16-782
Planning & Decision-making in Robotics

Planning Representations:
Implicit vs. Explicit Graphs;
Skeletonization, cell decomposition, lattices

Maxim Likhachev

Robotics Institute

Carnegie Mellon University

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

This class

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 **Explicit Graph** (allocated prior to the search itself):

1. *Create the graph G = {V, E} in-memory*

2. *Search the graph*

Using Explicit Graphs
is typical for low-D (i.e., 2D) problems in Robotics
(with the exception of PRMs, covered in a later lecture)

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

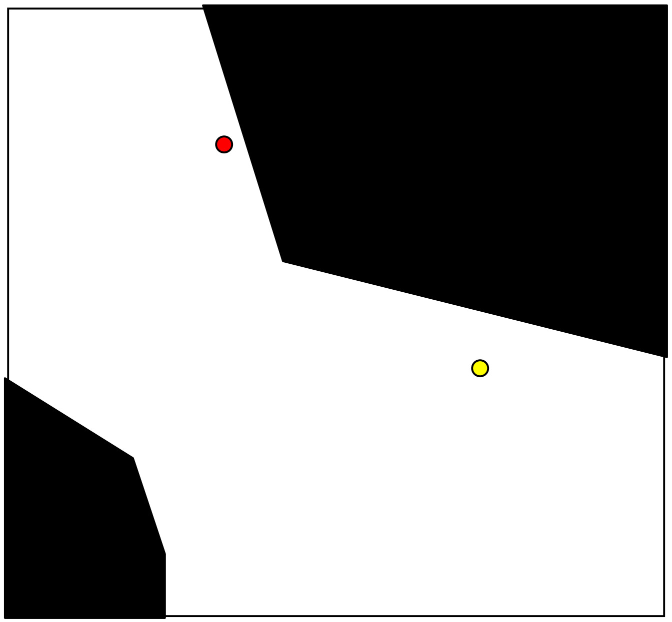
Using Implicit Graphs
is critical for most (>2D) problems
in Robotics

2D Planning for Omnidirectional Point Robot

Planning for omnidirectional point robot:

What is $M^R = \langle x,y \rangle$
What is $M^W = \langle$*obstacle/free space*$\rangle$
What is $s^R_{current} = \langle x_{current}, y_{current} \rangle$
What is $s^W_{current}$ = *constant*
What is C = *Euclidean Distance*
What is $G = \langle x_{goal}, y_{goal} \rangle$

Any ideas on how to construct a graph for planning?



Two Classes of Graph Construction Methods

- Skeletonization

  -Visibility graphs

  -Voronoi diagrams

  - Probabilistic roadmaps

- Cell decomposition

  - X-connected grids

  - lattice-based graphs

Two Classes of Graph Construction Methods

- Skeletonization

  -Visibility graphs

  -Voronoi diagrams

  - Probabilistic roadmaps

Will be covered in later classes

- Cell decomposition

  - X-connected grids

  - lattice-based graphs

Two Classes of Graph Construction Methods

- Skeletonization

  -Visibility graphs

  -Voronoi diagrams

  - Probabilistic roadmaps

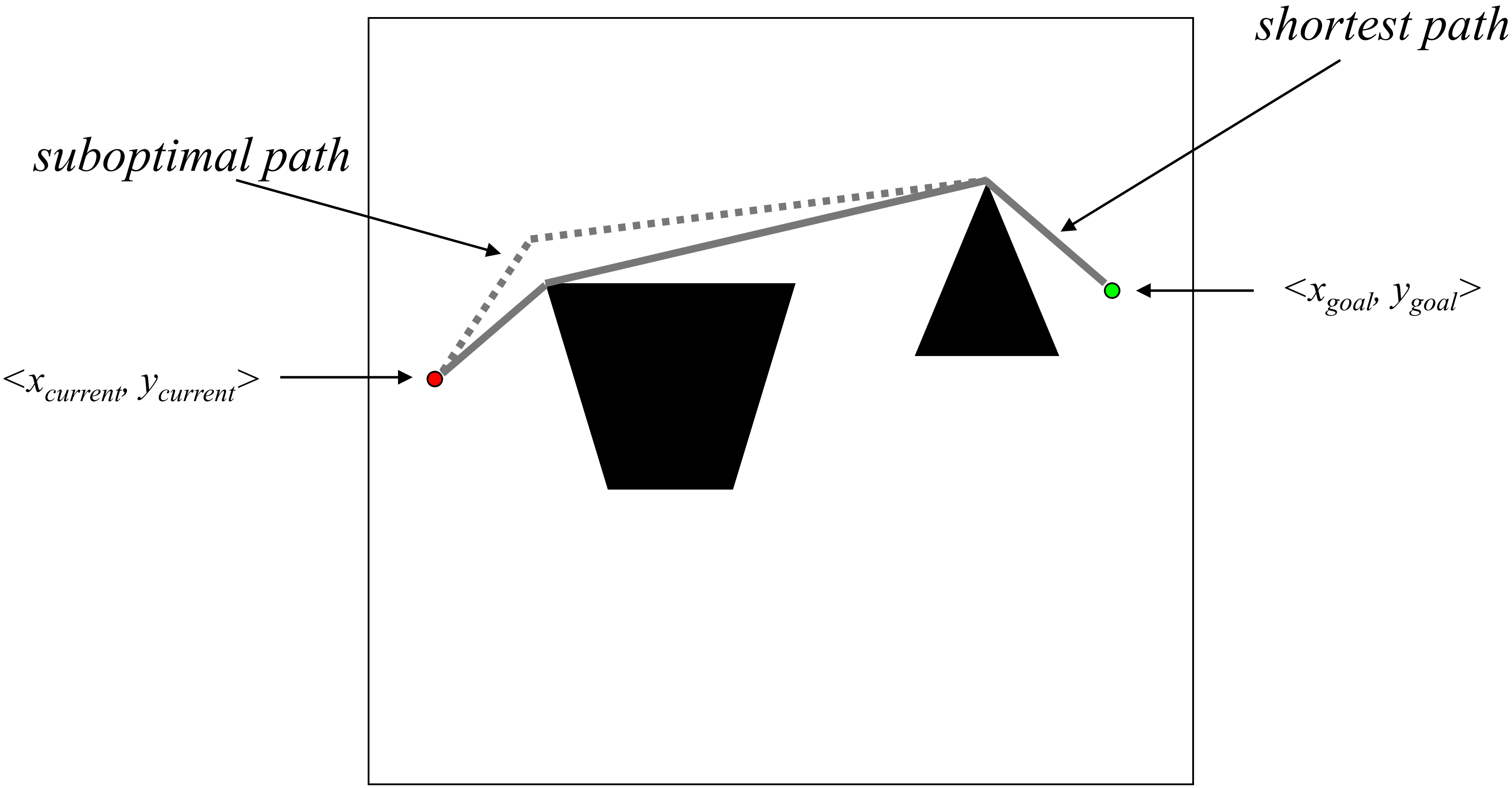
- Cell decomposition

  - X-connected grids

  - lattice-based graphs

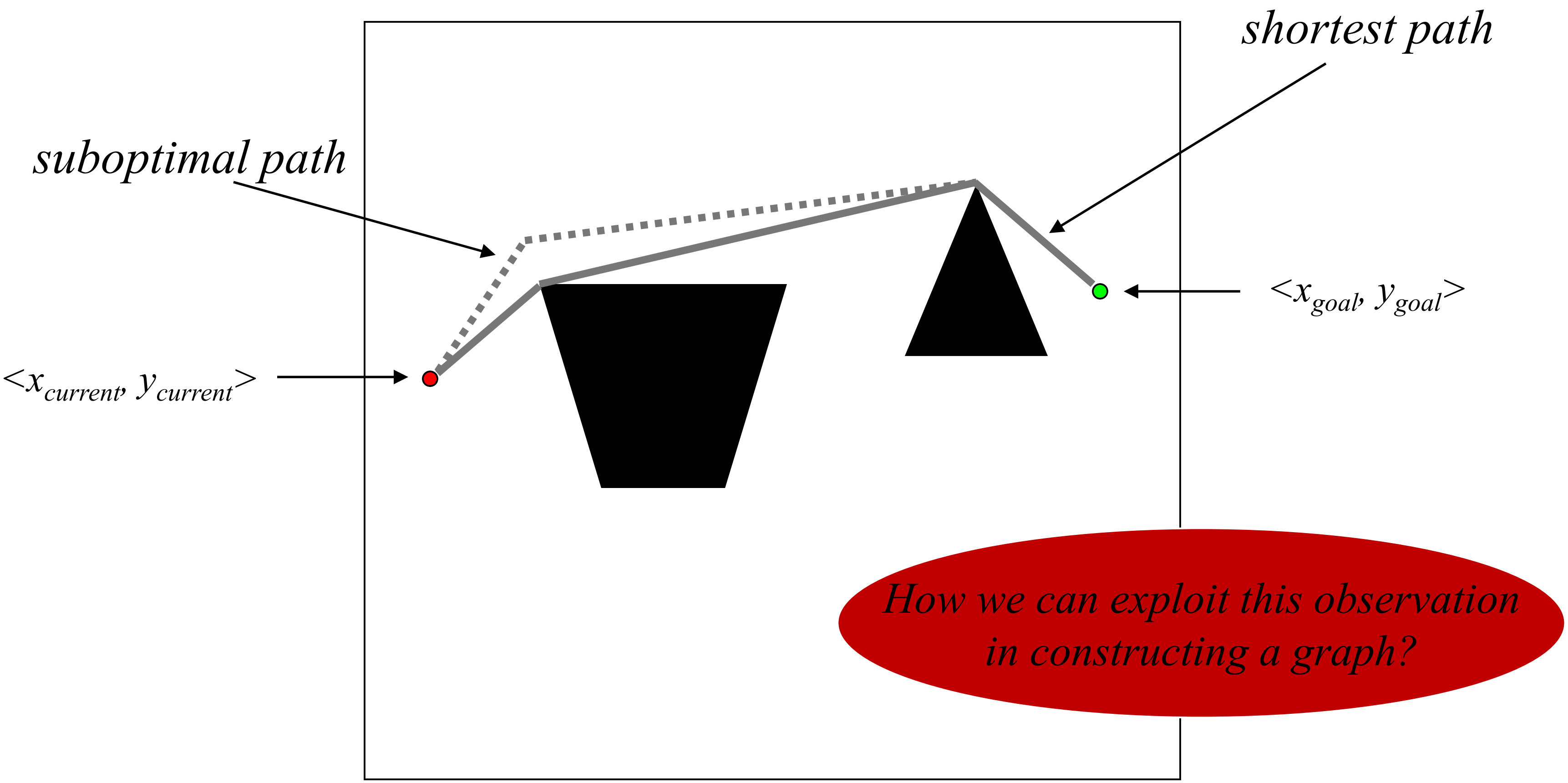
Skeletonization-based Graphs

- 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*



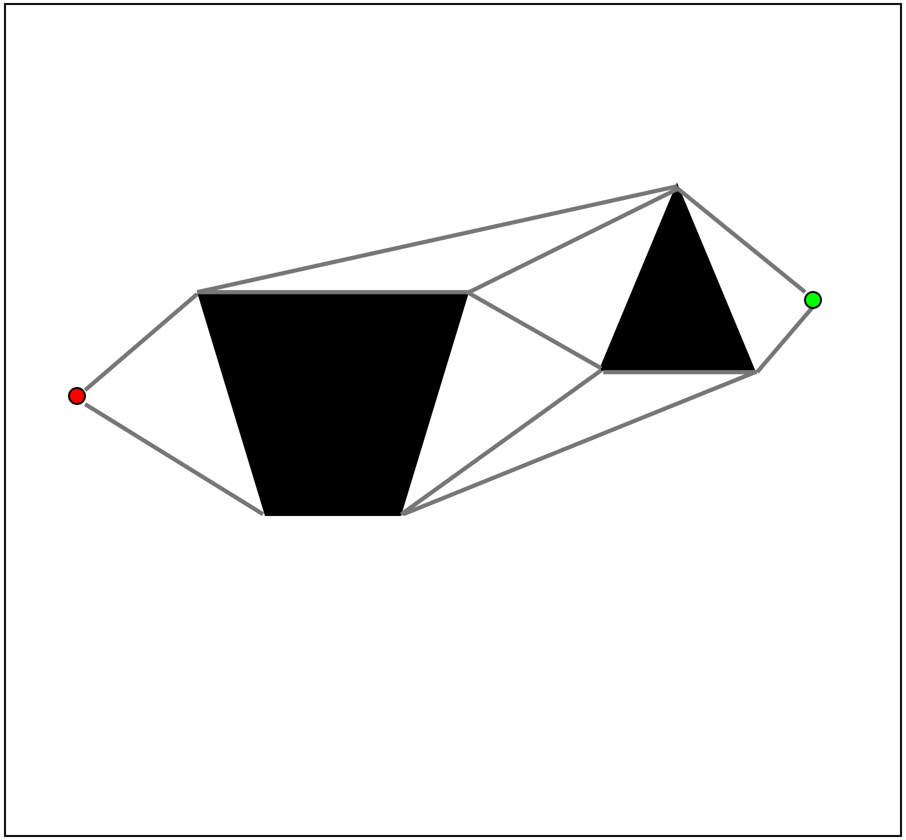
Skeletonization-based Graphs

- 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*



Skeletonization-based Graphs

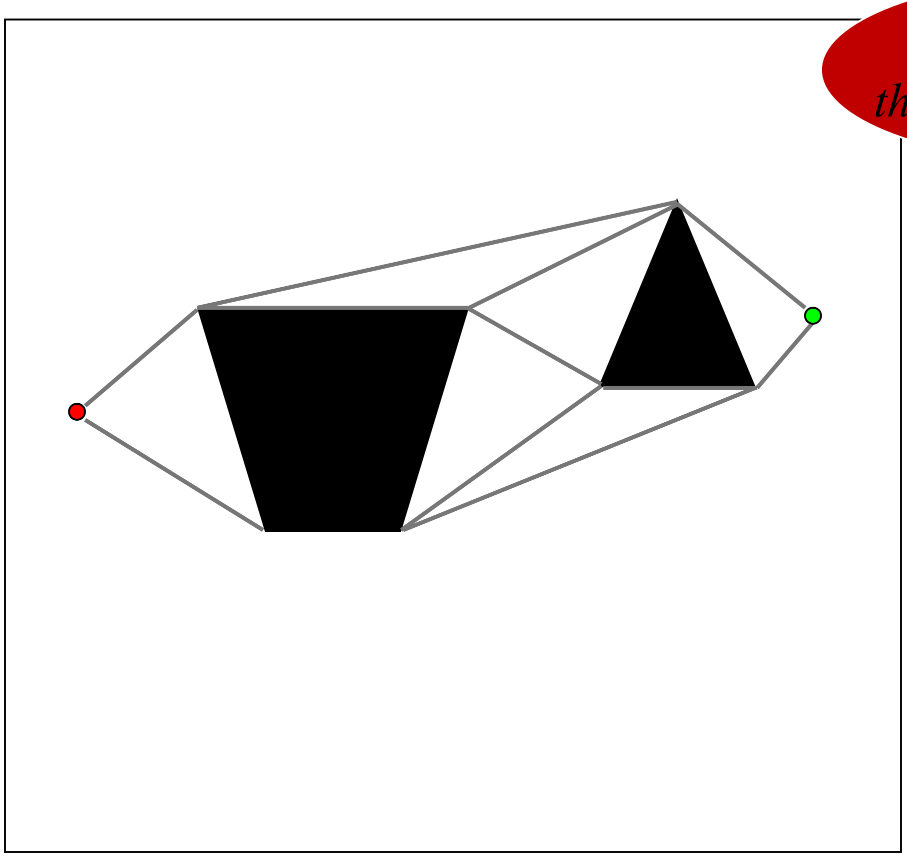
- Visibility Graphs [Wesley & Lozano-Perez '79]
  - construct a graph by connecting all vertices, start and goal by obstacle-free straight line segments (graph is $O(n^2)$, where n - # of vert.)



Skeletonization-based Graphs

- 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.)



Disadvantages of the Visibility Graphs?

Skeletonization-based Graphs

- Visibility Graphs
  - advantages:
    - independent of the size of the environment
  - disadvantages:
    - path is too close to obstacles
    - hard to deal with the cost function that is not distance
    - hard to deal with non-polygonal obstacles
    - hard to maintain the polygonal representation of obstacles
    - can be expensive in spaces higher than 2D

Two Classes of Graph Construction Methods

- Skeletonization

  -Visibility graphs

  -Voronoi diagrams

  - Probabilistic roadmaps

- Cell decomposition

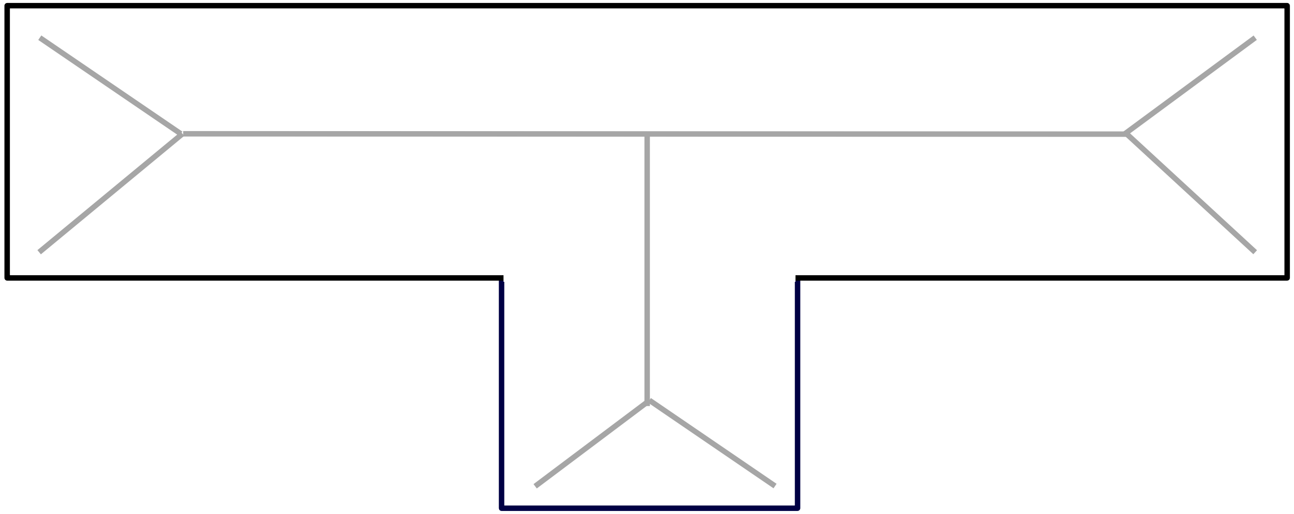
  - X-connected grids

  - lattice-based graphs

Skeletonization-based Graphs

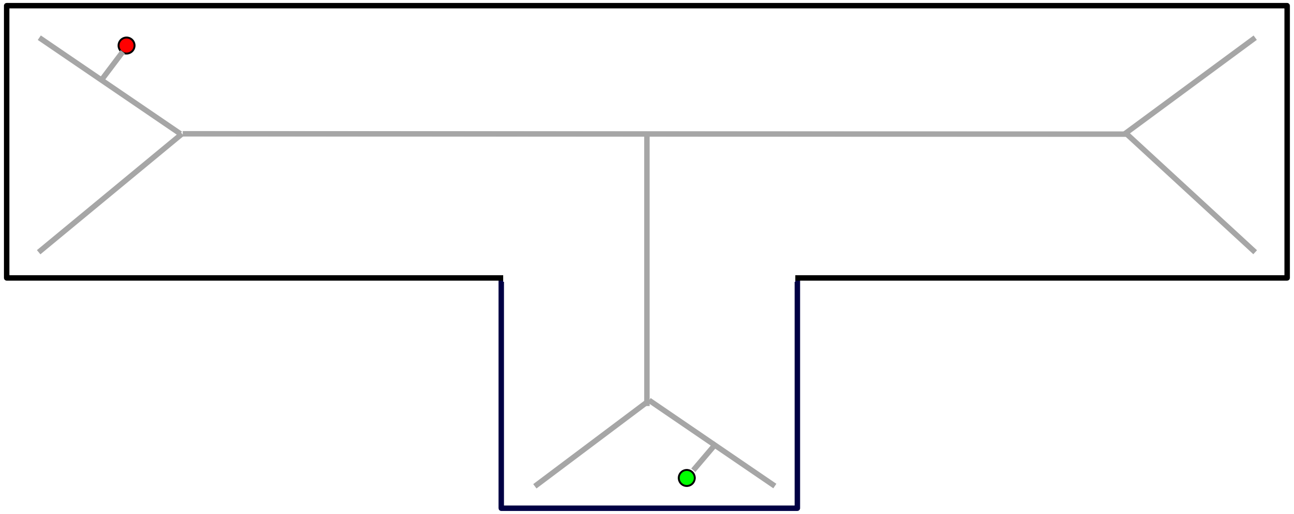
- Voronoi diagram [Rowat '79]
  - set of all points that are equidistant to two nearest obstacles

  (can be computed O (n log n), where n - # of points that represent obstacles)



Skeletonization-based Graphs

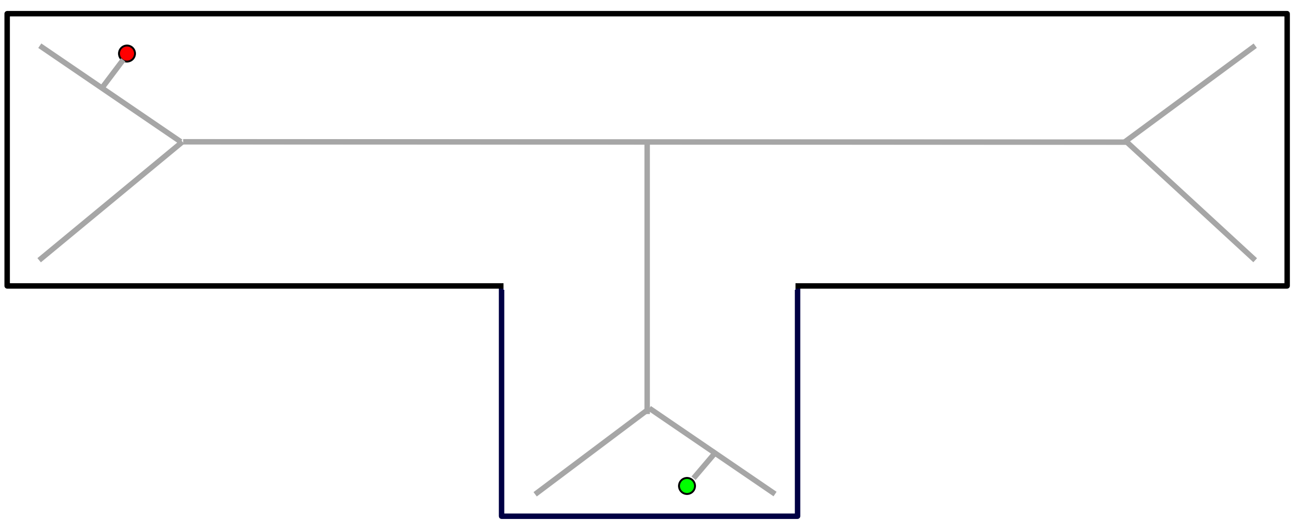
- Voronoi diagram-based graph
  - Edges: Boundaries in Voronoi diagram
  - Vertices: Intersection of boundaries
  - Add start and goal vertices
  - Add edges that correspond to:
    - shortest path segment from start to the nearest segment on the Voronoi diagram
    - shortest path segment from goal to the nearest segment on the Voronoi diagram



Skeletonization-based Graphs

- Voronoi diagram-based graph
  - Edges: Boundaries in Voronoi diagram
  - Vertices: Intersection of boundaries
  - Add start and goal vertices
  - Add edges that correspond to:
    - shortest path segment from start to the nearest segment on the Voronoi diagram
    - shortest path segment from goal to the nearest segment on the Voronoi diagram

Disadvantages of the Voronoi diagram-based Graphs?



Skeletonization-based Graphs

- Voronoi diagram-based graph
  - advantages:
    - tends to stay away from obstacles
    - independent of the size of the environment
    - can work with any obstacles represented as set of points
  - disadvantages:
    - can result in highly suboptimal paths
    - hard to deal with the cost function that is not distance
    - hard to use/maintain beyond 2D

Two Classes of Graph Construction Methods

- Skeletonization

  -Visibility graphs

  -Voronoi diagrams

  - Probabilistic roadmaps

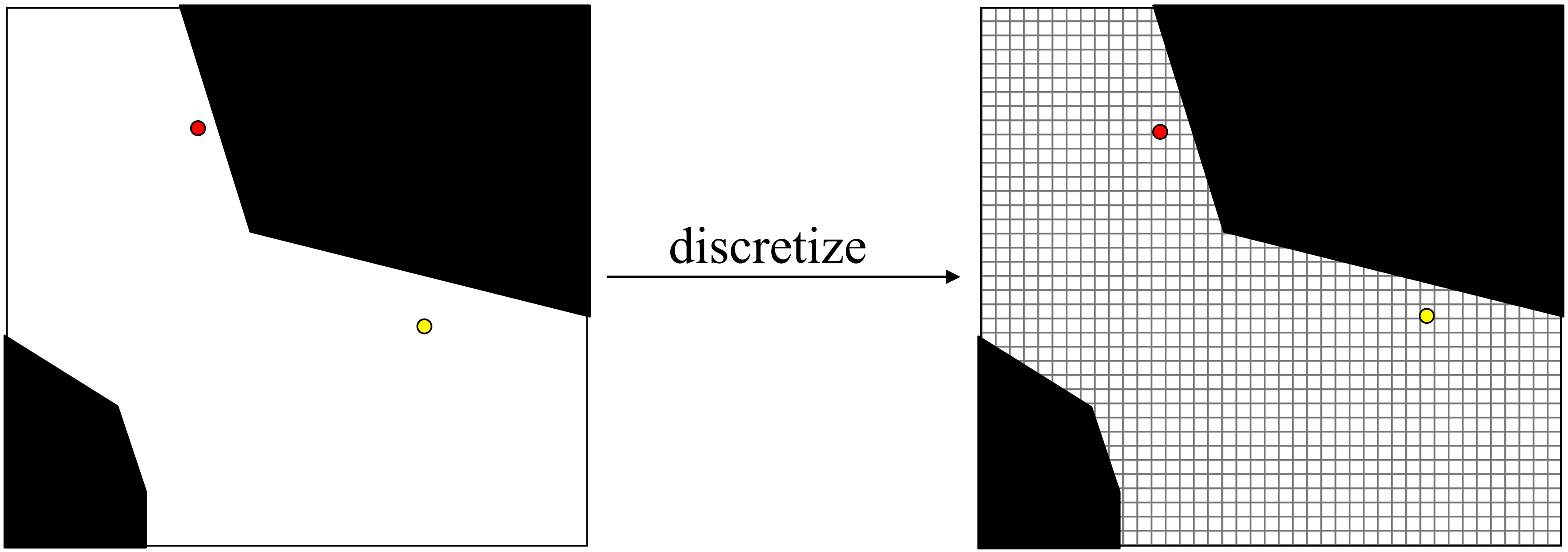
- Cell decomposition

  - X-connected grids

  - lattice-based graphs

Grid-based Graphs

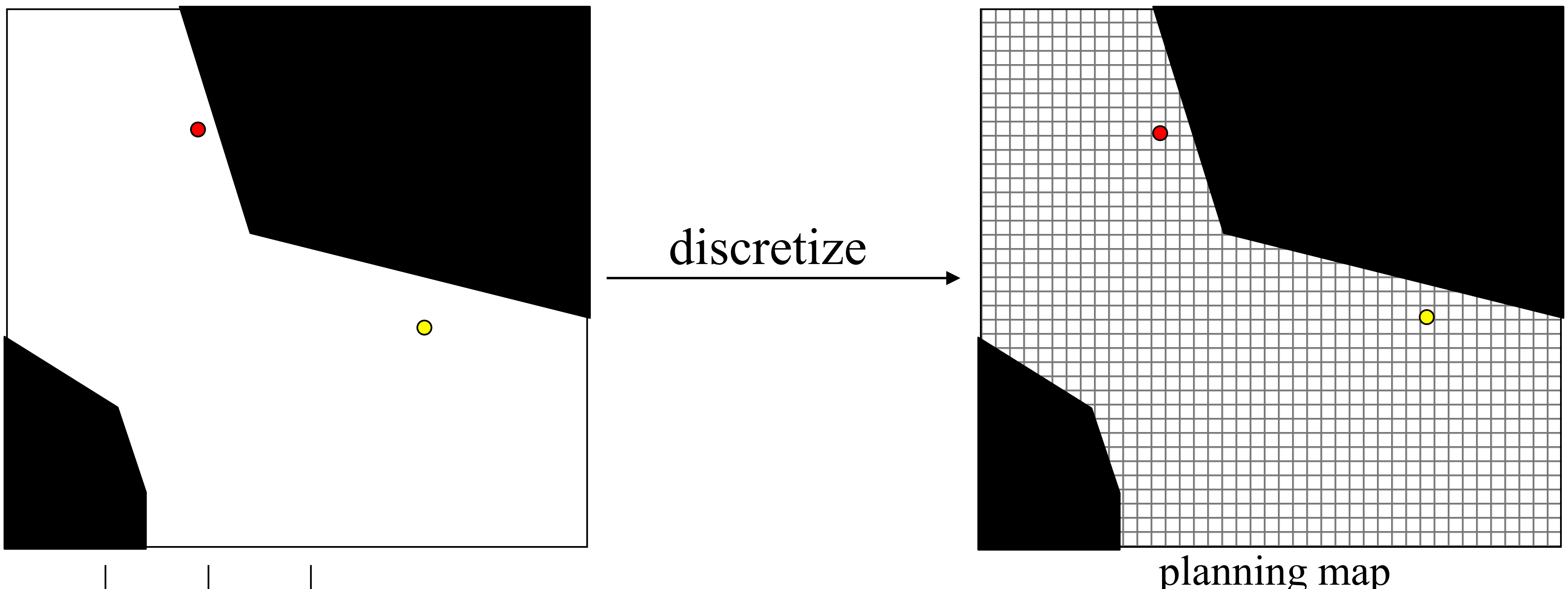
- Approximate Cell Decomposition:
  - overlay uniform grid (discretize)

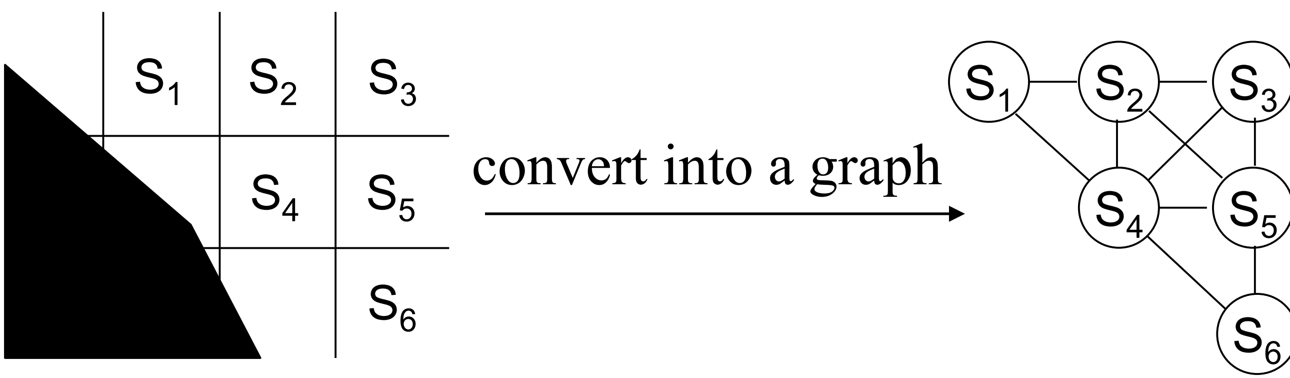


planning map

Grid-based Graphs

- Approximate Cell Decomposition:
  - construct a graph

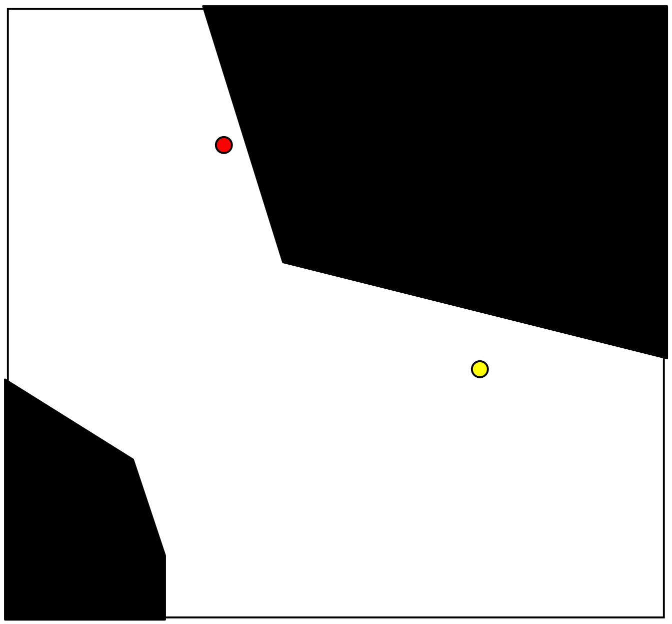




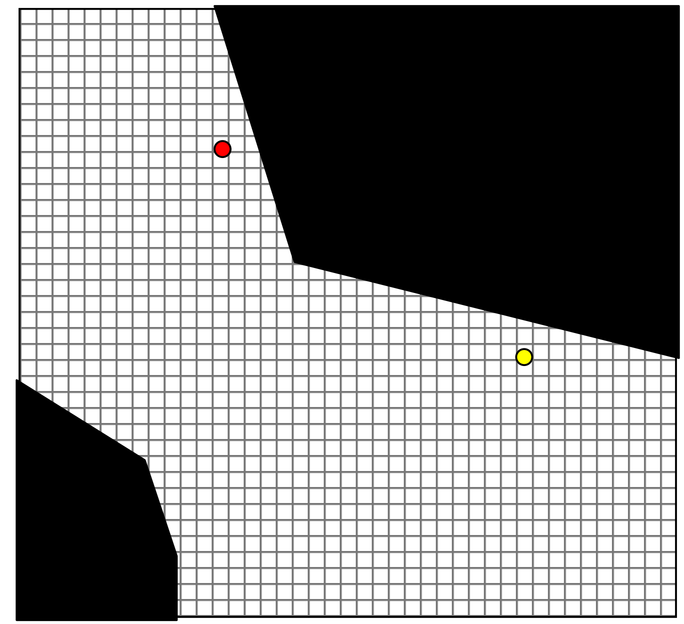
search the graph for a least-cost path from s_{start} to s_{goal}

Grid-based Graphs

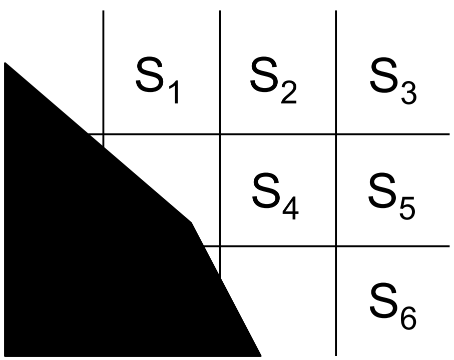
- Approximate Cell Decomposition:
  - construct a graph



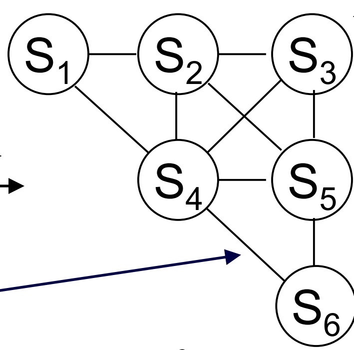
discretize



planning map



convert into a graph

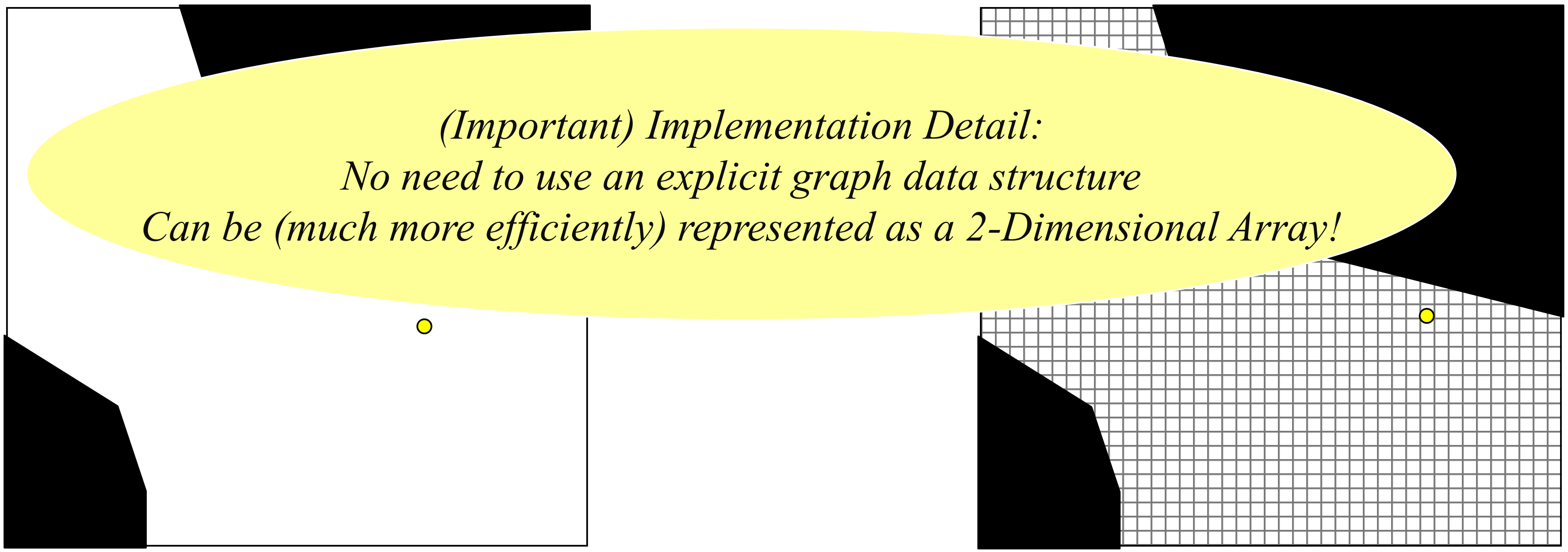


search the graph for a least-cost path from s_{start} to s_{goal}

edgecosts can represent ***any*** *cost function*

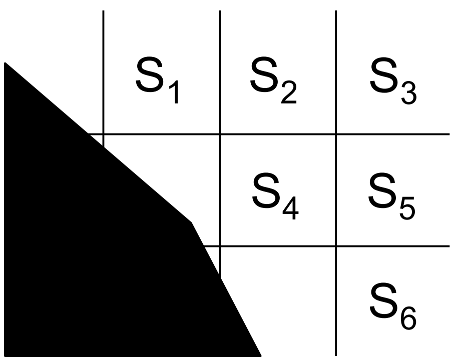
Grid-based Graphs

- Approximate Cell Decomposition:
  - construct a graph

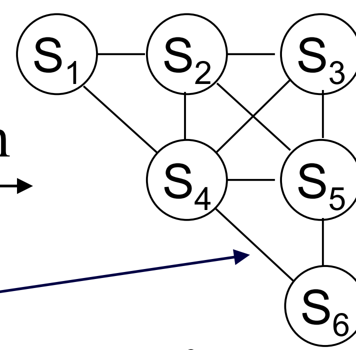


(Important) Implementation Detail:
No need to use an explicit graph data structure
Can be (much more efficiently) represented as a 2-Dimensional Array!

planning map



convert into a graph

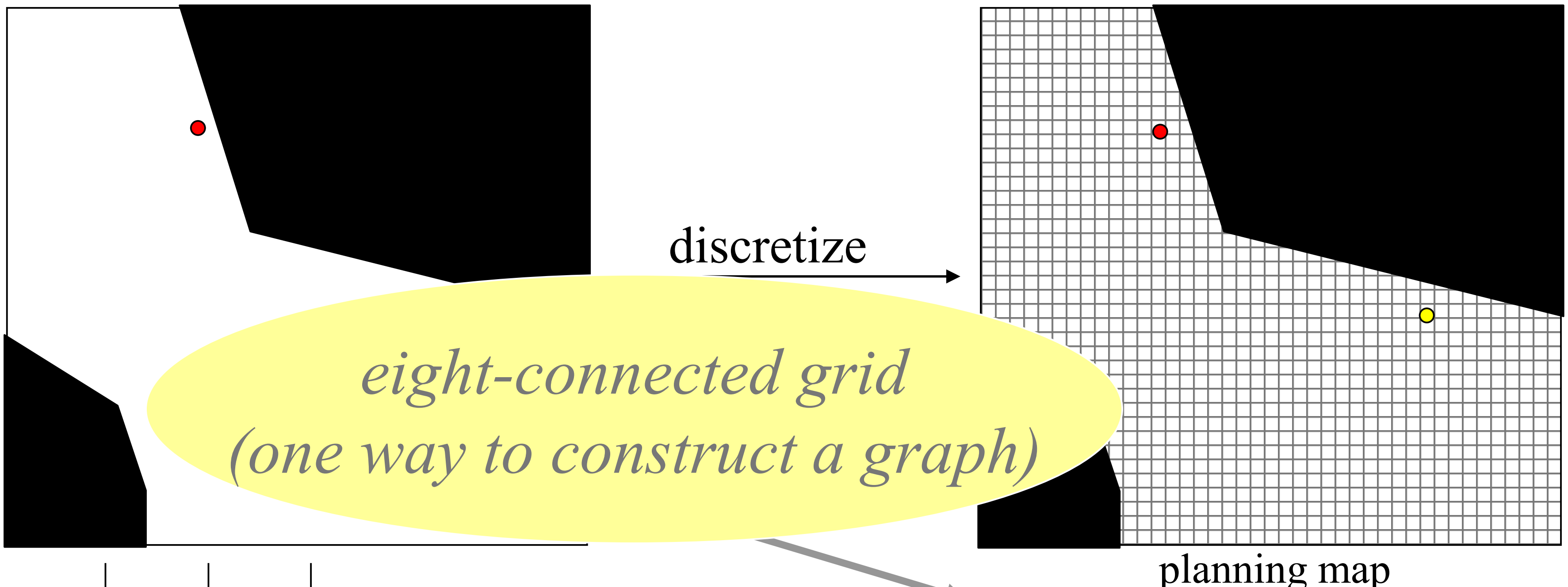


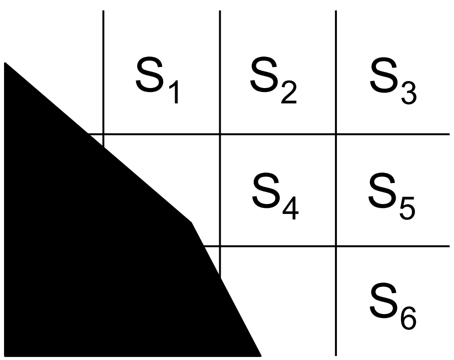
search the graph for a least-cost path from s_{start} to s_{goal}

*edgecosts can represent **any** cost function*

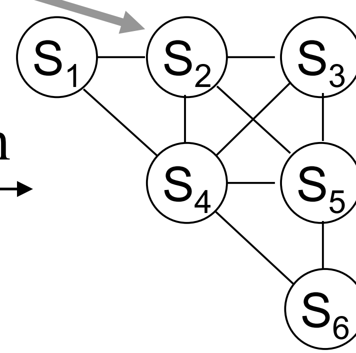
Grid-based Graphs

- Approximate Cell Decomposition:
  - construct a graph





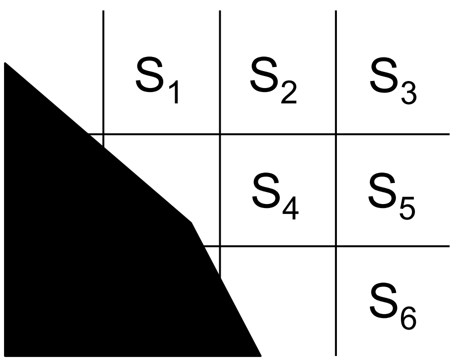
convert into a graph



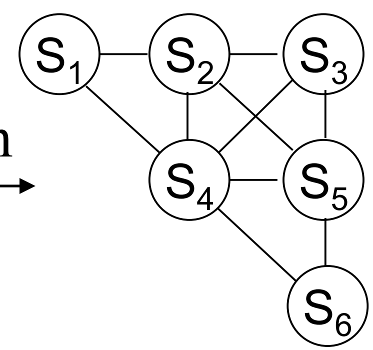
search the graph for a least-cost path from s_{start} to s_{goal}

Grid-based Graphs

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



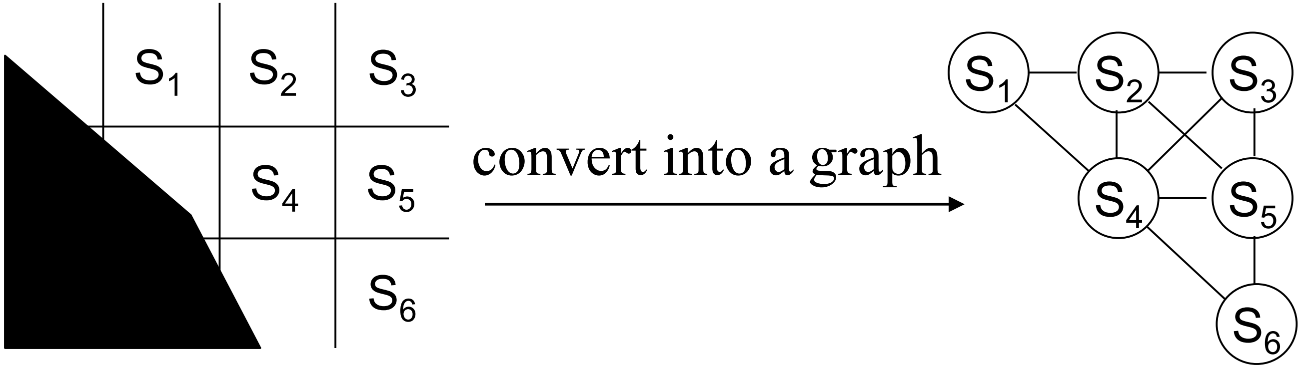
convert into a graph



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

Grid-based Graphs

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

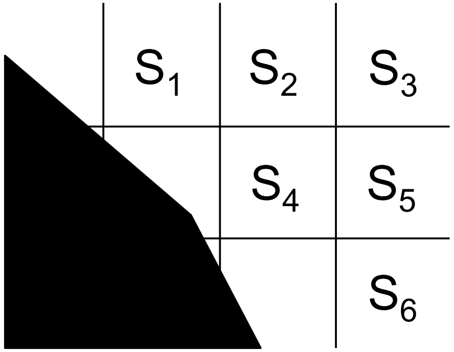


search the graph for a least-cost path from s_{start} to s_{goal}

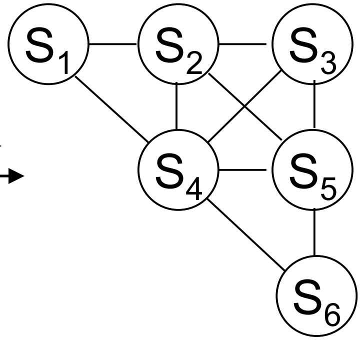
Grid-based Graphs

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

so, what's the solution?



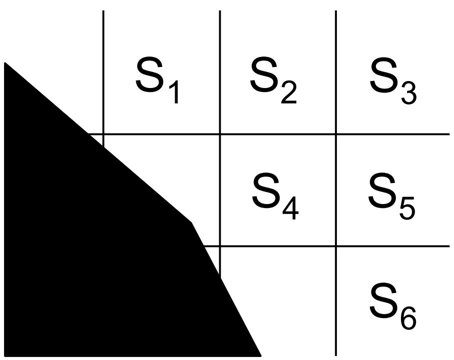
convert into a graph



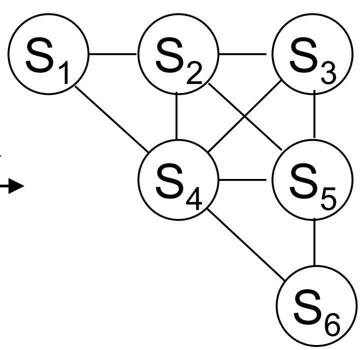
search the graph for a least-cost path from s_{start} to s_{goal}

Grid-based Graphs

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



convert into a graph

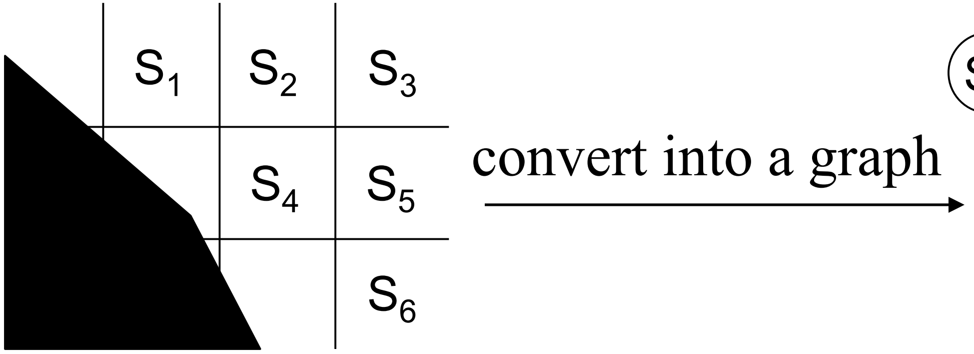


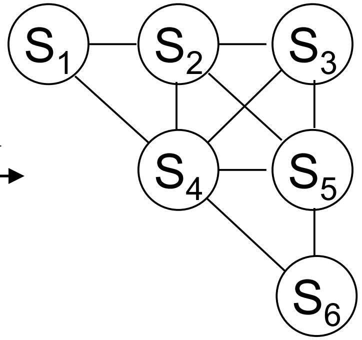
search the graph for a least-cost path from s_{start} to s_{goal}

Grid-based Graphs

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

Any ideas?



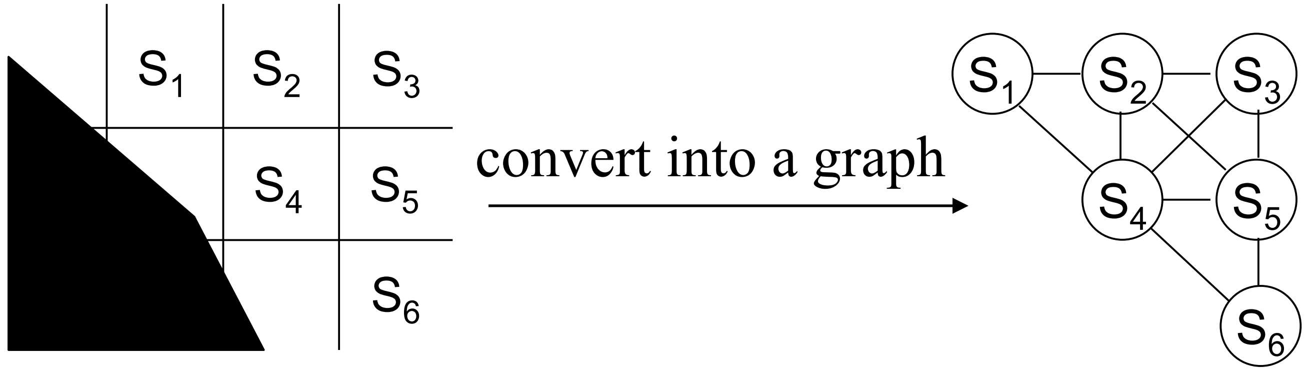


search the graph for a least-cost path from s_{start} to s_{goal}

Grid-based Graphs

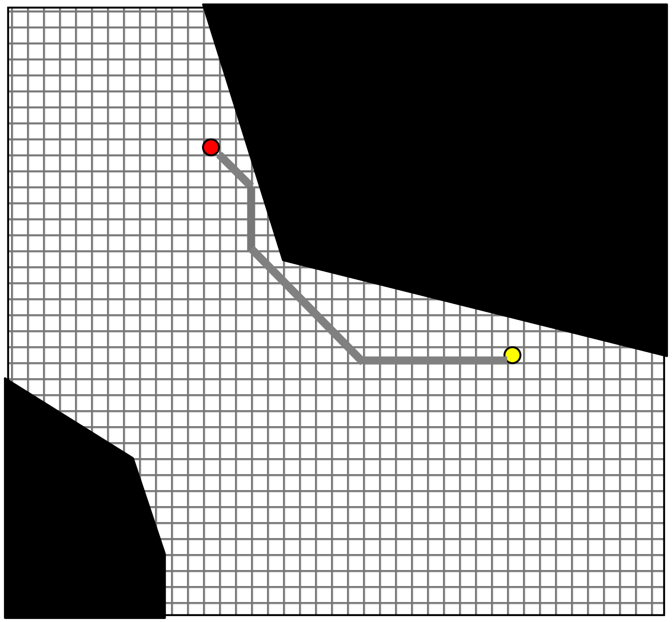
- Graph construction:
  - connect neighbors

8-connected grid



Grid-based Graphs

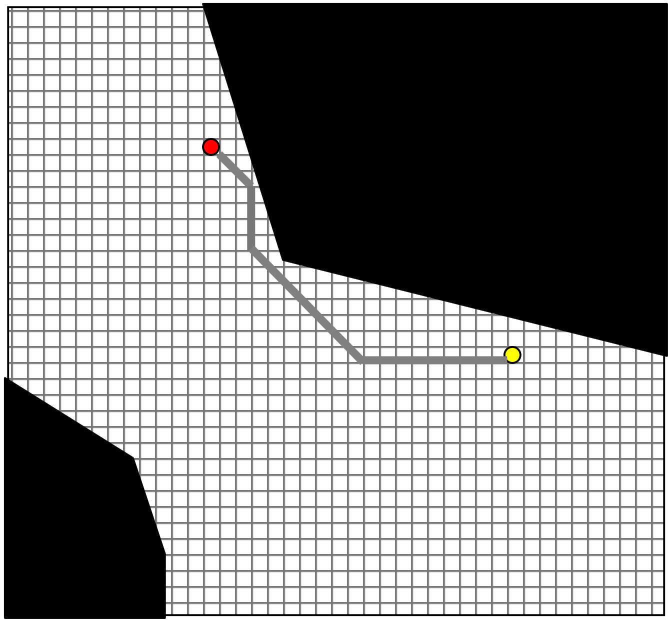
- Graph construction:
  - connect neighbors
  - path is restricted to 45^o degrees



Grid-based Graphs

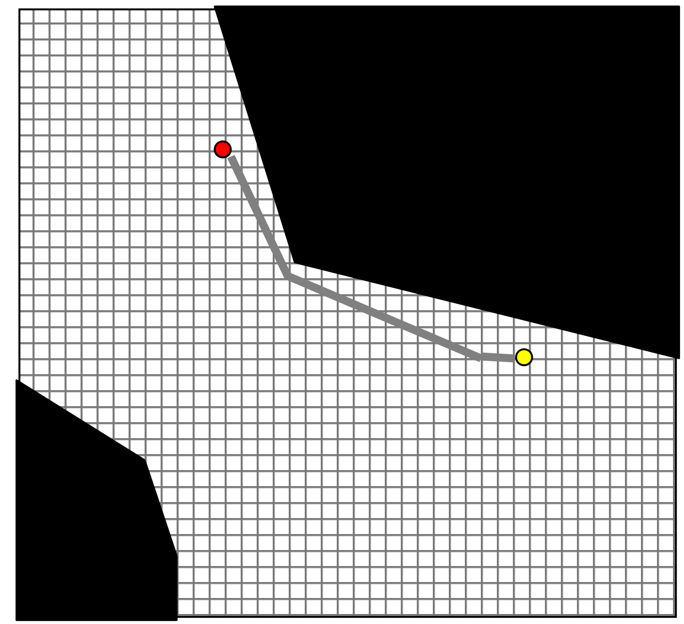
- Graph construction:
  - connect neighbors
  - path is restricted to 45^{o} degrees

Ideas to improve it?

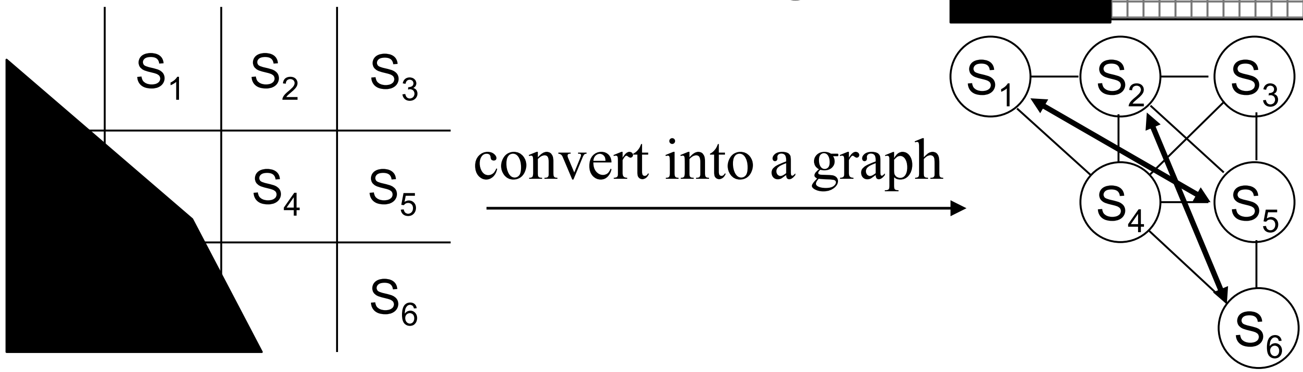


Grid-based Graphs

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

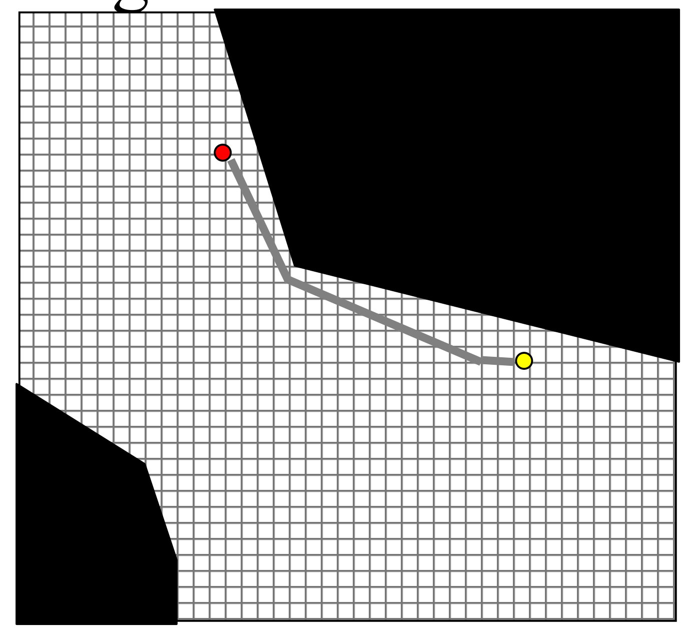


***16**-connected grid*

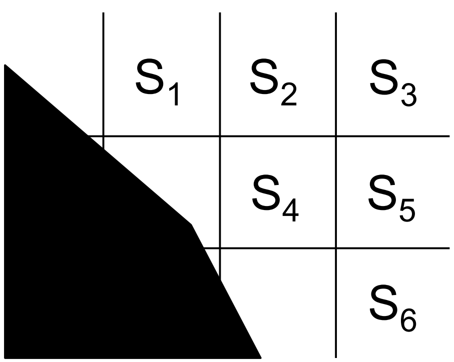


Grid-based Graphs

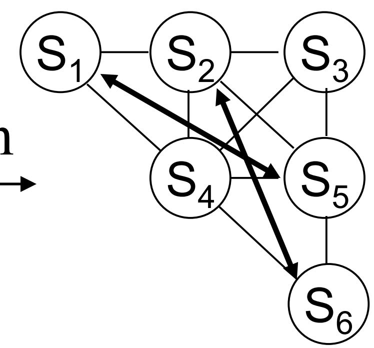
- Graph construction:
  - connect cells to neighbor of neighbors
  - path is restricted to **26.6º/63.4º** degrees



***16**-connected grid*



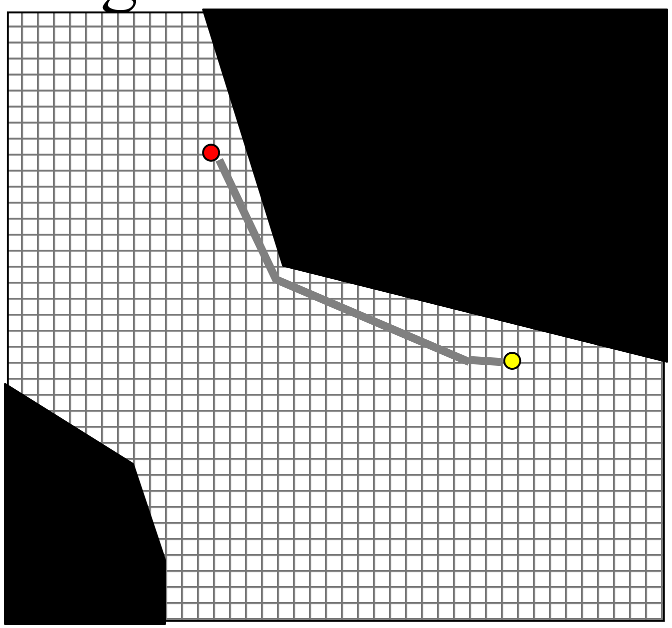
convert into a graph



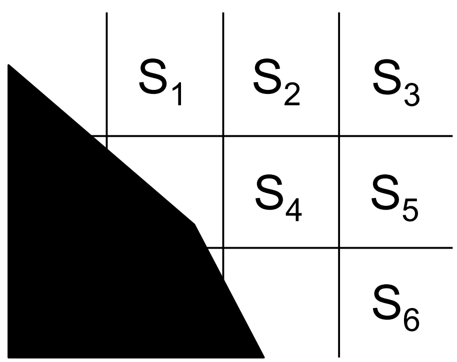
Grid-based Graphs

- Graph construction:
  - connect cells to neighbor of neighbors
  - path is restricted to **26.6º/63.4º** degrees

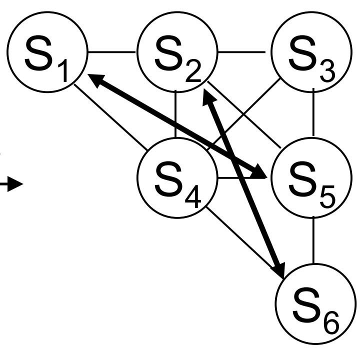
Disadvantages?



***16**-connected grid*



convert into a graph

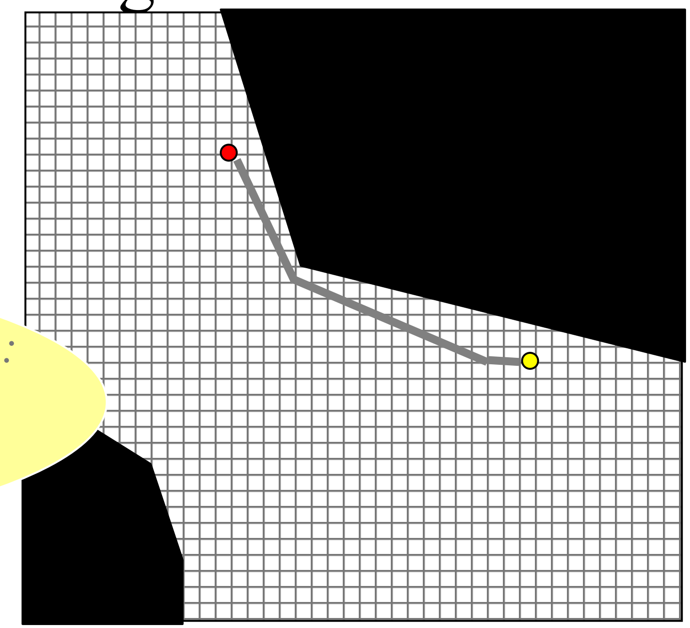


Grid-based Graphs

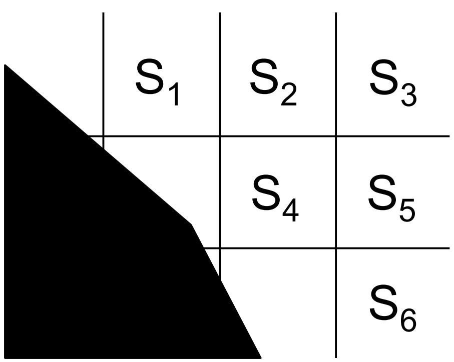
- Graph construction:
  - connect cells to neighbor of neighbors
  - path is restricted to **26.6º/63.4º** degrees

Disadvantages?

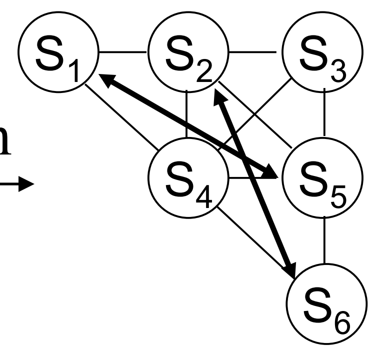
Dynamically generated directions (for low-d problems):
Field D [Ferguson & Stentz, '06],*
Theta [Nash & Koenig, '13]*



16-connected grid



convert into a graph



Cell Decomposition-based Graphs

- Grid-based graph
  - advantages:
    - very simple to implement (super popular)
    - can represent any dimensional space
    - works well with obstacles represented as set of points
    - works with any cost function
  - disadvantages:
    - size does depend on the size of the environment
    - can be expensive to compute/store if # of dimensions > 3

Cell Decomposition-based Graphs

- Grid-based graph
  - advantages:
    - very simple to implement (super popular)
    - can represent any dimensional space
    - works well with obstacles represented as set of points
    - works with any cost function
  - disadvantages:
    - size does depend on the size of the environment
    - can be expensive to compute/store if # of dimensions > 3

What can we do to avoid pre-computing/storing the whole N-dimensional grid?

Cell Decomposition-based Graphs

- Grid-based graph
  - advantages:
    - very simple to implement (super popular)
    - can represent any dimensional space
    - works well with obstacles represented as set of points
    - works with any cost function
  - disadvantages:
    - size does depend on the size of the environment
    - can be expensive to compute/store if # of dimensions > 3

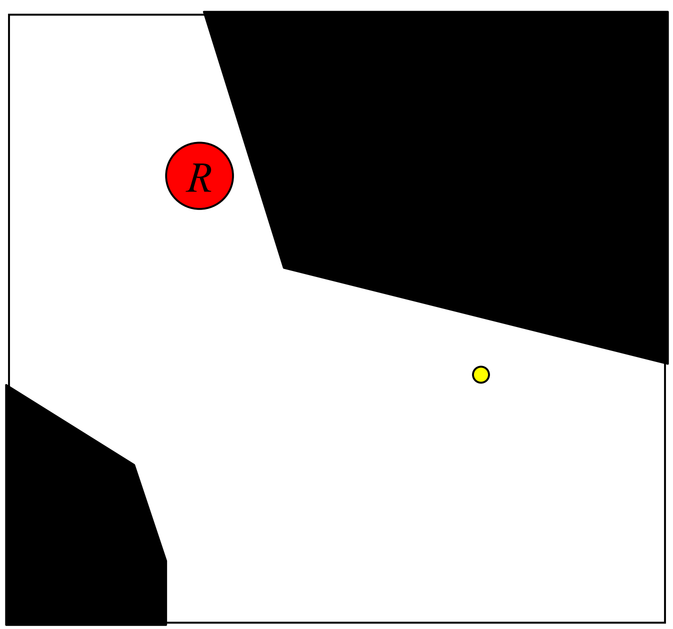
What can we do to avoid pre-computing/storing the whole N-dimensional grid?

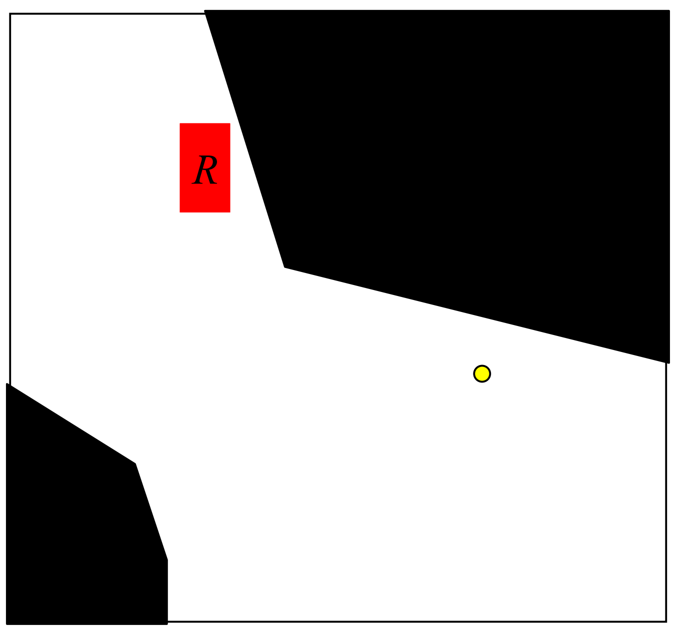
Use Implicit Graphs

2D Planning for Omnidirectional **Non-Circular Non-point** Robot

Planning for <u>omnidirectional point</u> robot:

What is $M^R = <x,y>$
What is $M^W = <obstacle/free\ space>$
What is $s^R_{current} = <x_{current}, y_{current}>$
What is $s^W_{current} = constant$
What is C = *Euclidean Distance*
What is $G = <x_{goal}, y_{goal}>$



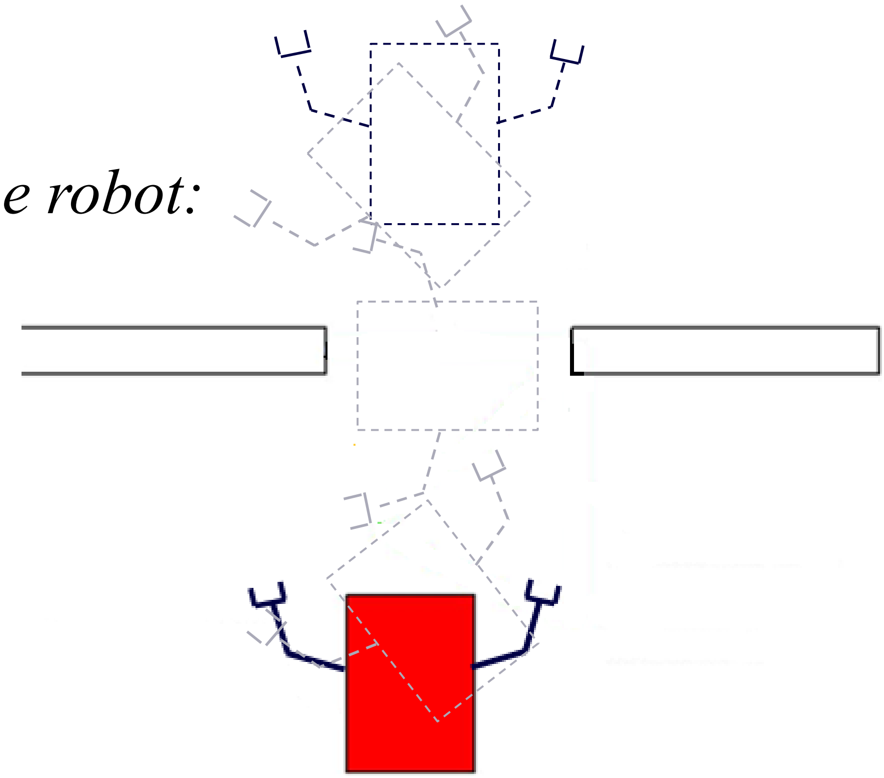


Configuration Space

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

- Configuration Space is the set of legal configurations

Legal configurations for the base of the robot:

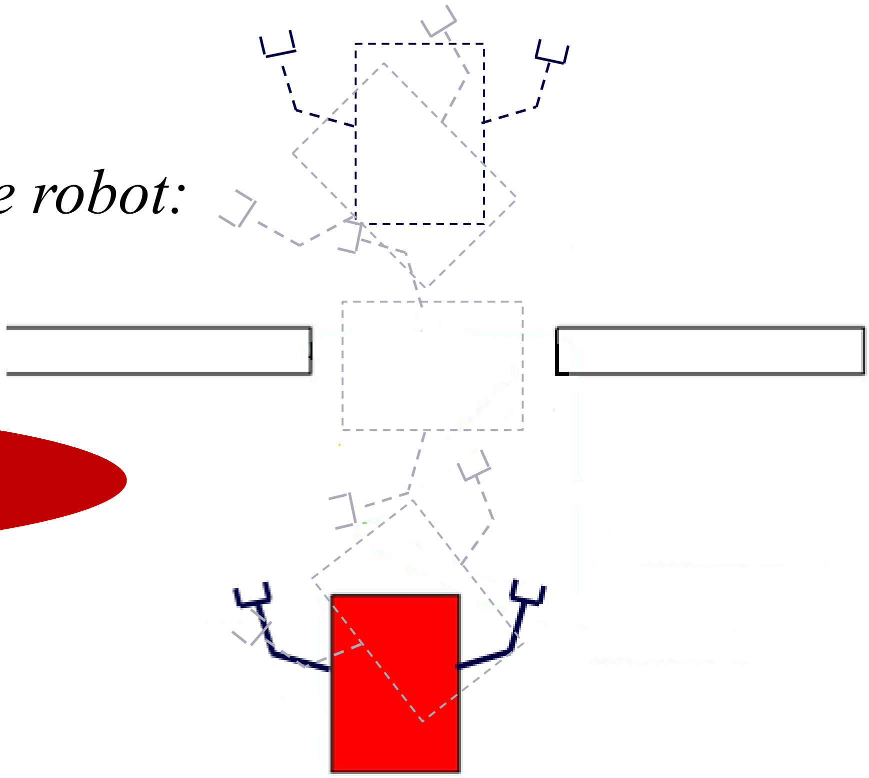


Configuration Space

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

- Configuration Space is the set of legal configurations

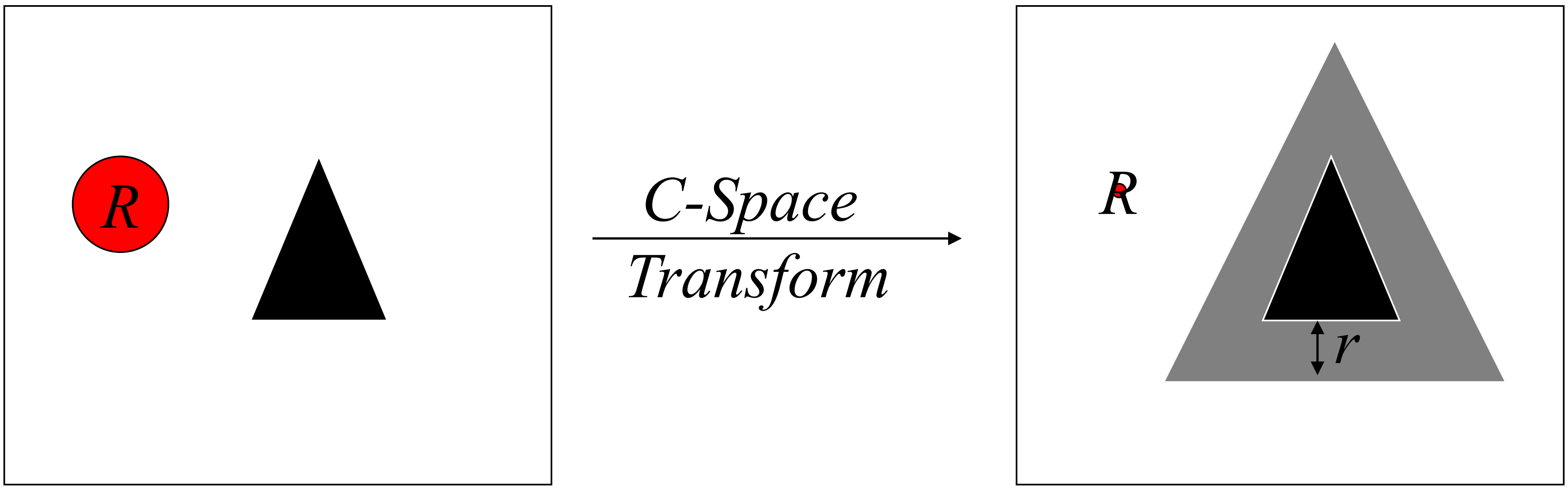
Legal configurations for the base of the robot:



What is the dimensionality of this configuration space?

C-Space Transform

- Configuration space for a robot base in 2D world is:
  - 2D if robot's base is circular

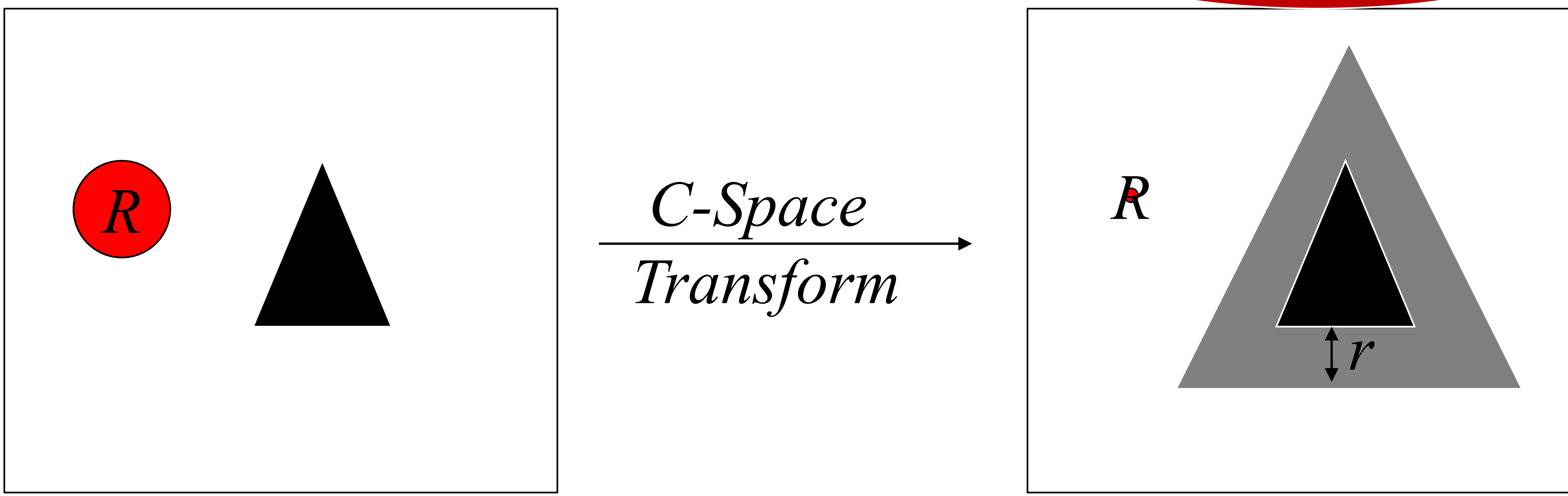


- expand all obstacles by radius r of the robot's base
- graph construction can then be done assuming point robot

C-Space Transform

- Configuration space for a robot base in 2D world is:
  - 2D if robot's base is circular

Is this a correct expansion?

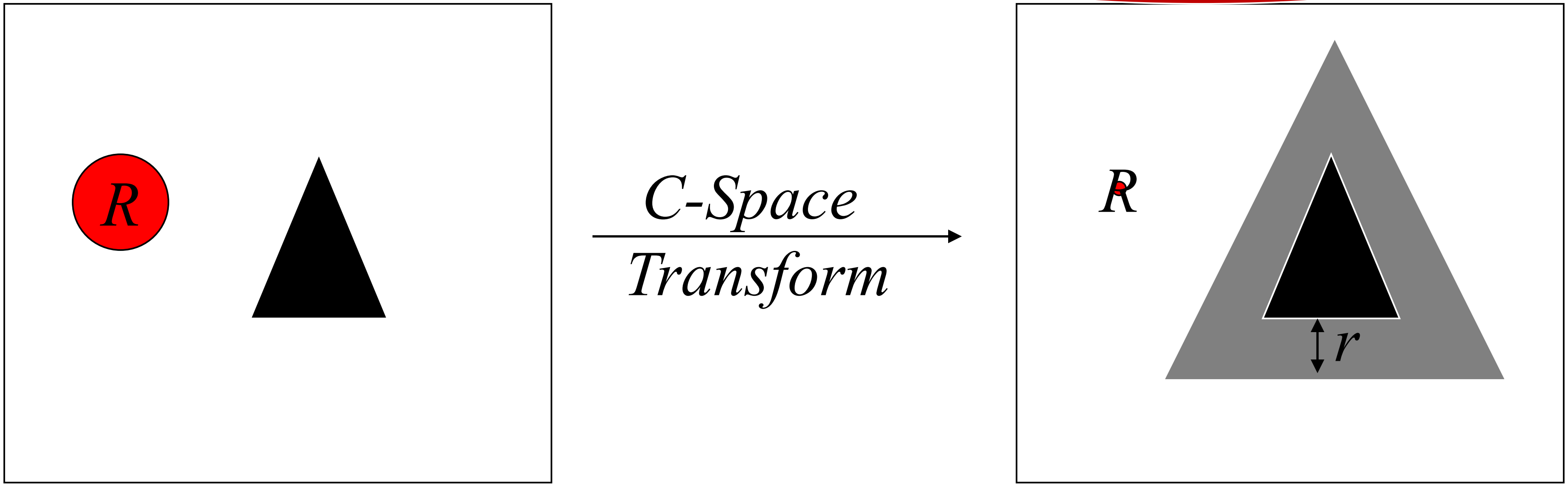


- expand all obstacles by radius r of the robot's base
- graph construction can then be done assuming point robot

C-Space Transform

- Configuration space for a robot base in 2D world is:
  - 2D if robot's base is circular

How to perform expansion of obstacles?



- expand all obstacles by radius r of the robot's base
- graph construction can then be done assuming point robot

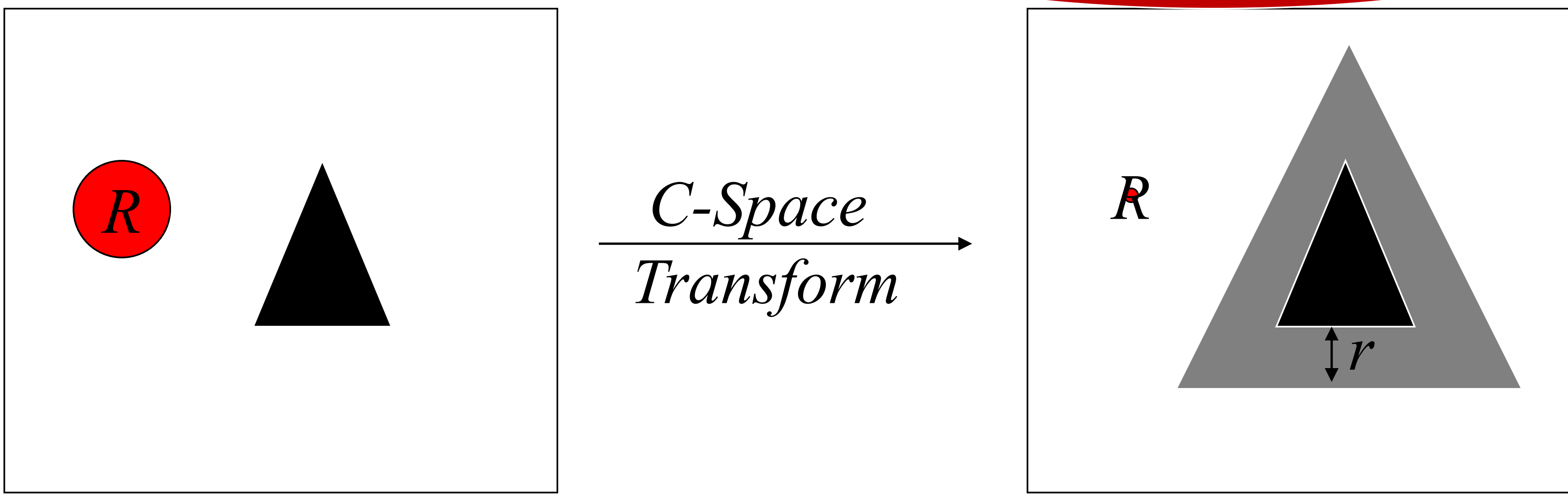
C-Space Transform

- Configuration space for a robot ba

  - 2D if robot's base is circular

O(n) methods exist to compute distance transforms efficiently

How to perform expansion of obstacles?

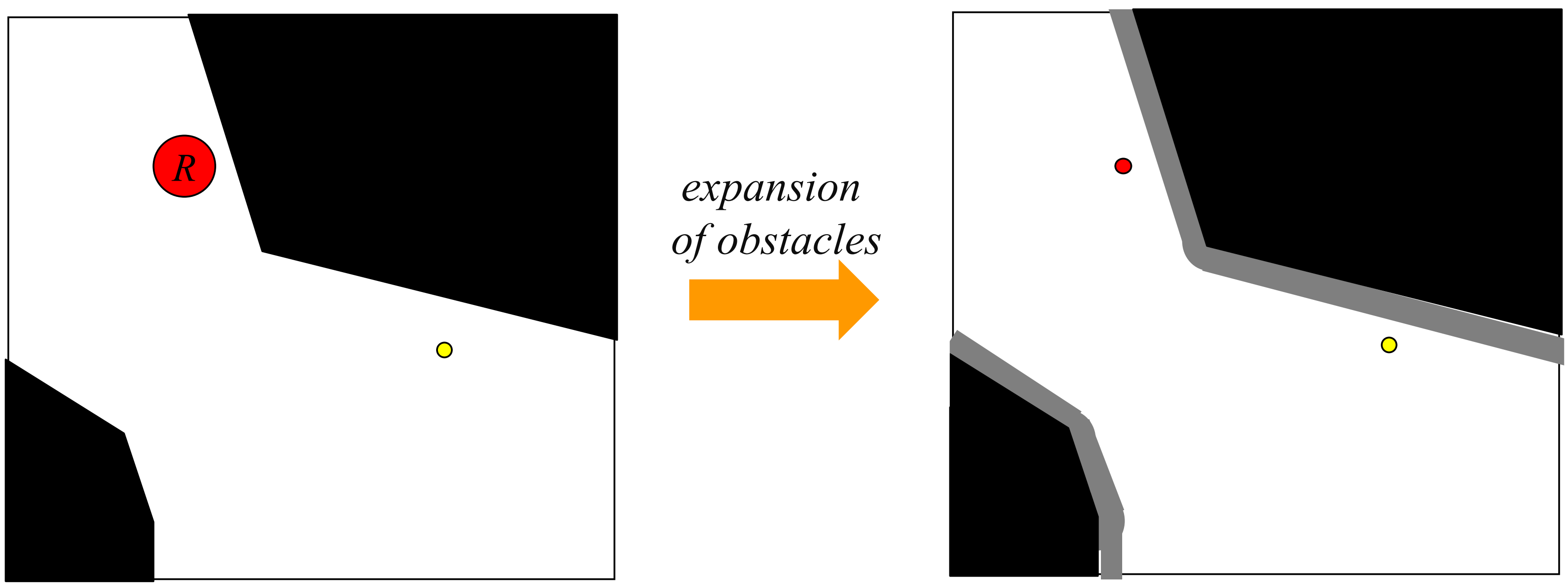


- expand all obstacles by radius r of the robot's base
- graph construction can then be done assuming point robot

2D Planning for Omnidirectional **Non-Circular Non-point** Robot

Planning for omnidirectional circular robot:

What is $M^R = <x,y>$
What is $M^W = <obstacle/free\ space>$
What is $s^R_{current} = <x_{current}, y_{current}>$
What is $s^W_{current} = constant$
What is C = *Euclidean Distance*
What is $G = <x_{goal}, y_{goal}>$

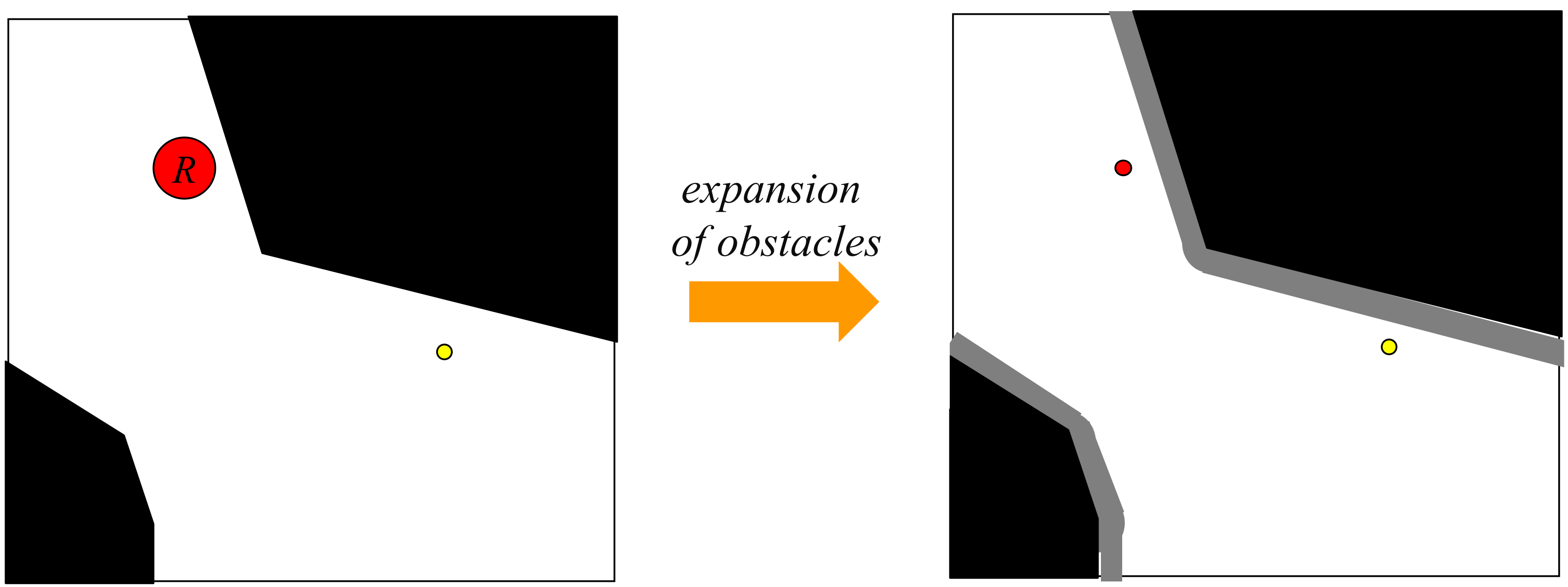


2D Planning for Omnidirectional **Non-Circular Non-point** Robot

Planning for omnidirectional circular robot:

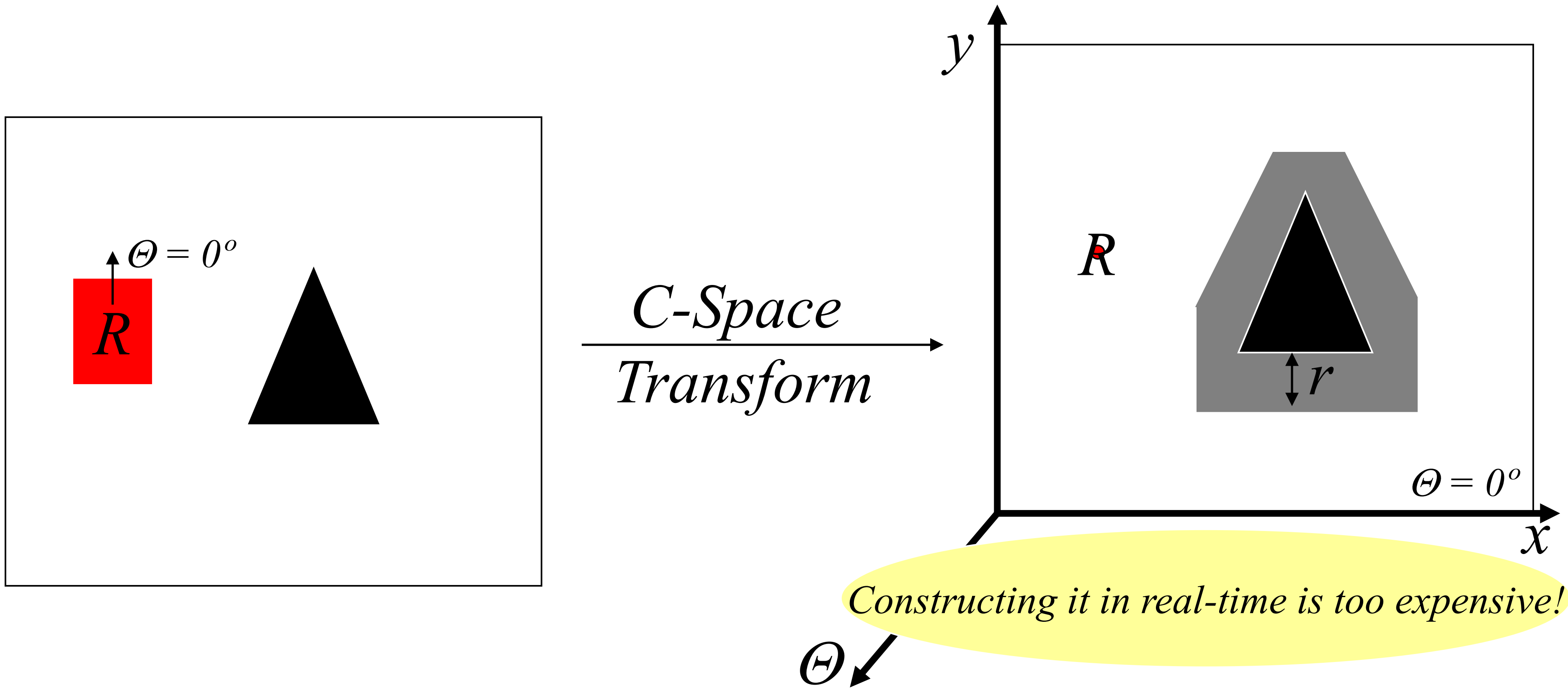
What is $M^R = <x,y>$
What is $M^W = <obstacle/free\ space>$
What is $s^R_{current} = <x_{current}, y_{current}>$
What is $s^W_{current} = constant$
What is C *= Euclidean Distance*
What is $G = <x_{goal}, y_{goal}>$

We can now construct a graph using previously discussed methods (grids, Voronoi graphs, Visibility graphs)



C-Space Transform

- Configuration space for a robot base in 2D world is:
  - 3D if robot's base is non-circular

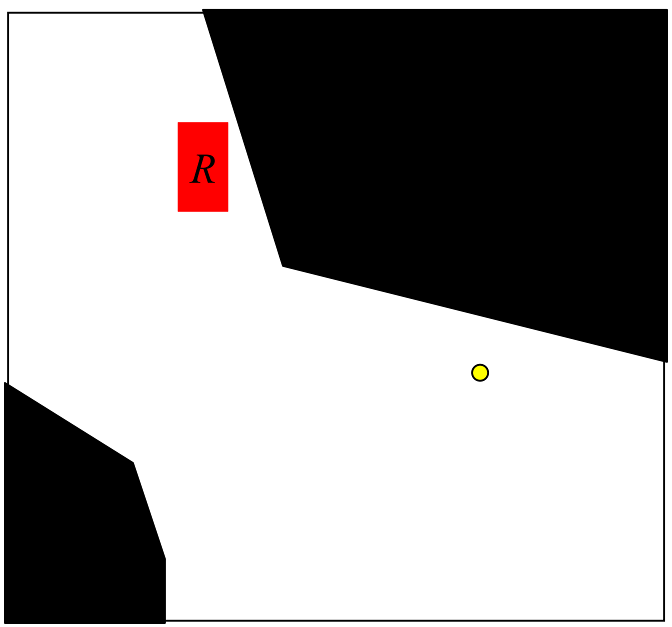


Planning for Omnidirectional **Non-Circular Non-point** Robot

Planning for omnidirectional non-circular robot:

What is $M^R = <x,y,\Theta>$
What is $M^W = <obstacle/free\ space>$
What is $s^R_{current} = <x_{current}, y_{current}, \Theta_{current}>$
What is $s^W_{current} = constant$
What is C = *Euclidean Distance*
What is $G = <x_{goal}, y_{goal}, \Theta_{goal}>$

Interleave
Graph Construction and Graph Search steps!

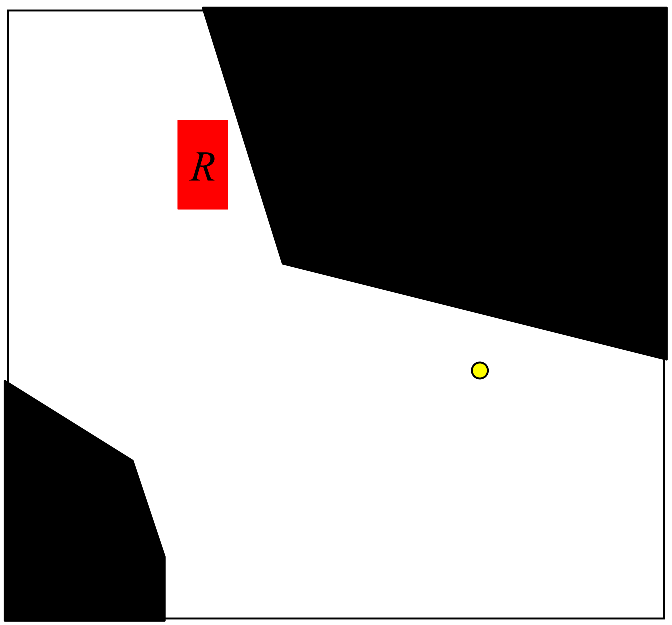


Construct a 3D grid (x,y,Θ) assuming point robot (i.e., a cell (x,y,Θ) is free whenever its (x,y) is free) and compute the ***actual*** *validity of only those cells that get computed by the graph search*

Planning for Omnidirectional **Non-Circular Non-point** Robot

Planning for omnidirectional non-circular robot:

What is $M^R = <x,y,\Theta>$
What is M^W = *<obstacle/free space>*
What is $s^R_{current} = <x_{current}, y_{current}, \Theta_{current}>$
What is $s^W_{current}$ = *constant*
What is C = Euclidean Distance
What is $G = <x_{goal}, y_{goal}, \Theta_{goal}>$



Interleave
Graph Construction and Graph Search steps!

*Construct a 3D grid (x,y,Θ) assuming point robot (i.e., a cell (x,y,Θ) is free whenever its (x,y) is free) and compute the **actual** validity of only those cells that get computed by the graph search*

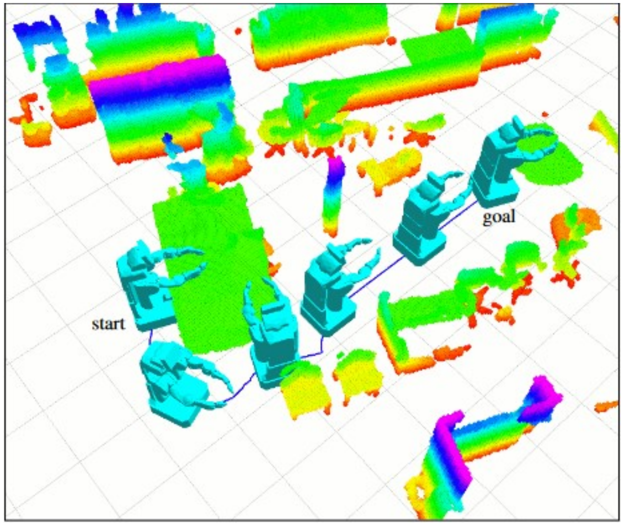
How to compute the actual validity of cell (x,y,Θ)?

Planning for Omnidirectional **Non-Circular Non-point** Robot

Planning for omnidirectional non-circular robot:

What's different when planning for a robot that has a complex 3D body?

What is $M^R = \langle x, y, \Theta \rangle$
What is $M^W = \langle obstacle/free\ space \rangle$
What is $s^R_{current} = \langle x_{current}, y_{current}, \Theta_{current} \rangle$
What is $s^W_{current} = constant$
What is $C = Euclidean\ Distance$
What is $G = \langle x_{goal}, y_{goal}, \Theta_{goal} \rangle$



Two Classes of Graph Construction Methods

- Skeletonization

  -Visibility graphs

  -Voronoi diagrams

  - Probabilistic roadmaps

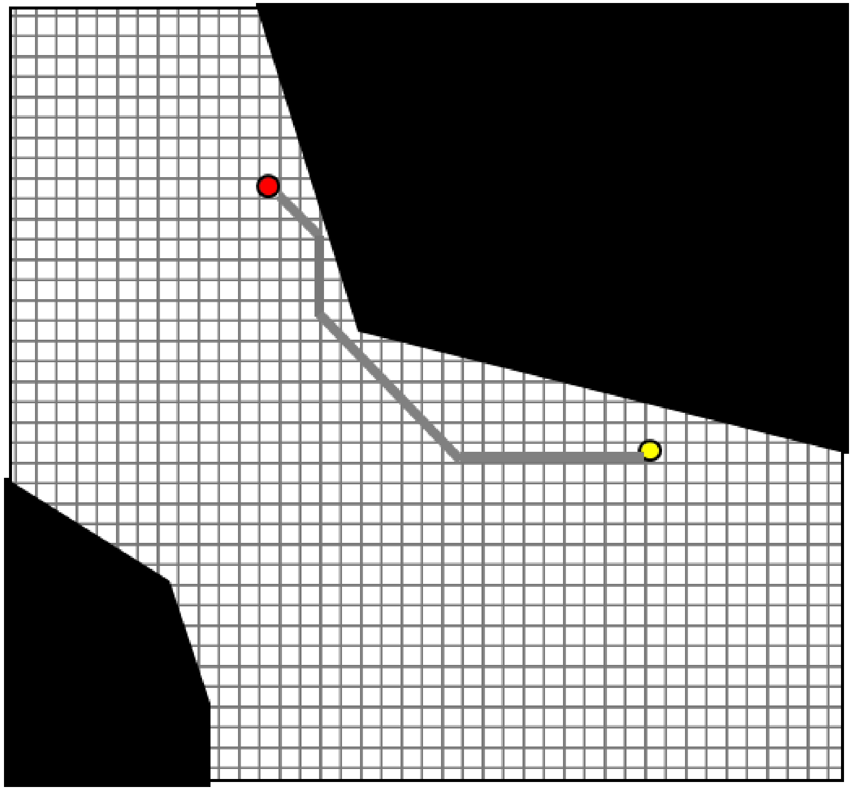
- Cell decomposition

  - X-connected grids

  - lattice-based graphs

Beyond Planning for Omnidirectional Robots

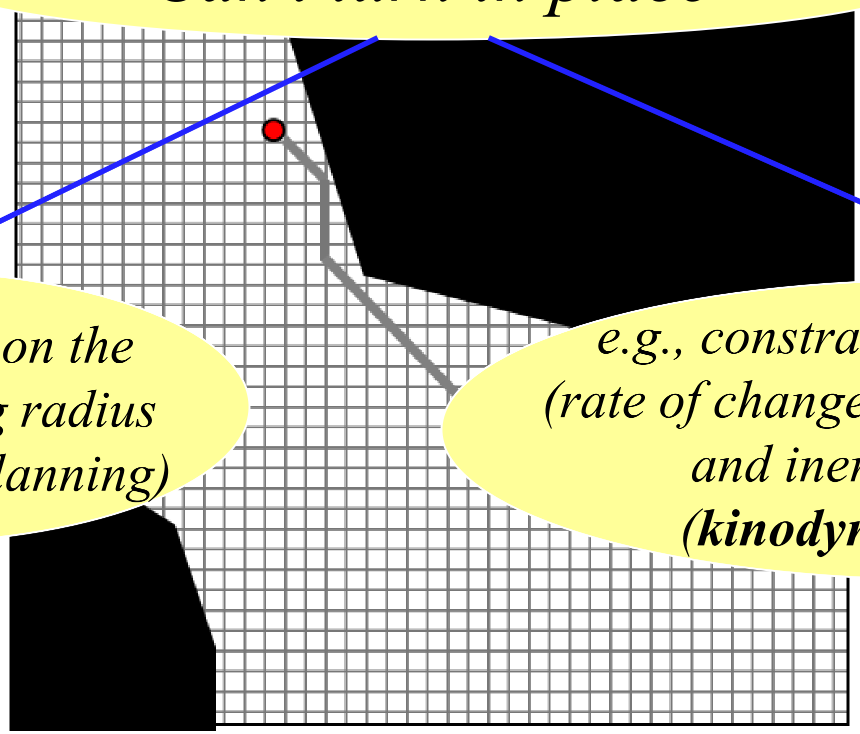
What's wrong with using
Grid-based Graphs when planning
for non-omnidirectional robots?

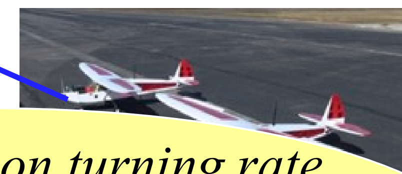


Beyond Planning for Omnidirectional Robots

What's wrong with using Grid-based Graphs when planning for non-omnidirectional robots?

"Can't turn in place"



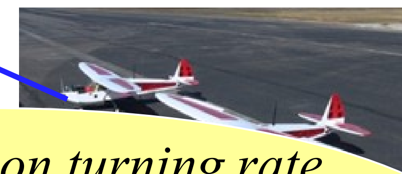


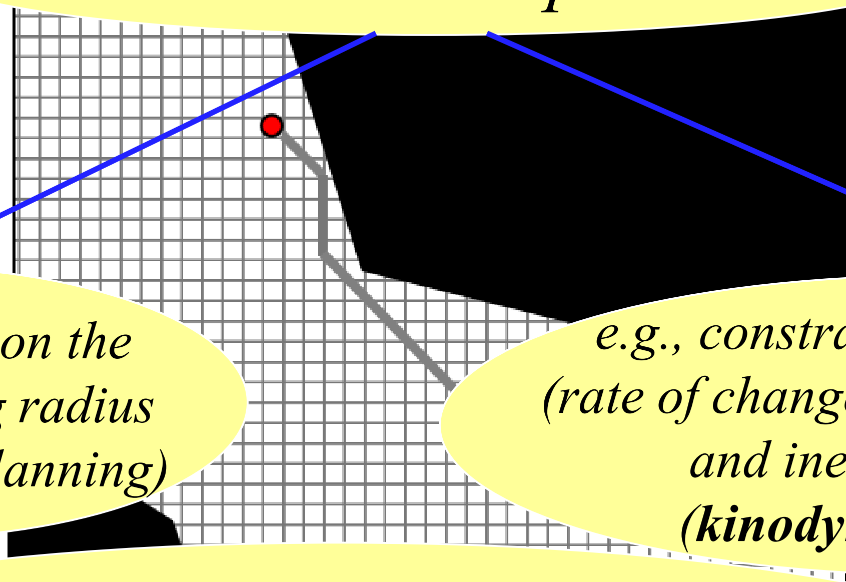
e.g., constraints on the minimum turning radius (still ***kinematic*** *planning)*

*e.g., constraints on turning rate (rate of change in wheel orientation) and inertial constraints (****kinodynamic*** *planning)*

Beyond Planning for Omnidirectional Robots

What's wrong with using
Grid-based Graphs when planning
for non-omnidirectional robots?





"Can't turn in place"

e.g., constraints on the
minimum turning radius
(still ***kinematic*** *planning)*

e.g., constraints on turning rate
(rate of change in wheel orientation)
and inertial constraints
*(**kinodynamic** planning)*

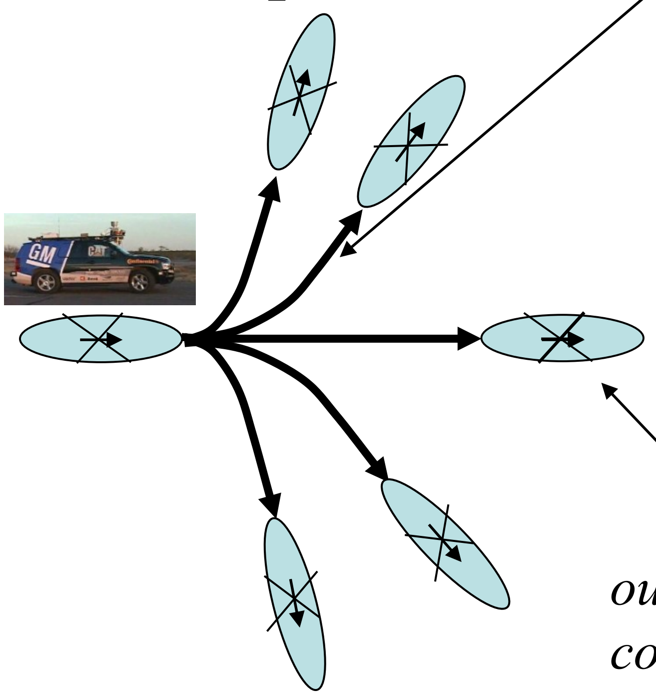
***Kinodynamic planning**:*
Planning representation includes {X, $\dot{X}$},
where X-configuration and $\dot{X}$-derivative of X (dynamics of X)

Lattice Graphs [Pivtoraiko & Kelly ’05]

- Graph *{V, E}* where
  - *V*: centers of the grid-cells
  - *E*: motion primitives that connect centers of cells via short-term **feasible** motions

each transition is feasible
(typically, constructed beforehand)

motion primitives

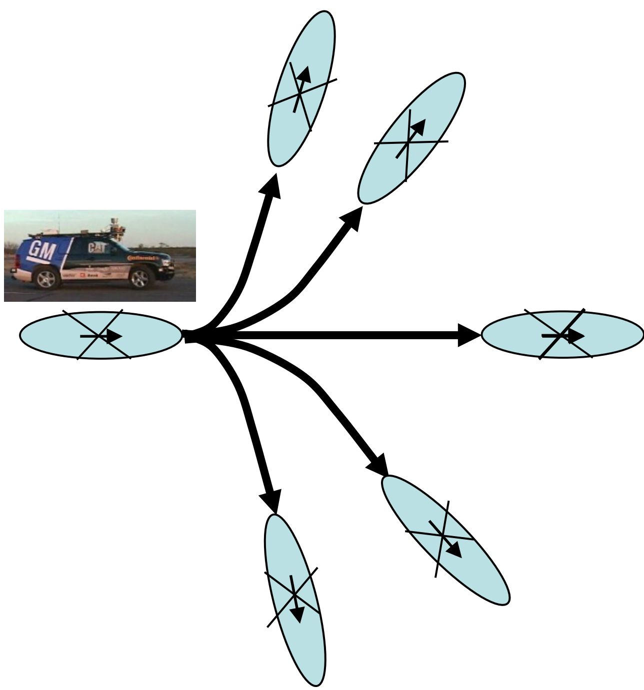


outcome state is the center of the corresponding cell in a grid

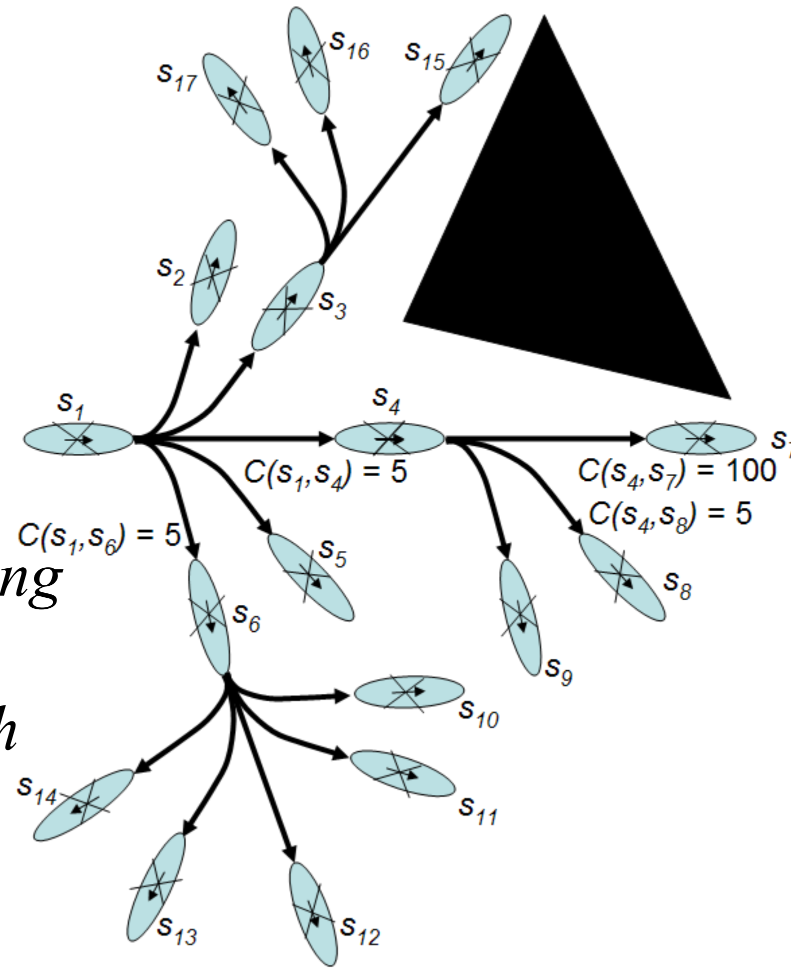
Lattice Graphs [Pivtoraiko & Kelly ’05]

- Graph *{V, E}* where
  - *V*: centers of the grid-cells
  - *E*: motion primitives that connect centers of cells via short-term **feasible** motions

motion primitives



replicate it during planning to generate lattice graph

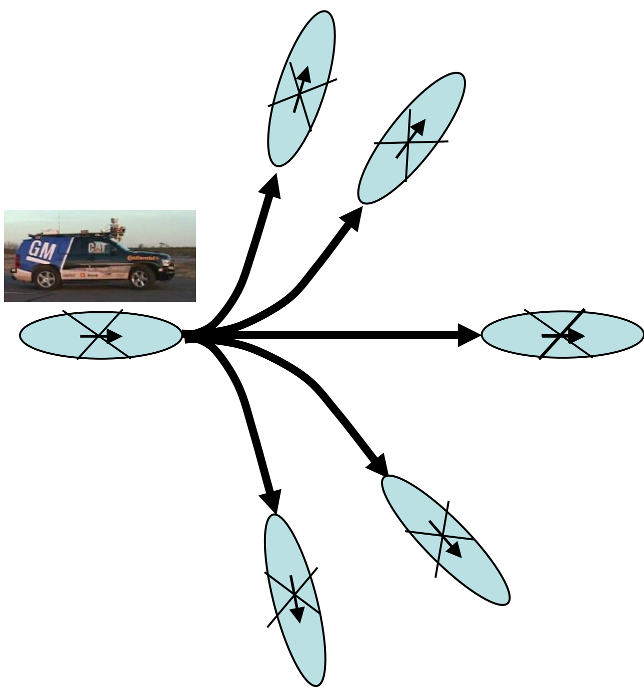


Lattice Graphs [Pivtoraiko & Kelly '05]

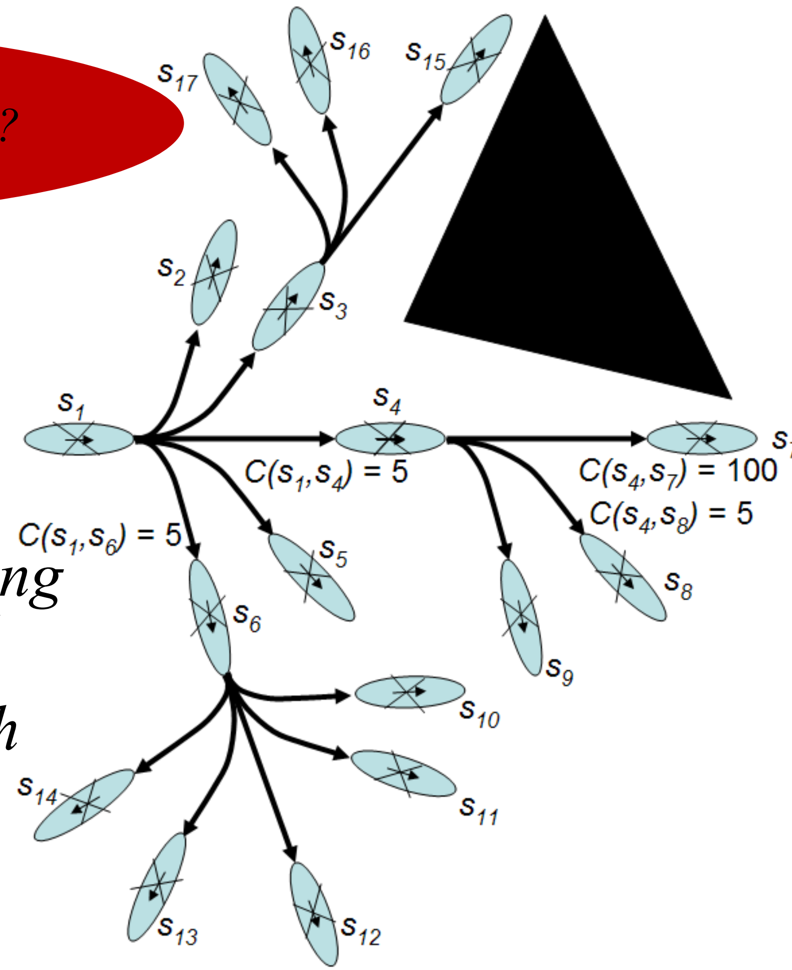
- Graph *{V, E}* where
  - *V*: centers of the grid-cells
  - *E*: motion primitives that connect centers of cells via short-term **feasible** motions

How do edgecosts get assigned?

motion primitives



replicate it during planning to generate lattice graph



What You Should Know…

- Explicit vs. Implicit graphs
- What visibility graphs are
- What Voronoi diagram-based graphs are
- X-connected N-dimensional grids
- Lattice-based graphs