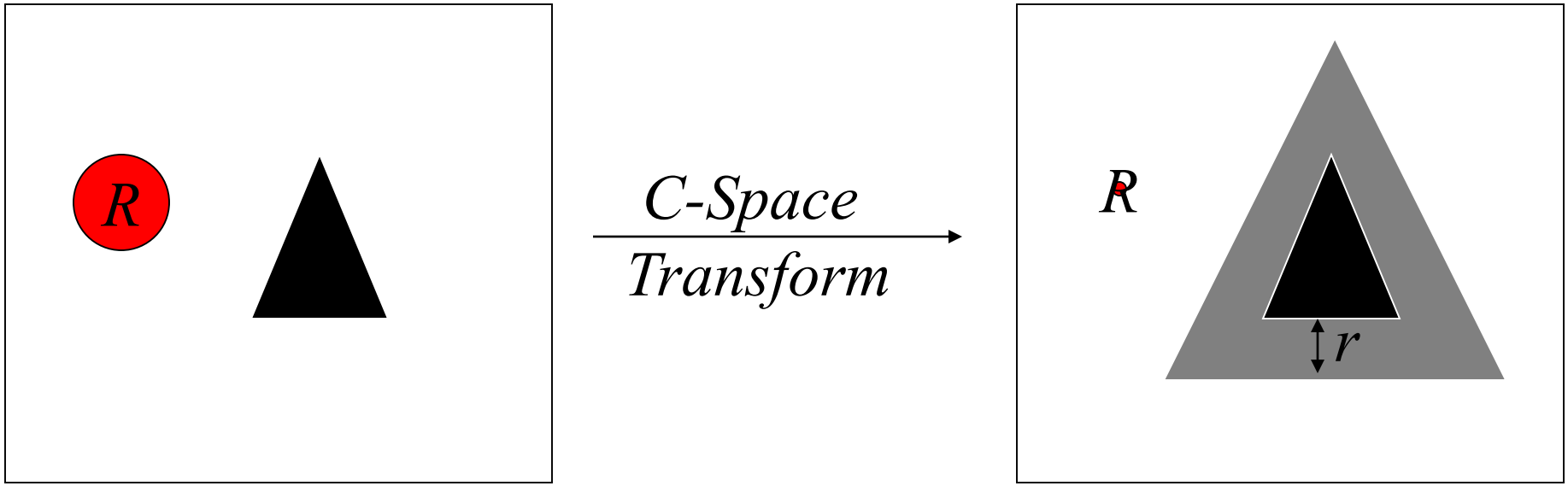
- Configuration space for rigid-body objects in 2D world is:
  - 2D if object is circular



- expand all obstacles by the radius of the object r
- planning can be done for a point R (and not a circle anymore)

C-Space Transform

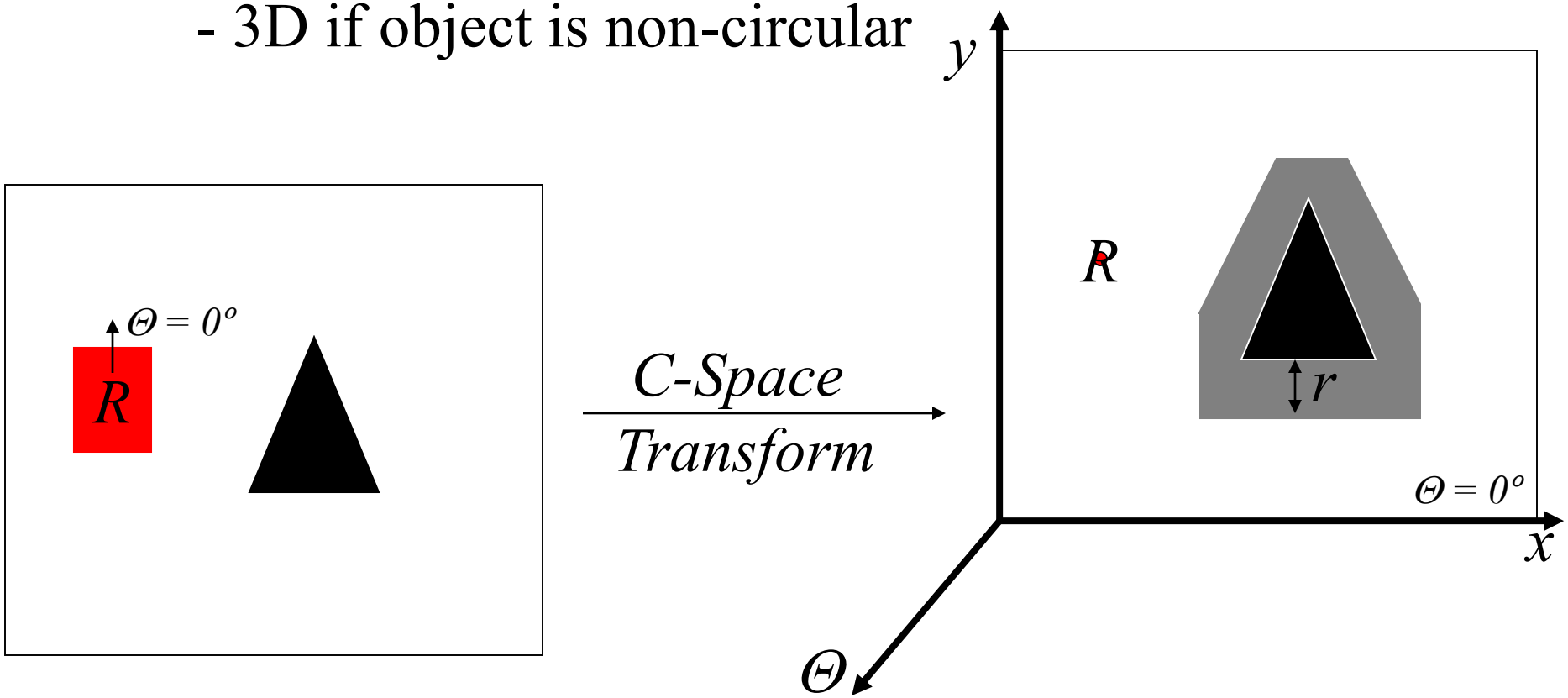
- Configuration space for rigid-body objects in 2D world is:
  - 2D if object is circular



- advantage: planning is faster for a single point *why?*
- disadvantage: need to expand obstacles every time map is updated (O(n) methods exist to compute distance transforms)

C-Space Transform

- Configuration space for arbitrary objects in 2D world is:
  - 3D if object is non-circular



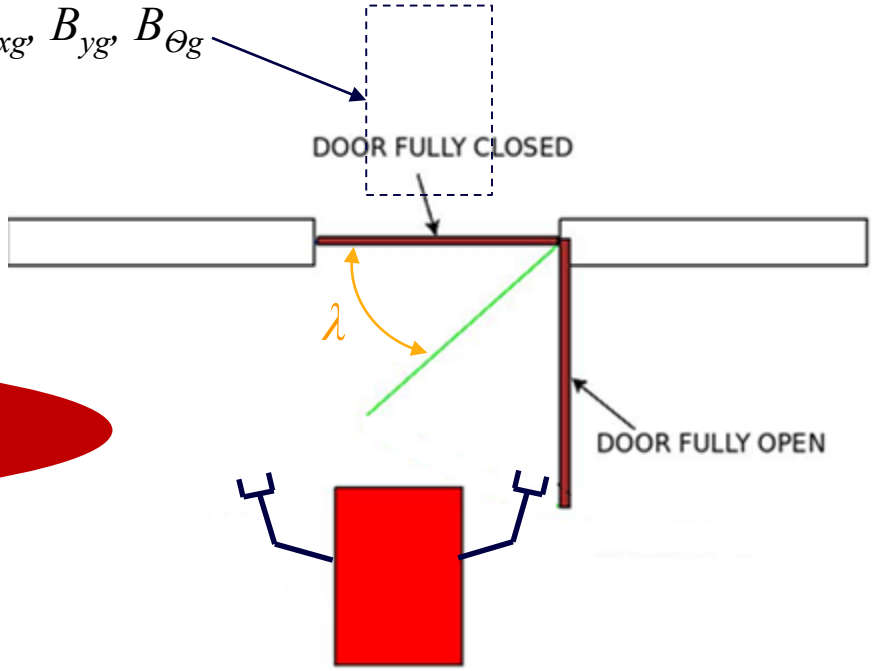
- advantage: planning is faster for a single point
- disadvantage: constructing C-space is expensive

Example of Lower-level Planning

- Opening and moving through a door

– *State:* $<B_x, B_y, B_\Theta, LArm_{q1}, \ldots, LArm_{q7}, RArm_{q1}, \ldots, RArm_{q7}, \lambda, Gripper=\{open,close\}>$
– *Actions:* $<dB_x, dB_y, dB_\Theta, dLArm_{q1}, \ldots, dLArm_{q7}, dRArm_{q1}, \ldots, dRArm_{q7}, dGripper>$
– *Goal:* $B_x=B_{xg}, B_y=B_{yg}, B_\Theta=B_{\Theta g}$

– *Constraints:*
  - *Environmental (e.g., obstacles)*
  - *Kinematics of the robot*



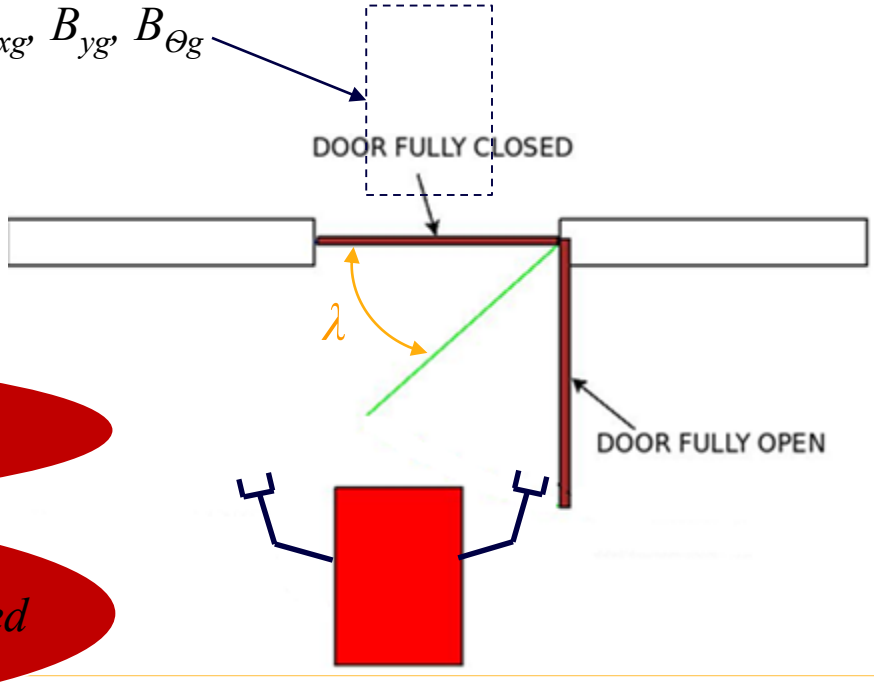
What is the dimensionality of this configuration space?

Example of Lower-level Planning

- Opening and moving through a door

– *State:* $<B_x, B_y, B_\Theta, LArm_{q1}, \ldots, LArm_{q7}, RArm_{q1}, \ldots, RArm_{q7}, \lambda, Gripper=\{open,close\}>$
– *Actions:* $<dB_x, dB_y, dB_\Theta, dLArm_{q1}, \ldots, dLArm_{q7}, dRArm_{q1}, \ldots, dRArm_{q7}, dGripper>$
– *Goal:* $B_x=B_{xg}, B_y=B_{yg}, B_\Theta=B_{\Theta g}$

– *Constraints:*
  - *Environmental (e.g., obstacles)*
  - *Kinematics of the robot*



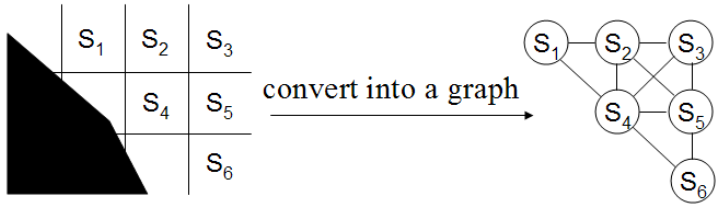
What is the dimensionality of this configuration space?

That's why usually configuration space is NOT explicitly constructed and collision checking is done on demand

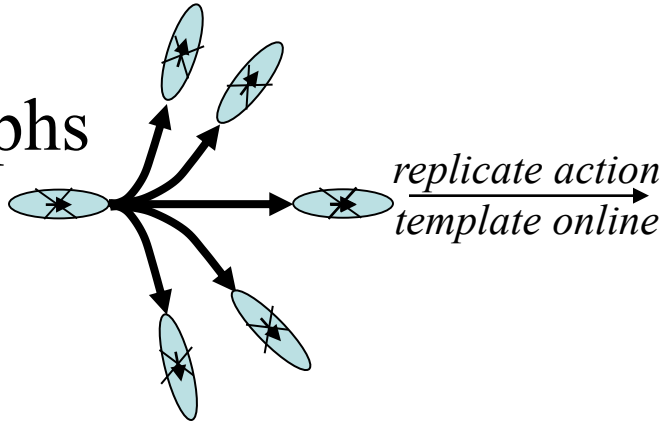
Graph Construction

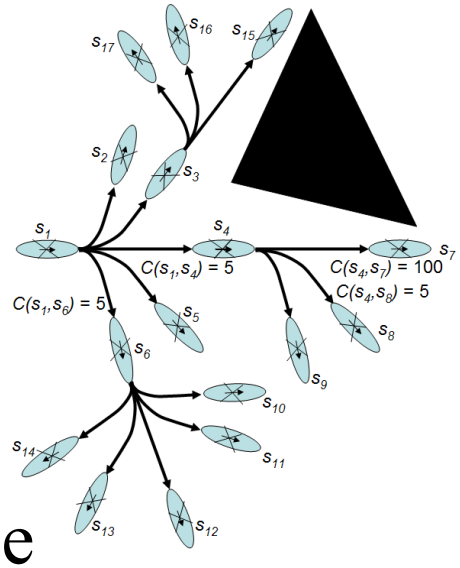
- Cell decomposition

  - X-connected grids



  - lattice-based graphs

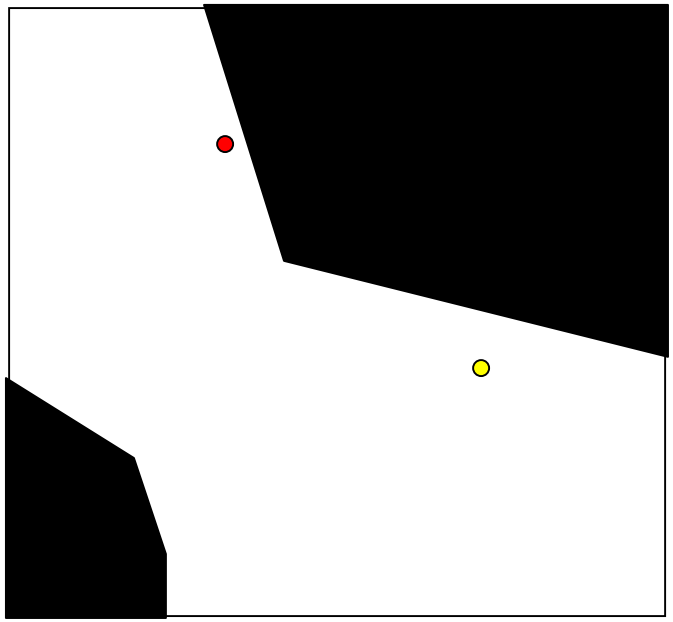




- Skeletonization of the environment/C-Space
  - -Visibility graphs
  - Voronoi diagrams
  - Probabilistic roadmaps

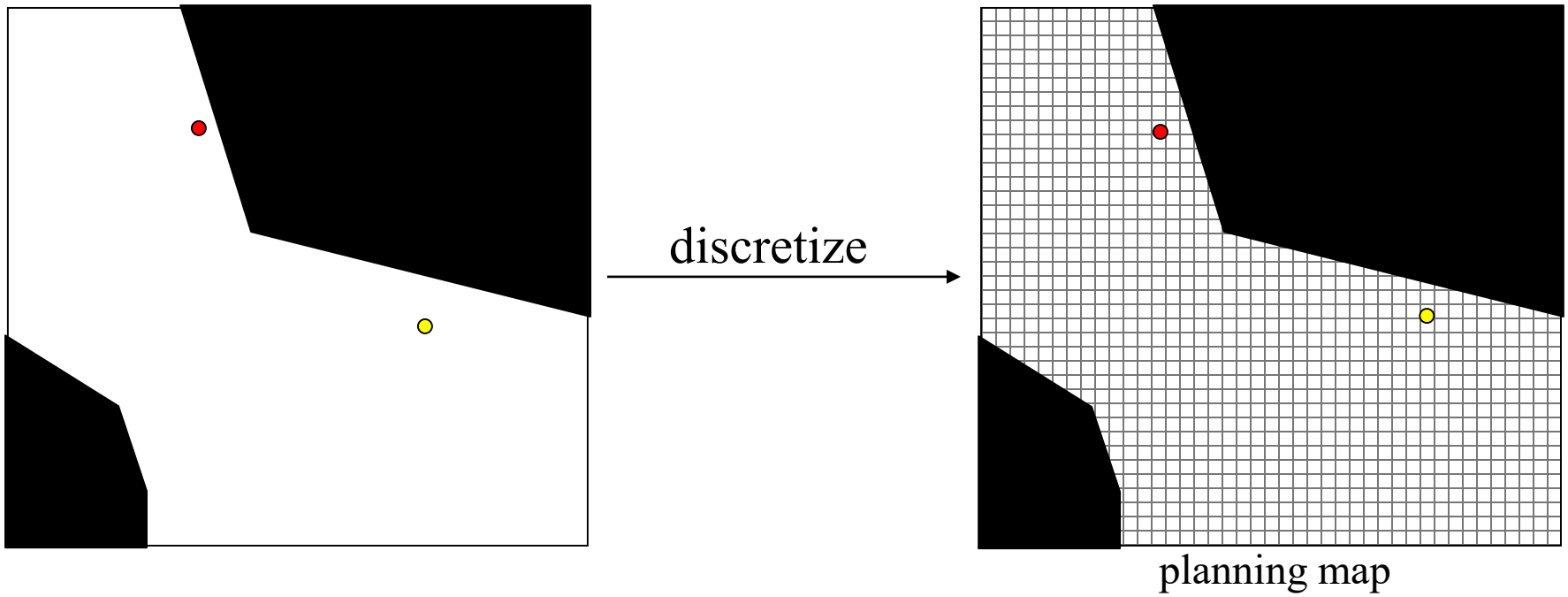
Planning via Cell Decomposition

- Exact Cell Decomposition:
  - overlay convex exact polygons over the free C-space
  - construct a graph, search the graph for a path
  - overly expensive for non-trivial environments and/or above 2D



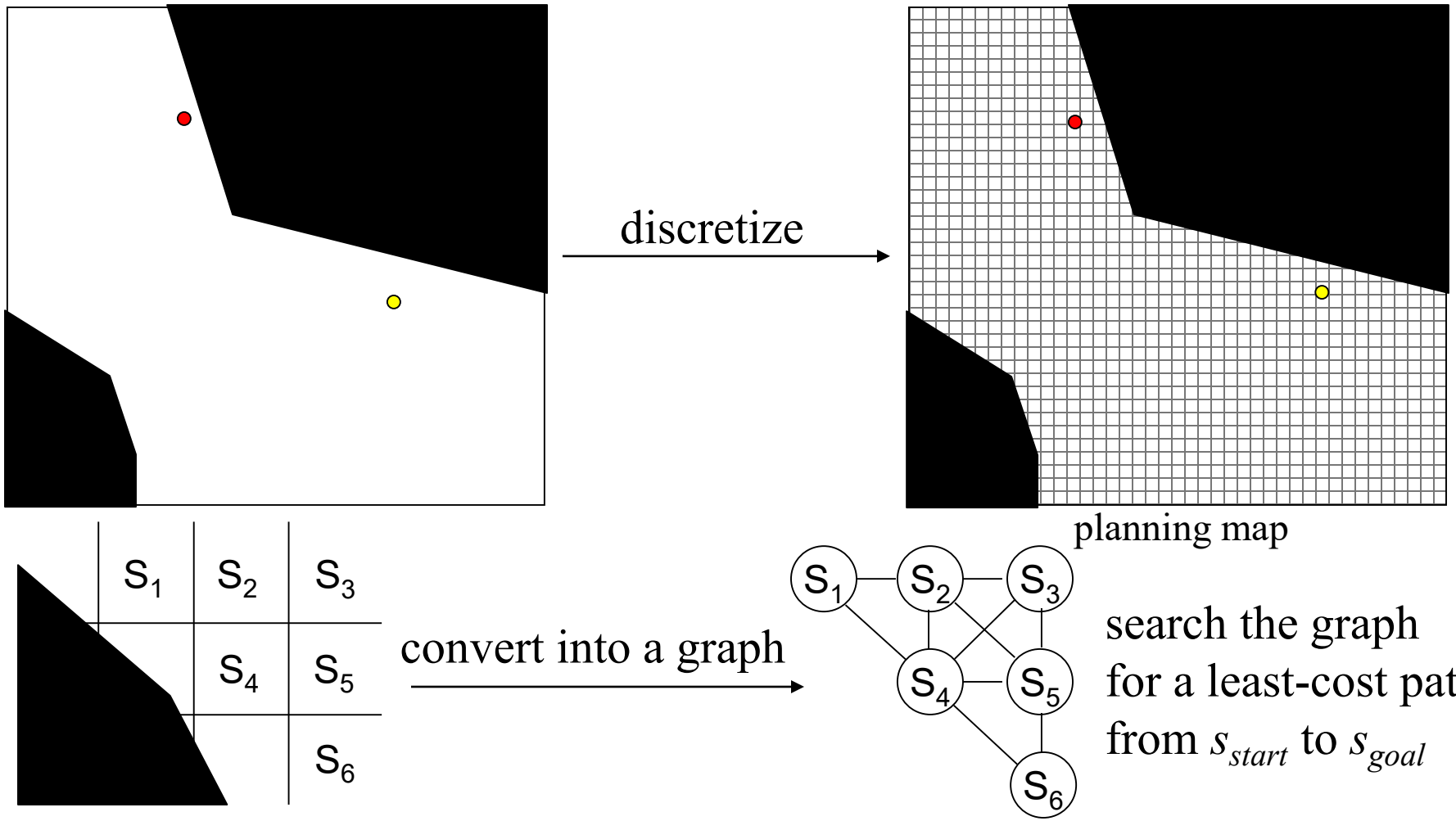
Planning via Cell Decomposition

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



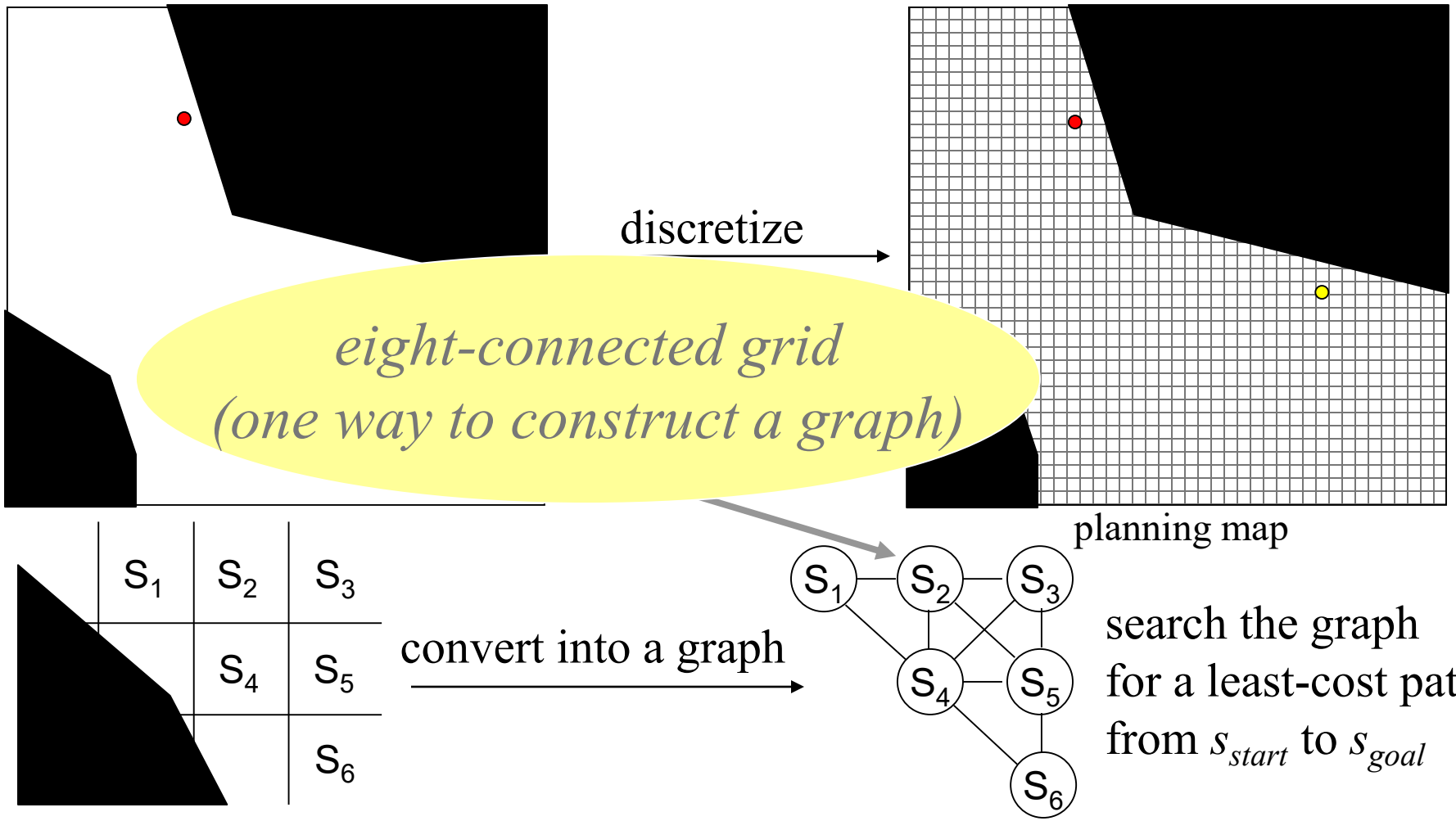
Planning via Cell Decomposition

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



Planning via Cell Decomposition

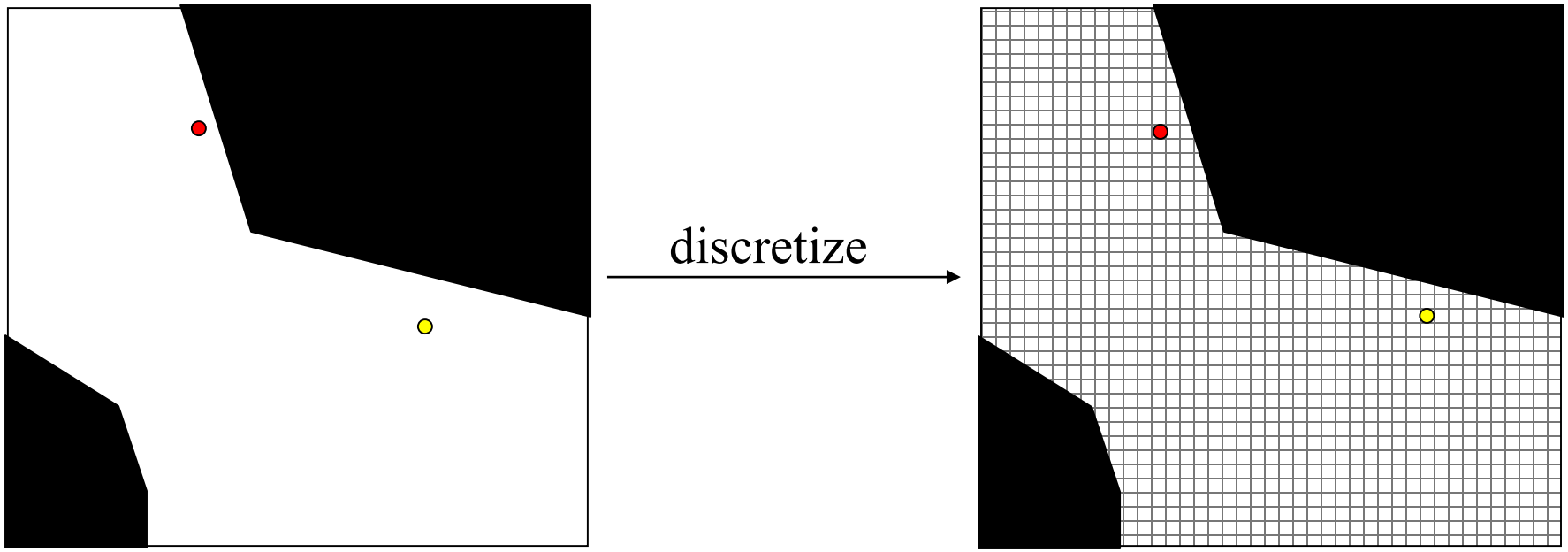
- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path



Planning via Cell Decomposition

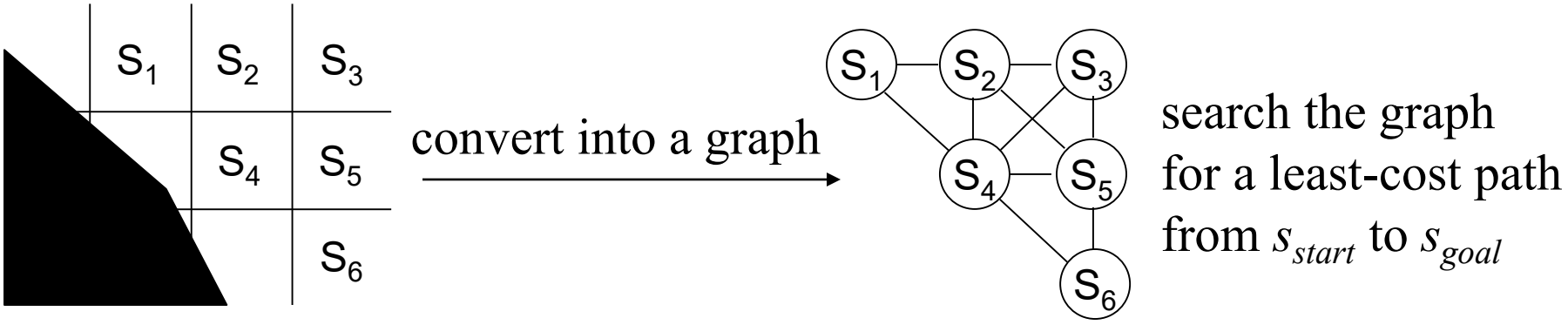
- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path
    - VERY popular due to its simplicity
      - expensive in high-dimensional spaces

construct the grid on-the-fly, i.e. while planning – still expensive



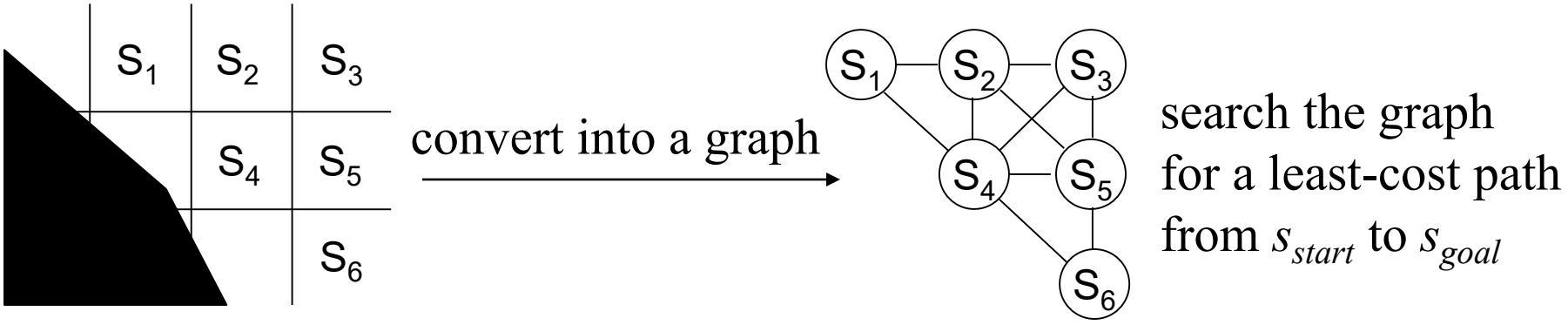
Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - what to do with partially blocked cells?



Planning via Cell Decomposition

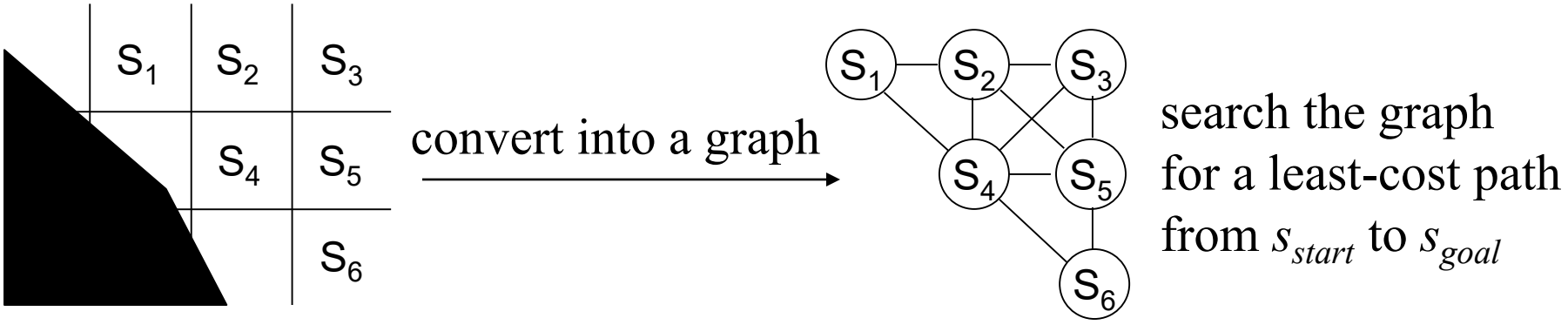
- Approximate Cell Decomposition:
  - what to do with partially blocked cells?
  - make it untraversable – incomplete (may not find a path that exists)



Planning via Cell Decomposition

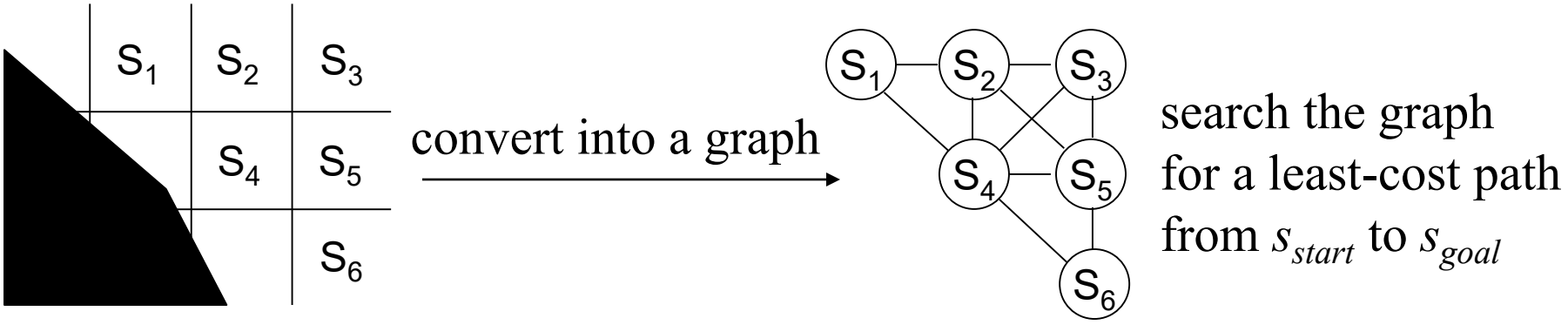
- Approximate Cell Decomposition:
  - what to do with partially blocked cells?
  - make it traversable – unsound (may return invalid path)

so, what's the solution?



Planning via Cell Decomposition

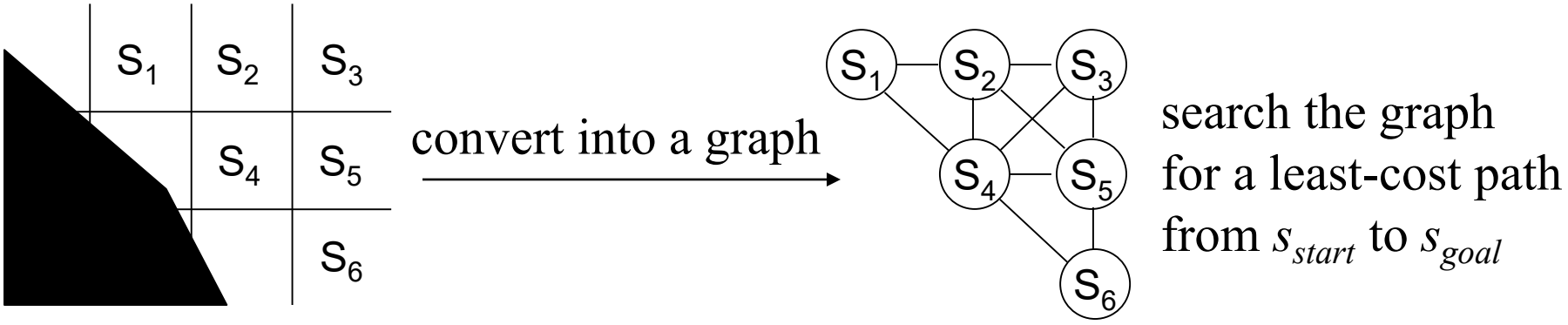
- Approximate Cell Decomposition:
  - solution 1:
    - make the discretization very fine
    - expensive, especially in high-D



Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - solution 2:
    - make the discretization adaptive
    - various ways possible

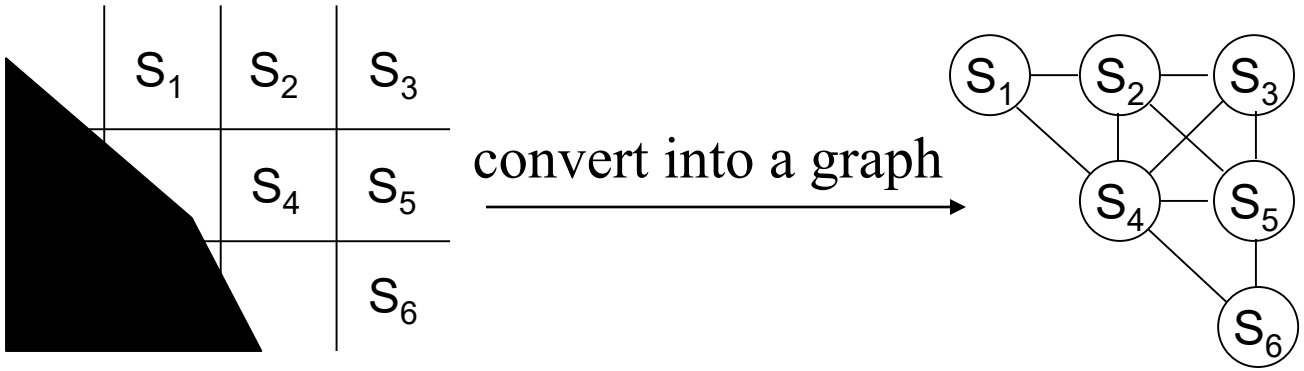
Any ideas?



Planning via Cell Decomposition

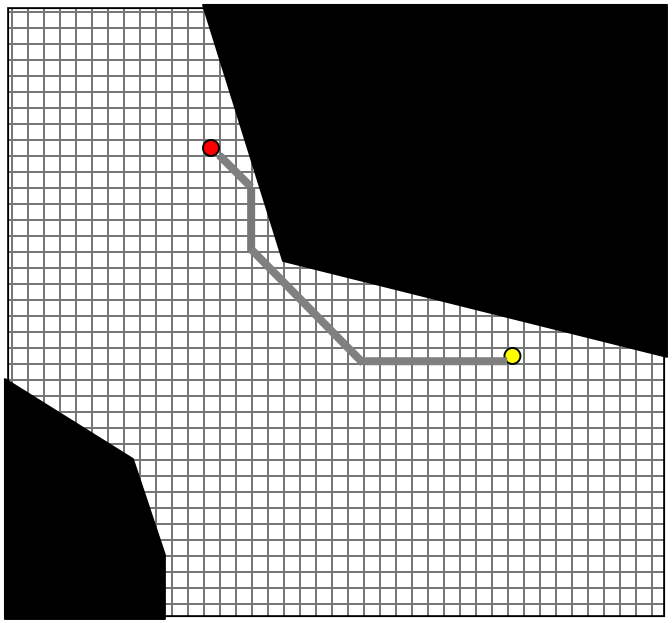
- Graph construction:
  - connect neighbors

eight-connected grid



Planning via Cell Decomposition

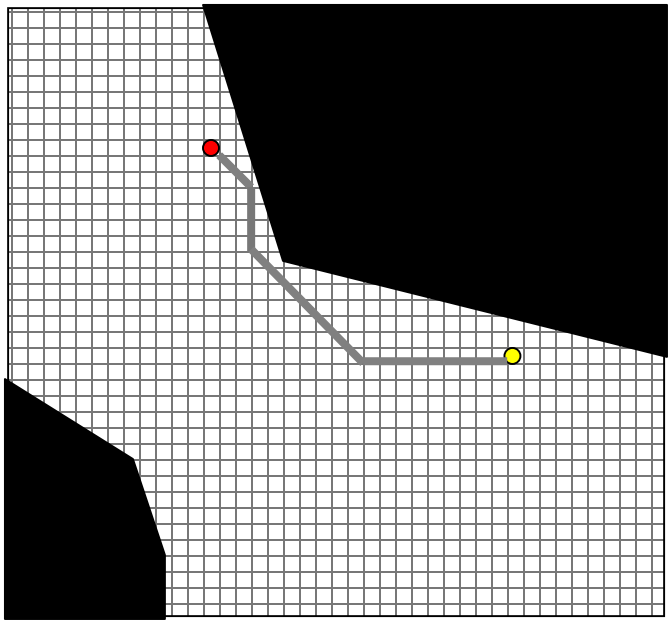
- Graph construction:
  - connect neighbors
  - path is restricted to 45º degrees



Planning via Cell Decomposition

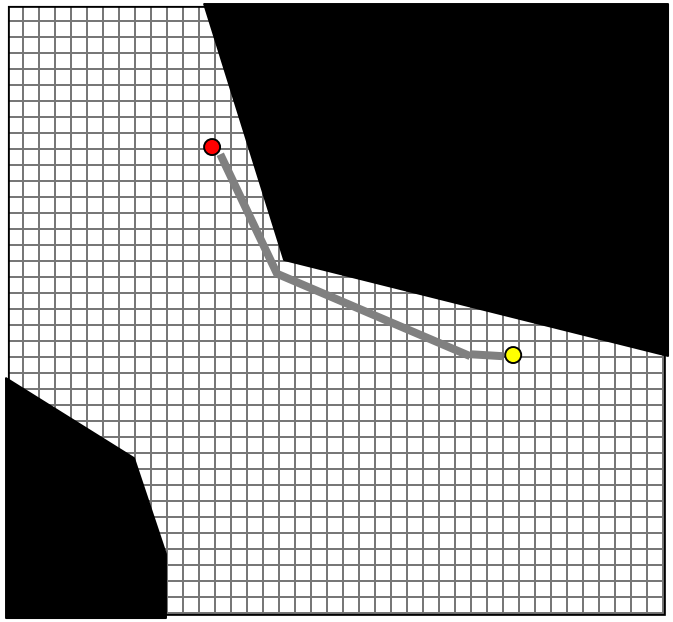
- Graph construction:
  - connect neighbors
  - path is restricted to 45º degrees

Ideas to improve it?

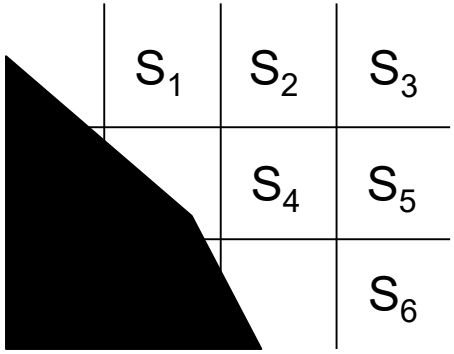


Planning via Cell Decomposition

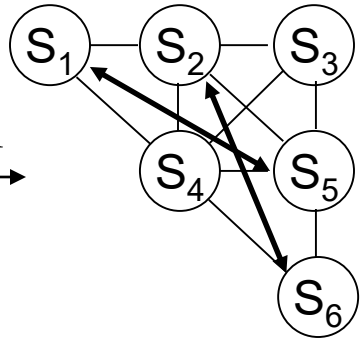
- Graph construction:
  - connect cells to neighbor of neighbors
  - path is restricted to 22.5º degrees



16-connected grid



convert into a graph

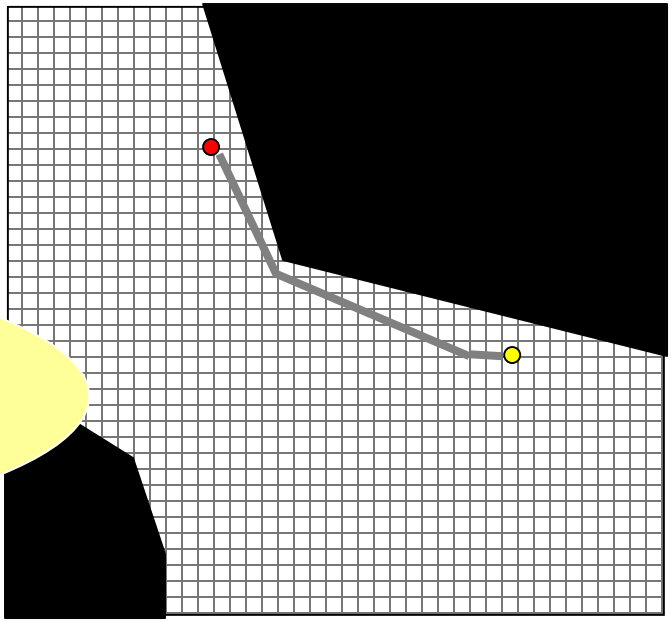


Planning via Cell Decomposition

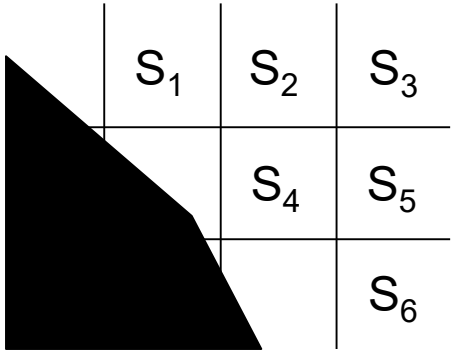
- Graph construction:
  - connect cells to neighbor of neighbors
  - path is restricted to 22.5º degrees

Disadvantages?

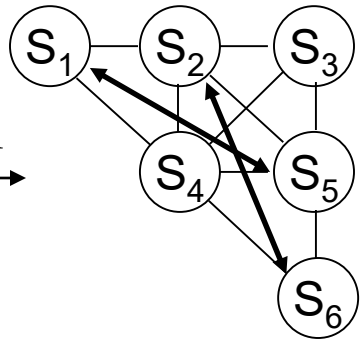
Dynamically generated directions (for low-d problems):
Field D [Ferguson & Stentz, '06],*
Theta [Nash & Koenig, '13]*



16-connected grid



convert into a graph



Planning via Cell Decomposition

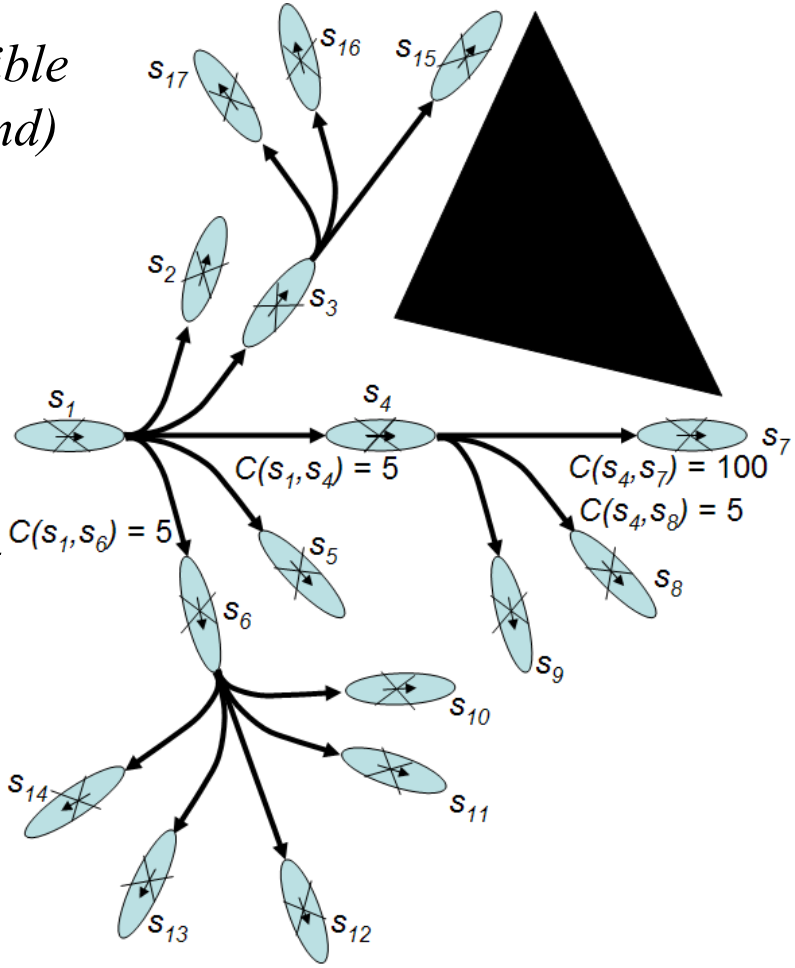
- Graph construction:
  - lattice graph for computing feasible paths [Pivtoraiko & Kelly '05]

outcome state is the center of the corresponding cell

each transition is feasible (constructed beforehand)

motion primitives

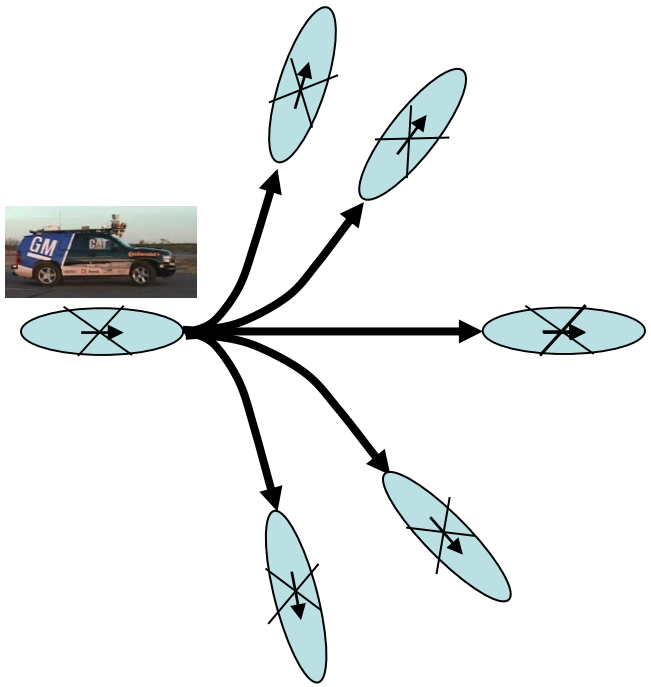
replicate it online



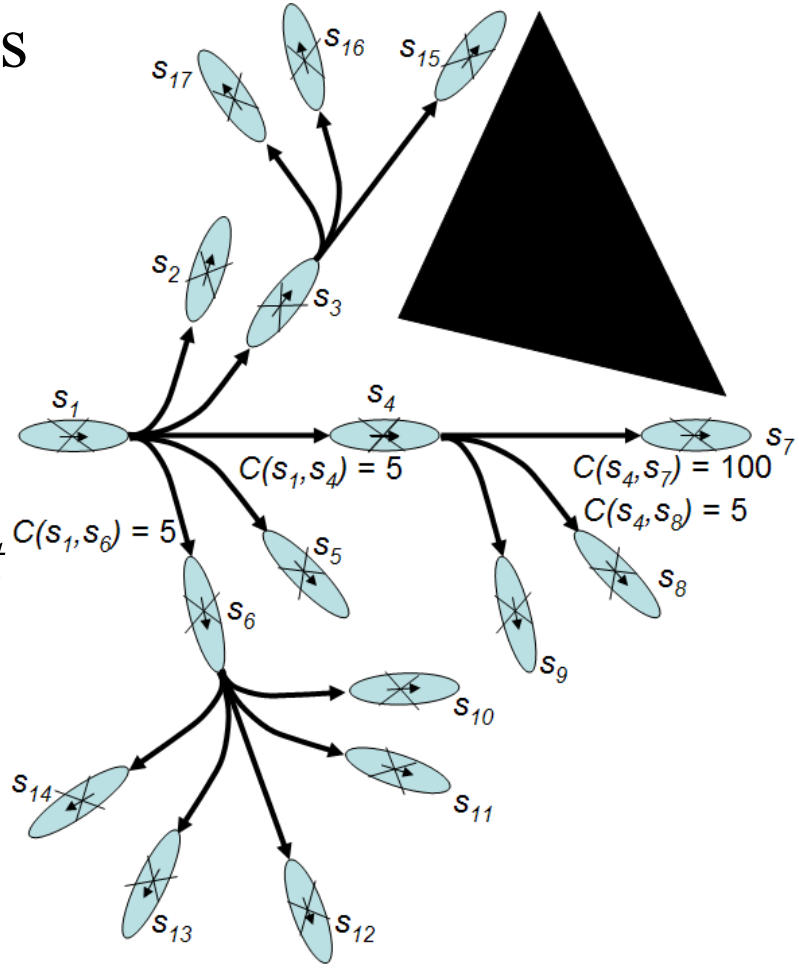
Planning via Cell Decomposition

- Graph construction:
  - lattice graph [Pivtoraiko & Kelly '05]
  - pros: sparse graph, feasible paths
  - cons: possible incompleteness

motion primitives

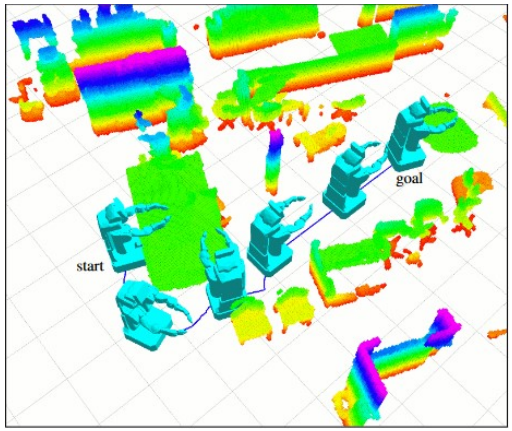


replicate it online

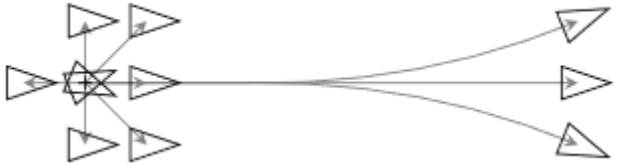


Planning via Cell Decomposition

- Graph construction:
  - lattice graph [Pivtoraiko & Kelly '05]



example of motion primitives for PR2

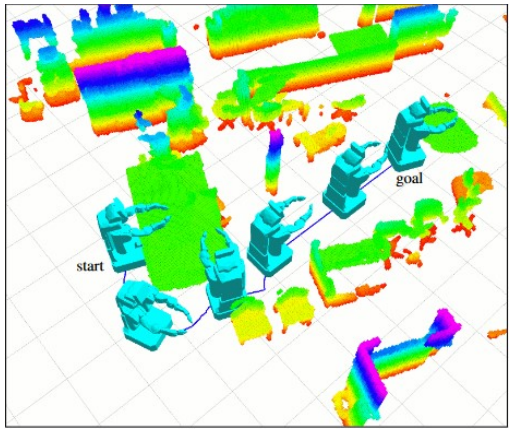


Why forward motion primitives duplicated (one short, one long)?

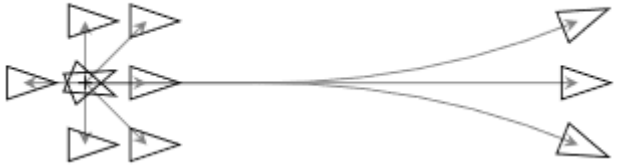
[Hornung et al., '12]

Planning via Cell Decomposition

- Graph construction:
  - lattice graph [Pivtoraiko & Kelly '05]



example of motion primitives for PR2



How is it different when planning dynamically feasible paths?

[Hornung et al., '12]

Skeletonization of the C-Space

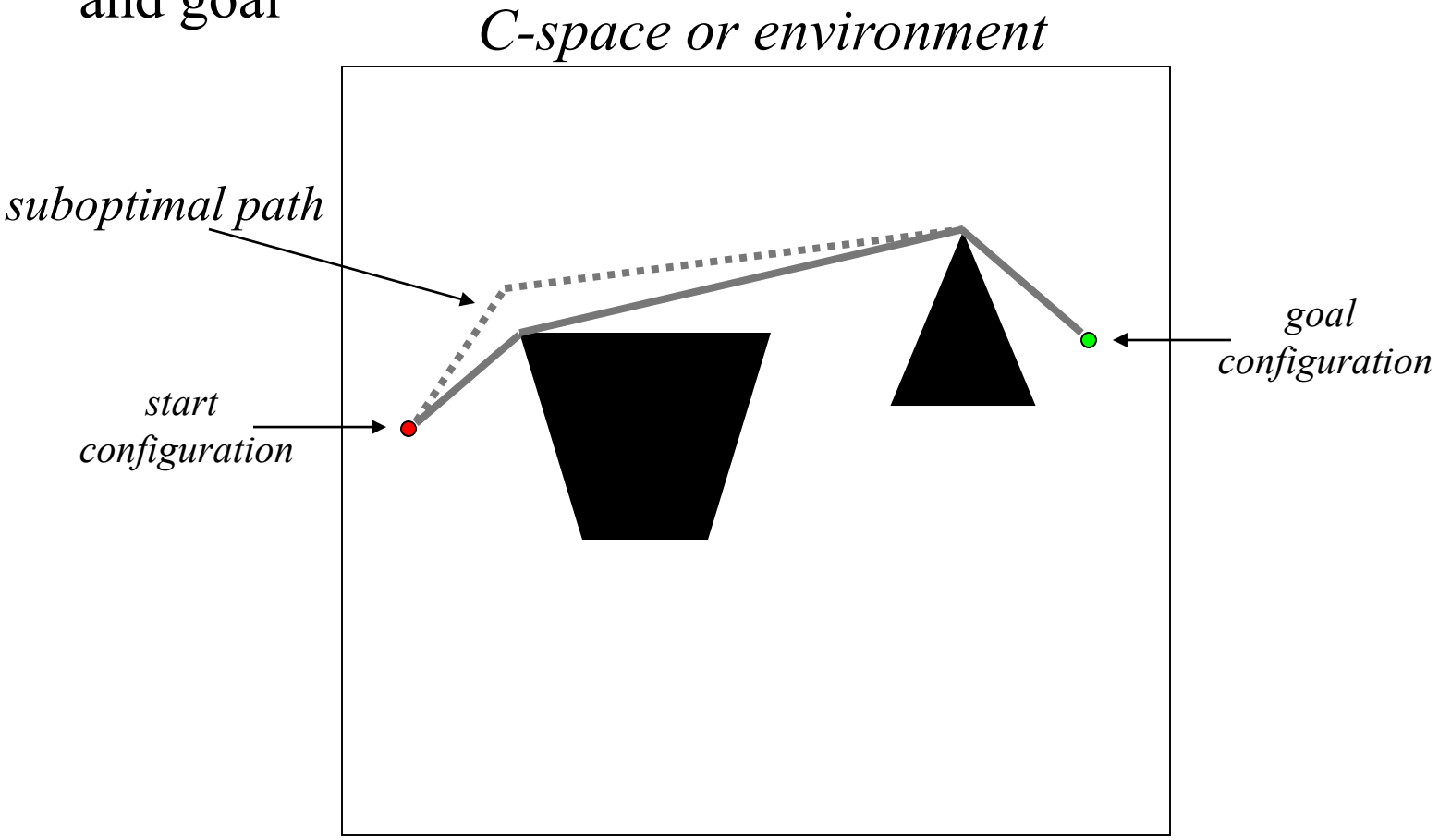
Skeletonization: construction of a unidimensional representation of the C-space

- Visibility graph
- Voronoi diagram
- Probabilistic road-map

Planning via Skeletonization

- Visibility Graphs [Wesley & Lozano-Perez '79]

  - based on idea that the shortest path consists of obstacle-free straight line segments connecting all obstacle vertices and start and goal

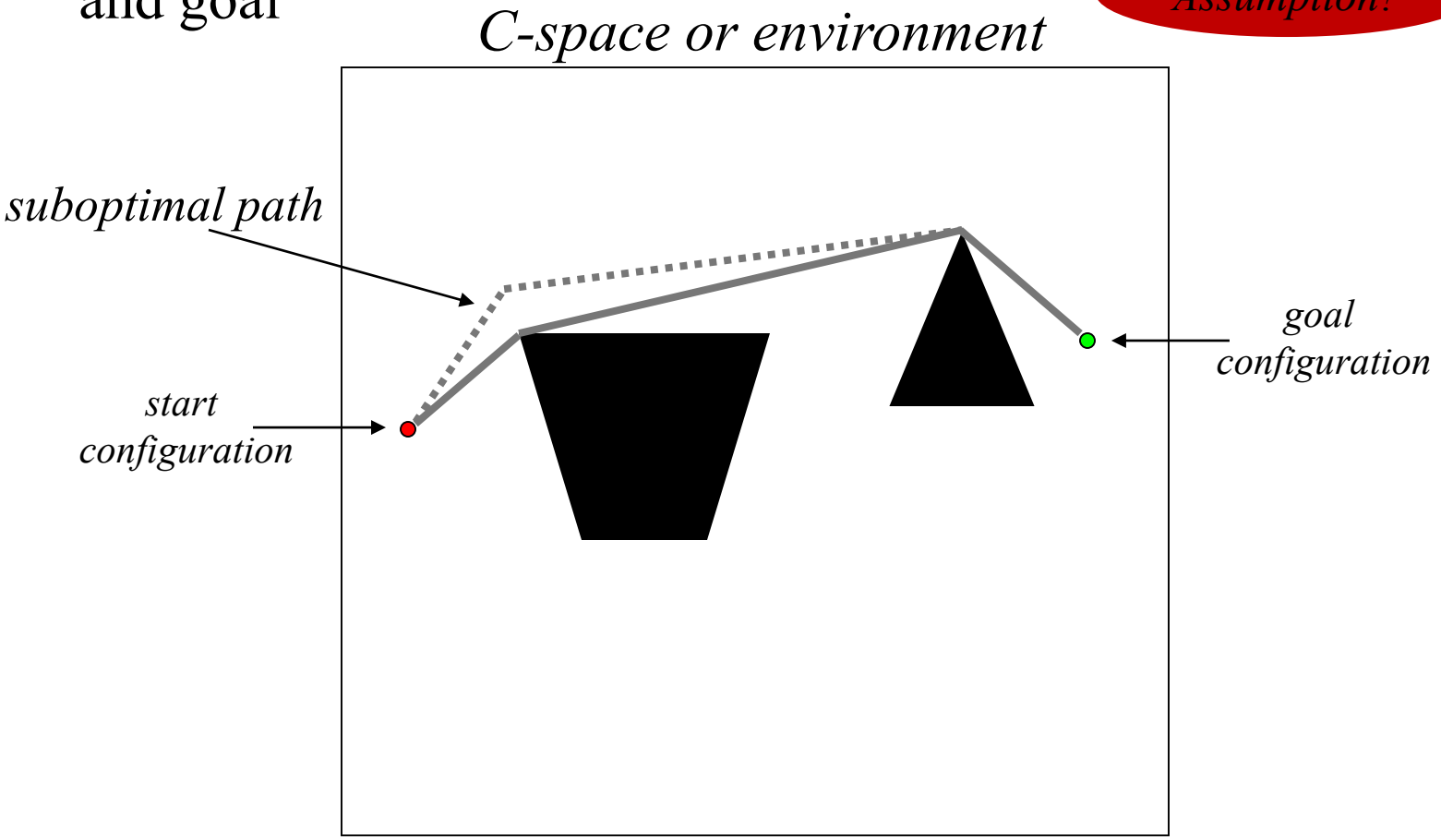


Planning via Skeletonization

- Visibility Graphs [Wesley & Lozano-Perez '79]

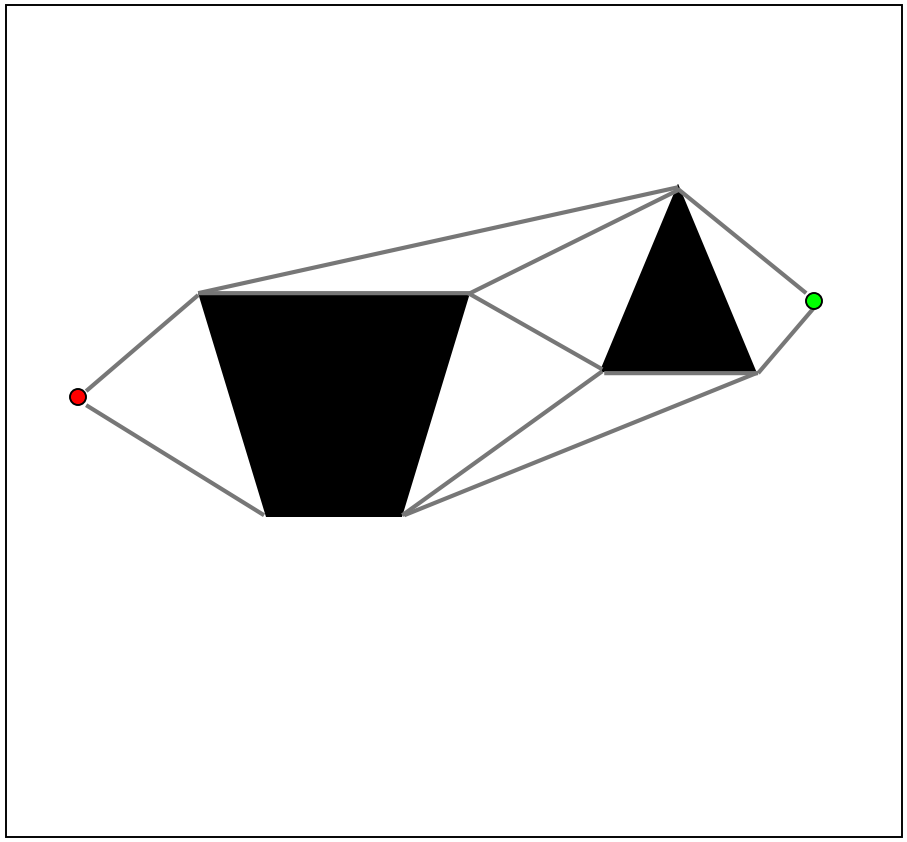
  - based on idea that the shortest path consists of obstacle-free straight line segments connecting all obstacle vertices and start and goal

Assumption?



Planning via Skeletonization

- Visibility Graphs [Wesley & Lozano-Perez '79]
  - construct a graph by connecting all vertices, start and goal by obstacle-free straight line segments (graph is $O(n^2)$, where n - # of vert.)
  - search the graph for a shortest path



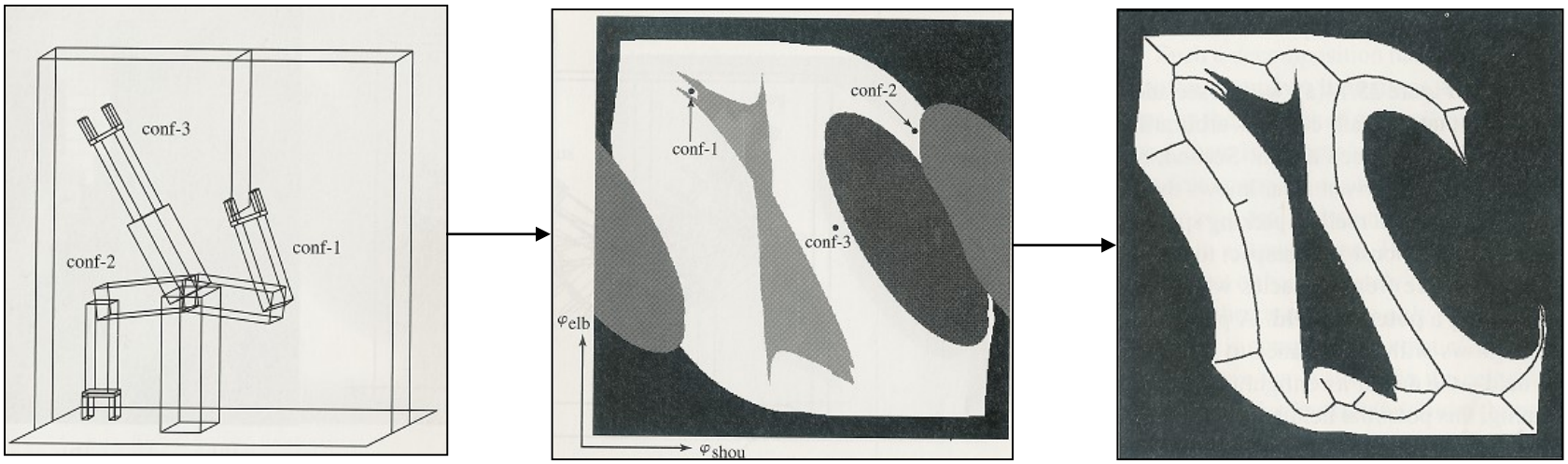
Planning via Skeletonization

- Visibility Graphs
  - advantages:
    - independent of the size of the environment
  - disadvantages:
    - path is too close to obstacles
    - hard to deal with non-uniform cost function
    - hard to deal with non-polygonal obstacles

Planning via Skeletonization

- Voronoi diagrams [Rowat '79]
  - voronoi diagram: set of all points that are equidistant to two nearest obstacles
  - based on the idea of maximizing clearance instead of minimizing travel distance

2DOF arm *Configuration space* *Voronoi diagram*

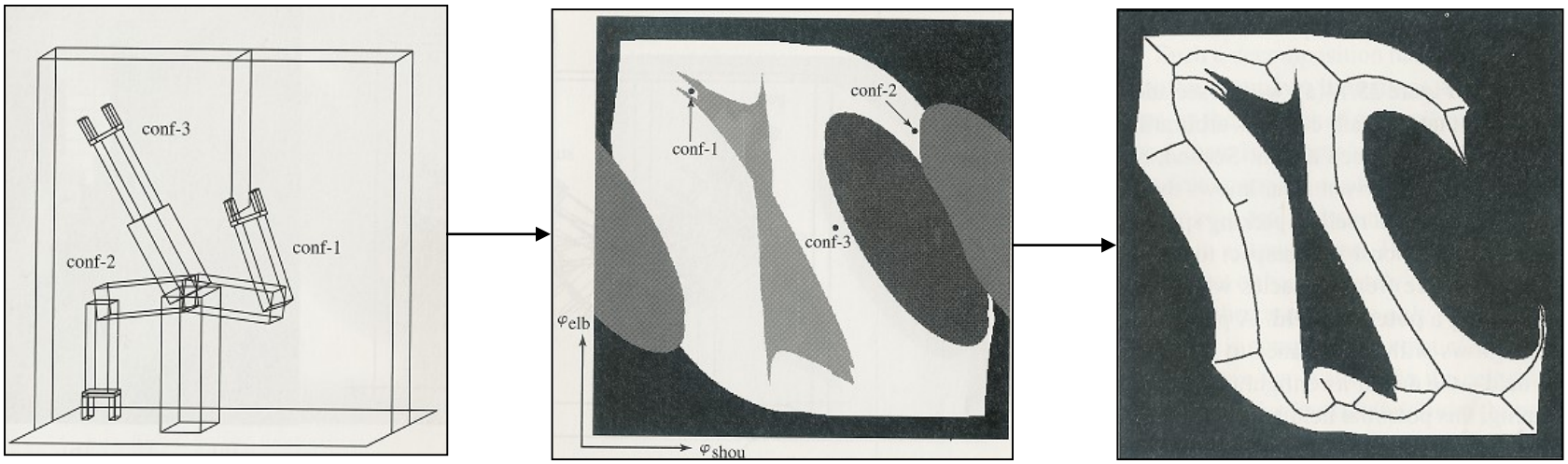


the example above is borrowed from "AI: A Modern Approach" by S. RussellL & P. Norvig

Planning via Skeletonization

- Voronoi diagrams
  - compute voronoi diagram (O (n log n), where n - # of invalid configurations)
  - add a shortest path segment from start to the nearest segment of voronoi diagram
  - add a shortest path segment from goal to the nearest segment of voronoi diagram
  - compute shortest path in the graph

2DOF arm *Configuration space* *Voronoi diagram*

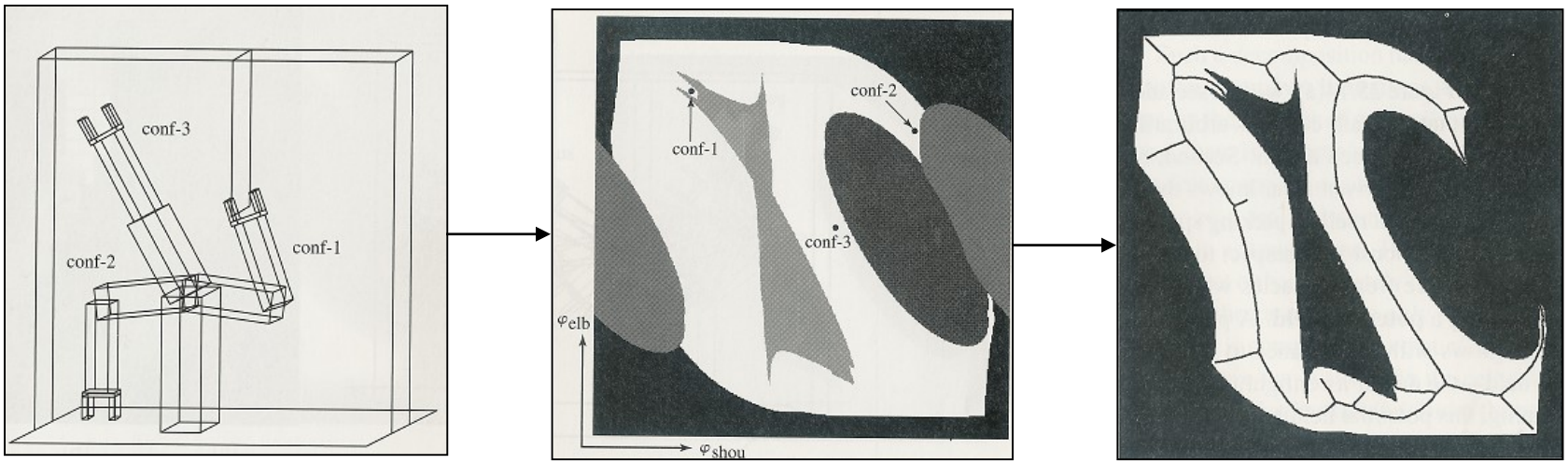


the example above is borrowed from "AI: A Modern Approach" by S. RussellL & P. Norvig

Planning via Skeletonization

- Voronoi diagrams
  - advantages:
    - tends to stay away from obstacles
    - independent of the size of the environment
  - disadvantages:
    - can result in highly suboptimal paths

2DOF arm *Configuration space* *Voronoi diagram*



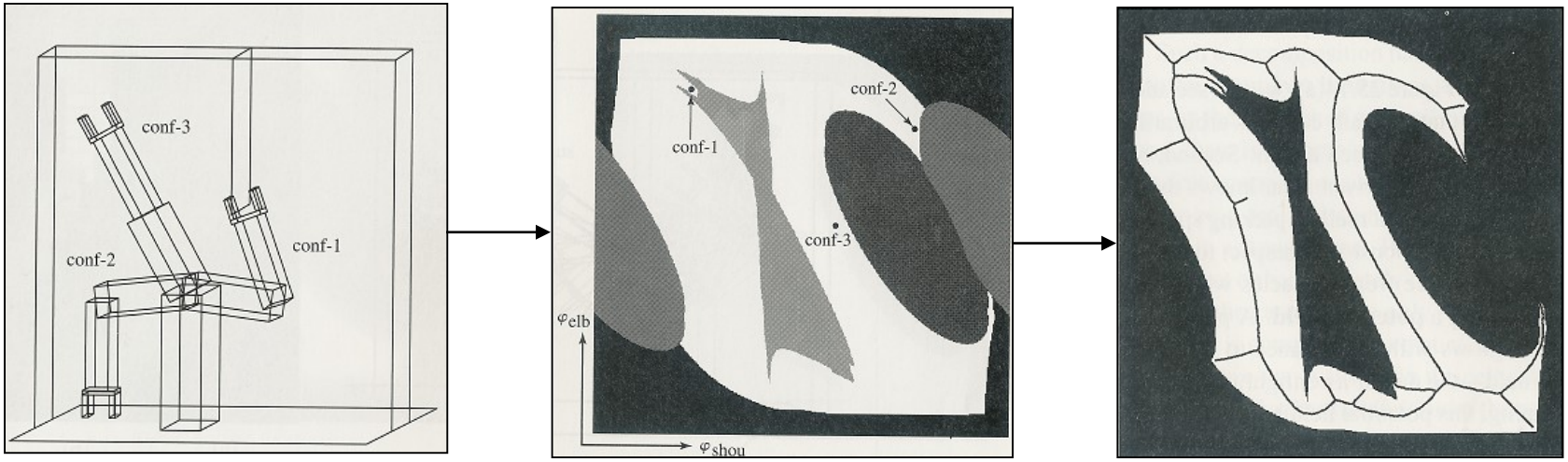
the example above is borrowed from "AI: A Modern Approach" by S. RussellL & P. Norvig

Planning via Skeletonization

- Voronoi diagrams
  - advantages:
    - tends to stay away from obstacles
    - independent of the size of the environment
  - disadvantages:
    - can result in highly suboptimal paths

In which environments?

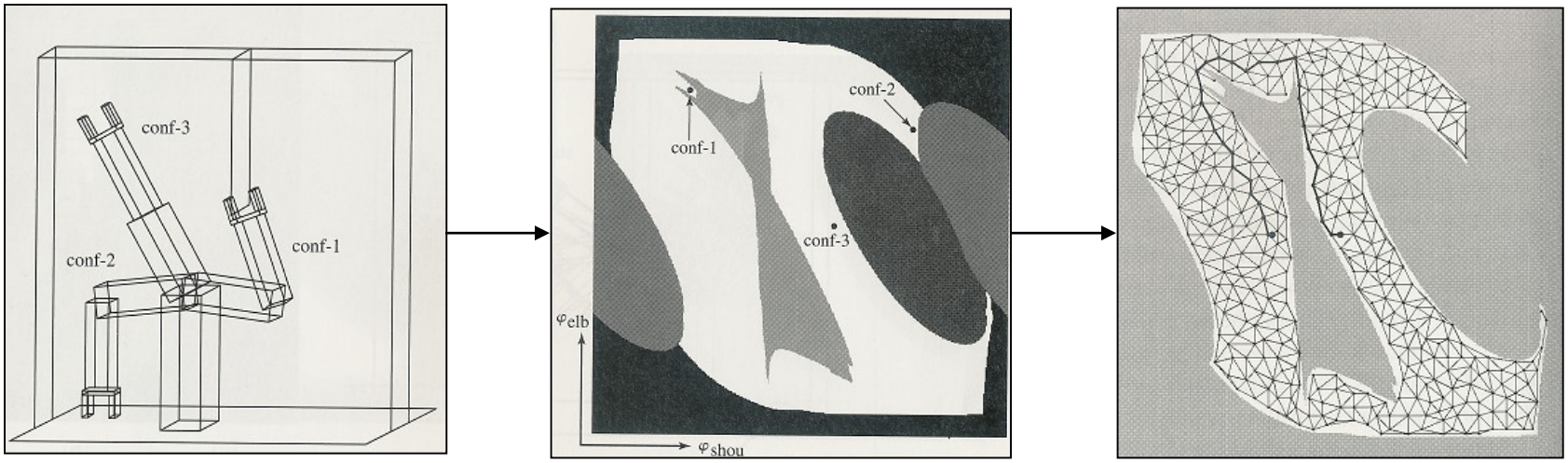
2DOF arm | *Configuration space* | *Voronoi diagram*



the example above is borrowed from "AI: A Modern Approach" by S. RussellL & P. Norvig

Planning via Skeletonization

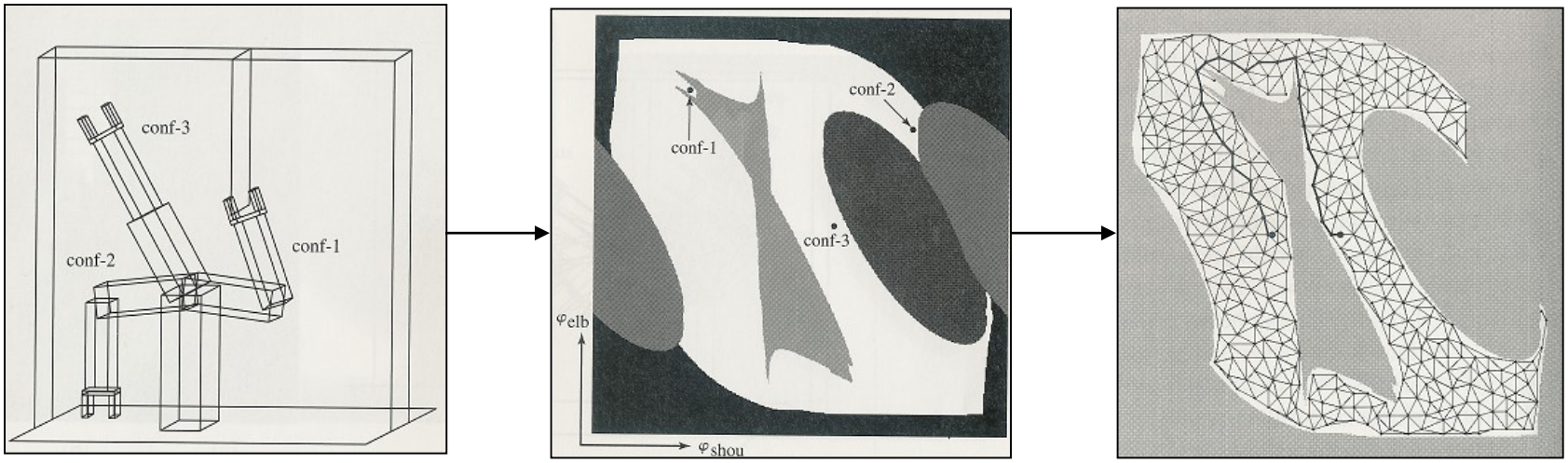
- Sampling-based planning:
  - generate a sparse (sample-based) representation (graph) of a free C-space (C_{free})
  - search the generated representation for a solution
  - can interleave the construction of the representation with the search



the example above is borrowed from "AI: A Modern Approach" by S. RussellL & P. Norvig

Planning via Skeletonization

- Sampling-based planning:
  - typically provides probabilistic completeness guarantees (guaranteed to find a solution, if one exists, in the limit of the number of samples)
  - in many domains, is much faster and requires much less memory
  - well-suited for high-dimensional planning

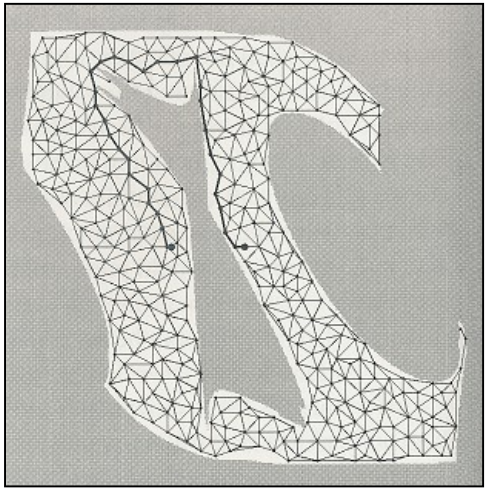


the example above is borrowed from "AI: A Modern Approach" by S. RussellL & P. Norvig

Probabilistic Roadmaps (PRMs) [Kavraki et al. '96]

Step 1. Preprocessing Phase: Build a roadmap (graph) $\mathcal{G}$ which, hopefully, should be accessible from any point in C_{free}

Step 2. Query Phase: Given a start configuration q_I and goal configuration q_G, connect them to the roadmap $\mathcal{G}$ using a local planner, and then search the augmented roadmap for a shortest path from q_I to q_G

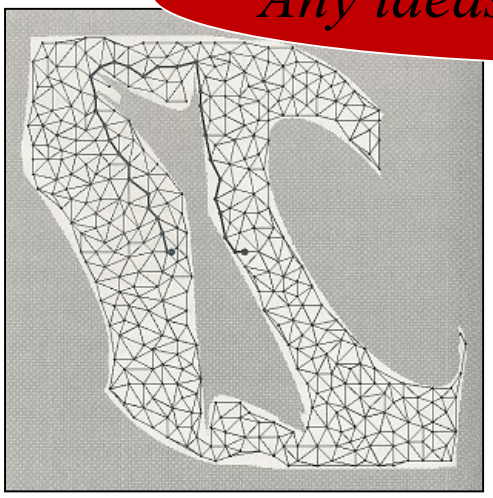


Probabilistic Roadmaps (PRMs) [Kavraki et al. '96]

Step 1. Preprocessing Phase: Build a roadmap (graph) $\mathcal{G}$ which, hopefully, should be accessible from any point in C_{free}

Step 2. Query Phase: Given a start configuration q_I and goal configuration q_G, connect them to the roadmap $\mathcal{G}$ using a local planner, and then search the augmented roadmap for a shortest path from q_I to q_G

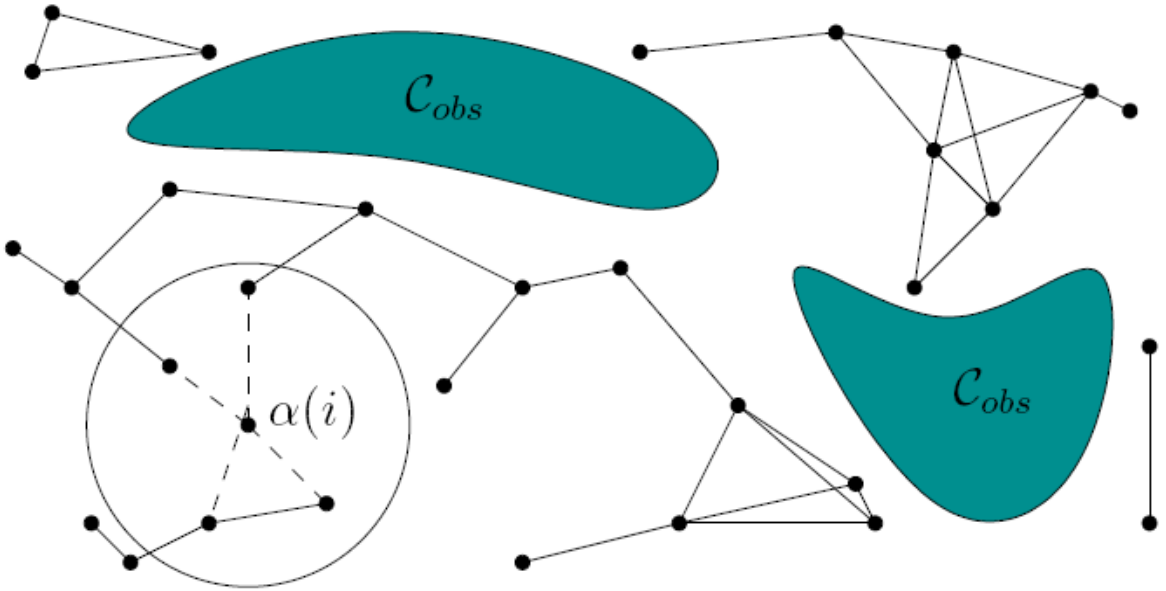
Any ideas for the local planner?



Probabilistic Roadmaps (PRMs) [Kavraki et al. '96]

Step 1: Preprocessing Phase.

```
BUILD_ROADMAP
 1  G.init(); i ← 0;
 2  while i < N
 3      if α(i) ∈ C_free then
 4          G.add_vertex(α(i)); i ← i + 1;
 5          for each q ∈ NEIGHBORHOOD(α(i),G)
 6              if ((not G.same_component(α(i), q)) and CONNECT(α(i), q)) then
 7                  G.add_edge(α(i), q);
```



borrowed from "Planning Algorithms" by S. LaValle

Probabilistic Roadmaps (PRMs) [Kavraki et al. '96]

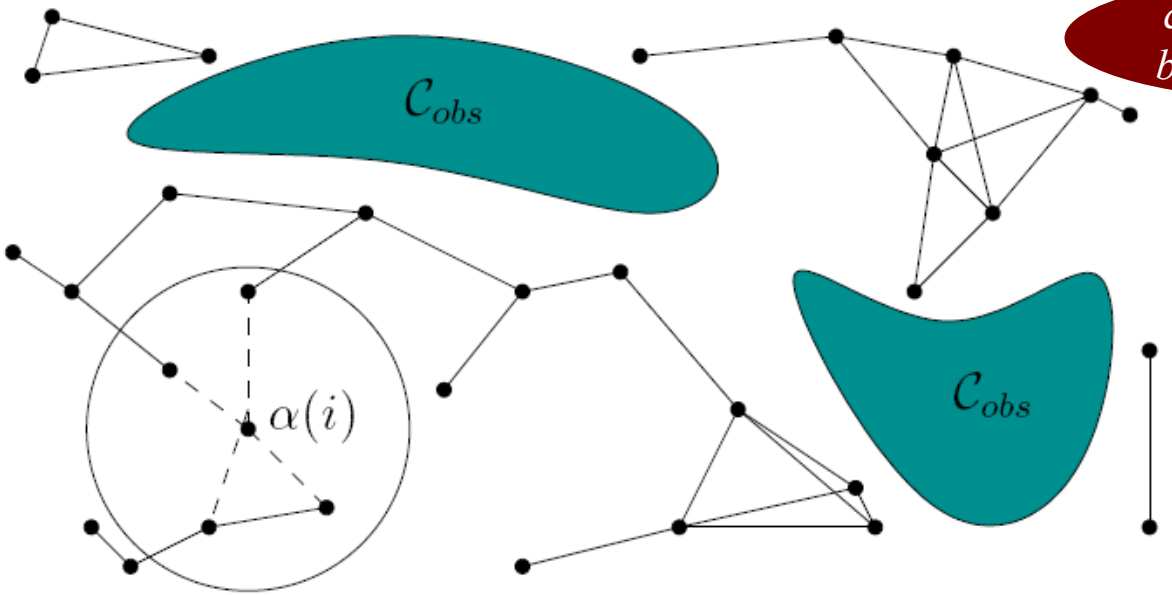
Step 1: Preprocessing Phase.

```
BUILD_ROADMAP
1  G.init(); i ← 0;
2  while i < N
3      if α(i) ∈ C_free then
4          G.add_vertex(α(i)); i ← i + 1;
5          for each q ∈ NEIGHBORHOOD(α(i),G)
6              if ((not G.same_component(α(i), q)) and CONNECT(α(i), q)) then
7                  G.add_edge(α(i), q);
```

new i^{th} configuration sample

some region around the configuration sample

can be connected by a local planner



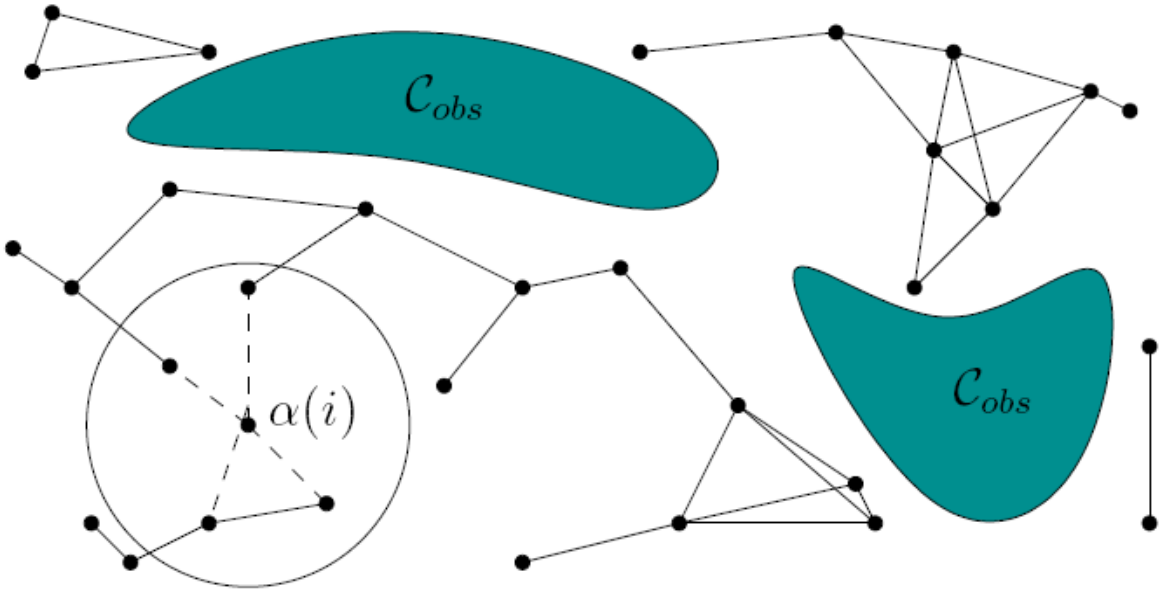
borrowed from "Planning Algorithms" by S. LaValle

Probabilistic Roadmaps (PRMs) [Kavraki et al. '96]

Step 1: Preprocessing Phase.

```
BUILD_ROADMAP
1  G.init(); i ← 0;
2  while i < N
3      if α(i) ∈ C_free then
4          G.add_vertex(α(i)); i ← i + 1;
5          for each q ∈ NEIGHBORHOOD(α(i),G)
6              if ((not G.same_component(α(i), q)) and CONNECT(α(i), q)) then
7                  G.add_edge(α(i), q);
```

can be replaced with: "number of successors of q < K"



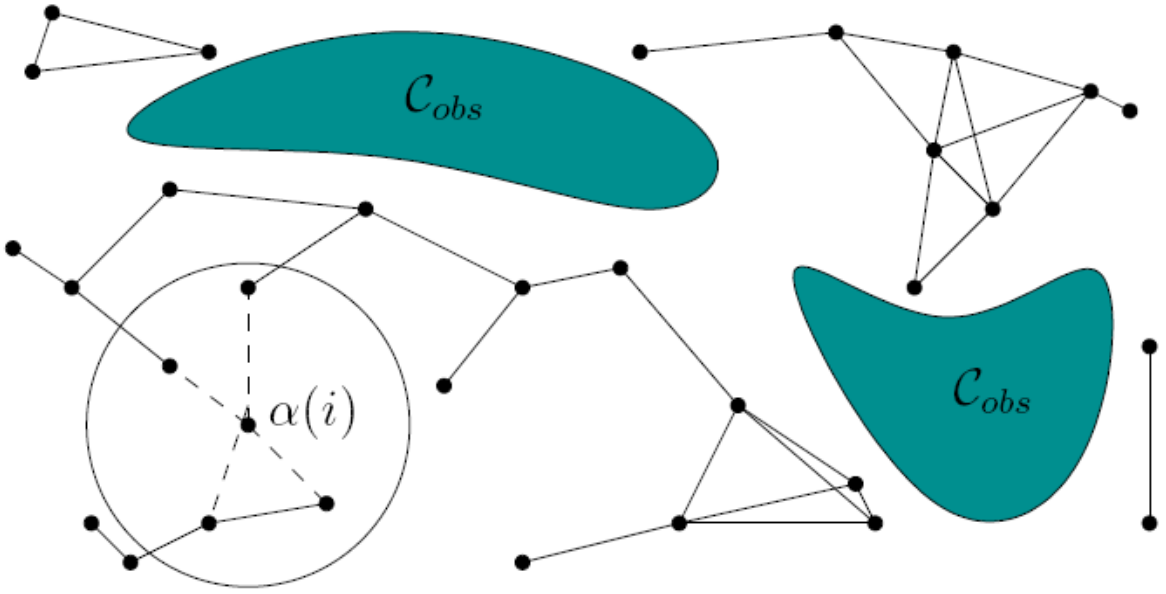
borrowed from "Planning Algorithms" by S. LaValle

Probabilistic Roadmaps (PRMs) [Kavraki et al. '96]

Step 1: Preprocessing Phase.

Efficient implementation of $q \in$ NEIGHBORHOOD$(\alpha(i), \mathcal{G})$

- select K vertices closest to $\alpha(i)$
- select K (often just 1) closest points from each of the components in $\mathcal{G}$
- select all vertices within radius r from $\alpha(i)$



borrowed from "Planning Algorithms" by S. LaValle

Probabilistic Roadmaps (PRMs) [Kavraki et al. '96]

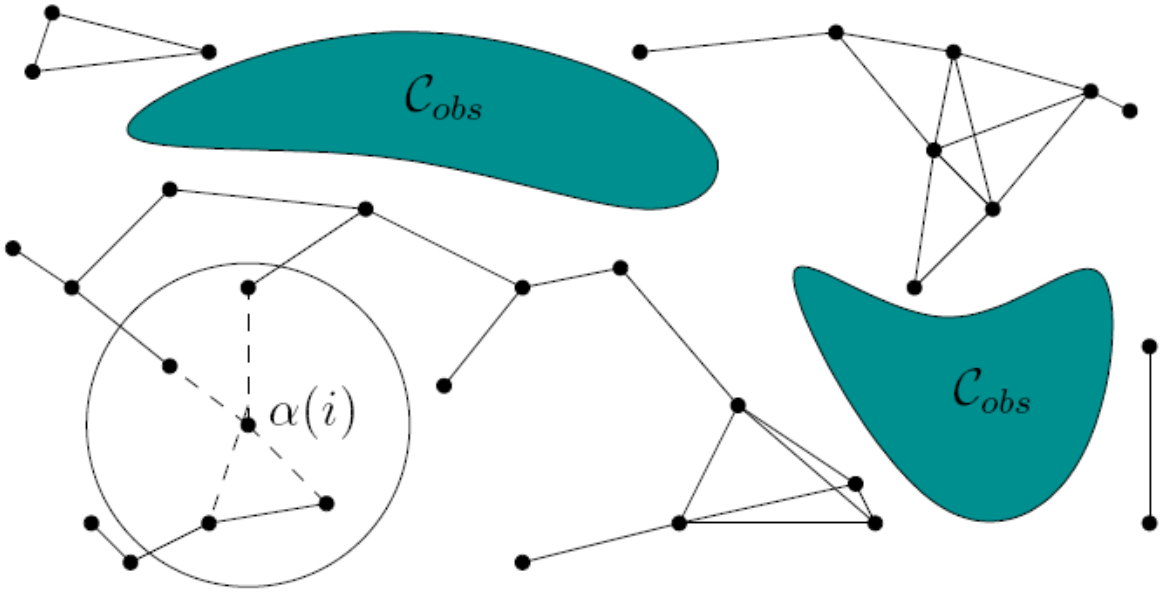
Step 1: Preprocessing Phase.

Sampling strategies

- sample uniformly from C_{free}

- select at random an existing vertex with a probability distribution inversely proportional to how well-connected a vertex is, and then generate a random motion from it to get a sample $\alpha(i)$

- bias sampling towards obstacle boundaries



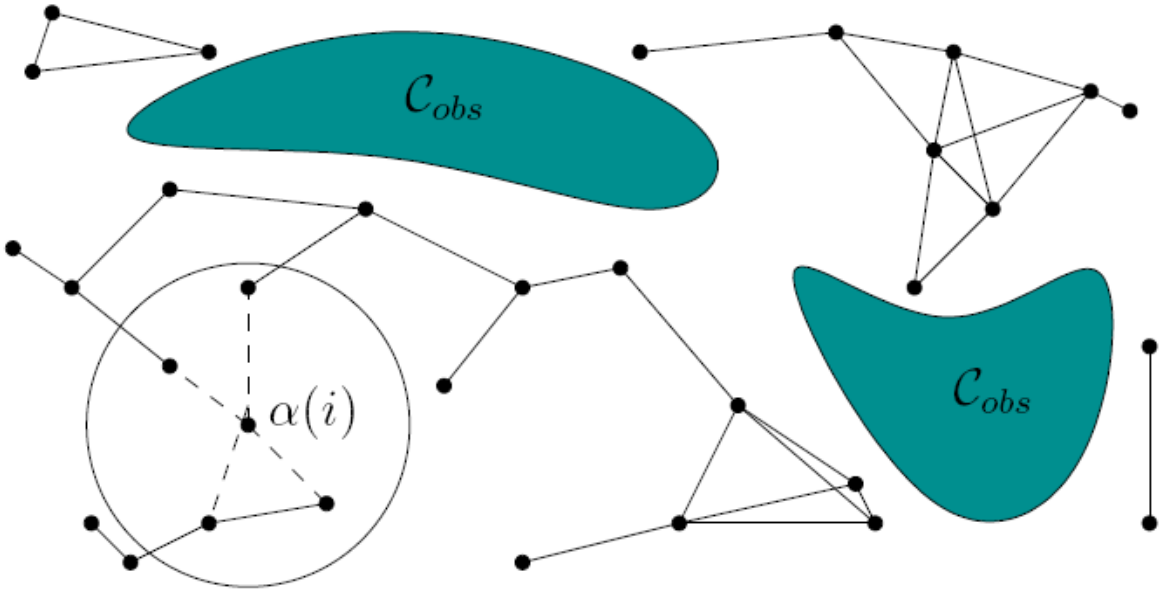
borrowed from "Planning Algorithms" by S. LaValle

Probabilistic Roadmaps (PRMs) [Kavraki et al. '96]

Step 1: Preprocessing Phase.

Sampling strategies

- sample q_1 and q_2 from Gaussian around q_1 and if either is in C_{obs}, then the other one is set as $\alpha(i)$
- sample q_1, q_2 , q_3 and set q_2 as $\alpha(i)$ if q_2 is in C_{free}, and q_1 and q_3 are in C_{obs}
- bias sampling away from obstacles



borrowed from "Planning Algorithms" by S. LaValle

Summary

- Graph constructions
  - Resolution complete methods
    - N-dimensional grids
    - Lattice-based graphs
  - Skeletonization methods
    - Visibility graphs
    - Voronoi diagrams
    - Probabilistic Roadmaps

- Methods for searching the graph – in later classes

- Interleaving the above two steps is critical