15-381: AI – Fall 2009

More on Path Planning

Manuela Veloso

Thanks to previous instructors

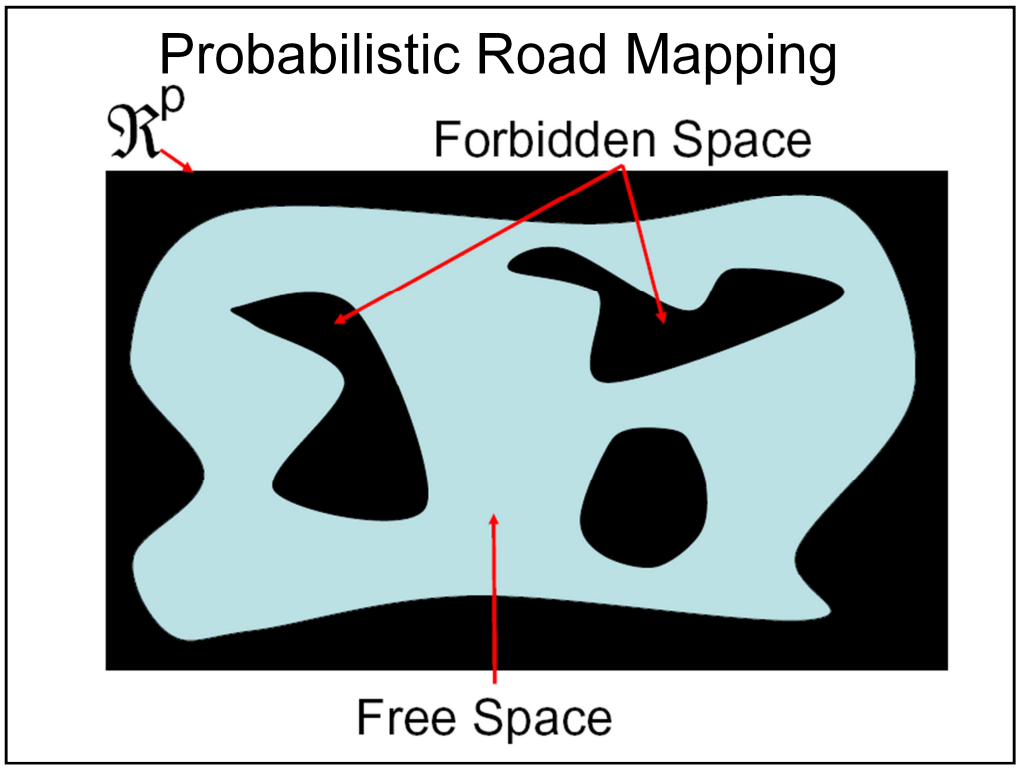
Carnegie Mellon University
Computer Science Department

Robot Motion Planning

- Probabilistic approaches:
  - PRM
  - RRT, ERRT
- Non-probabilistic appr
  - Visibility graphs
  - Voronoi diagrams
  - Cell decomposition
  - Potential fields
- Path planning by analogy

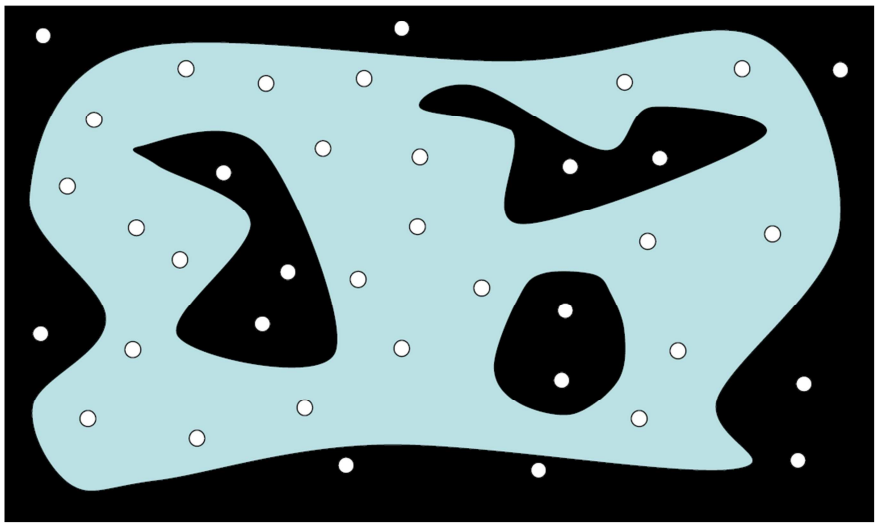
Reduce the intractable problem in continuous space to a tractable problem in a discrete space → Use known search techniques (A*, stochastic search, etc.)

Sampling, etc can help us reduce it to a discrete space.


Probabilistic Road Mapping
$\mathfrak{R}^p$
Forbidden Space
Free Space

Probabilistic Road Mapping

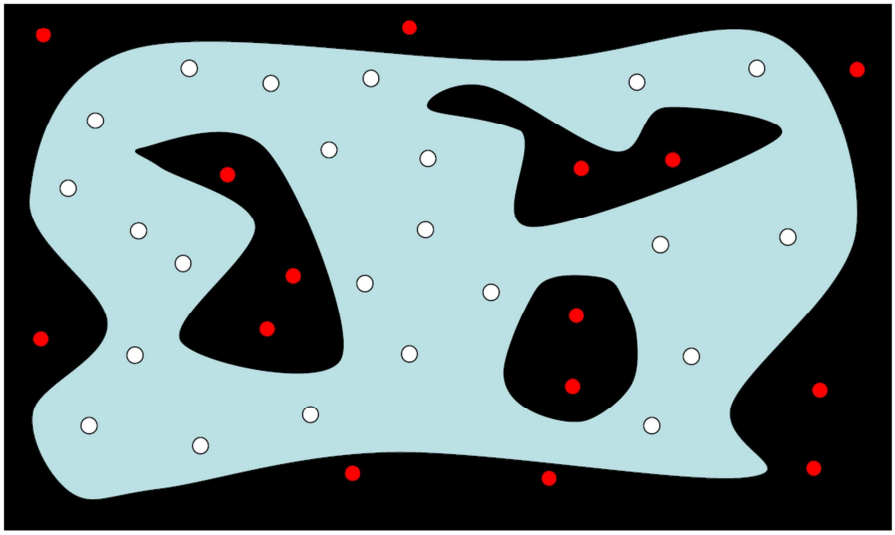
Sample random locations



Now the space is discretized…we only need to consider the sampled points.

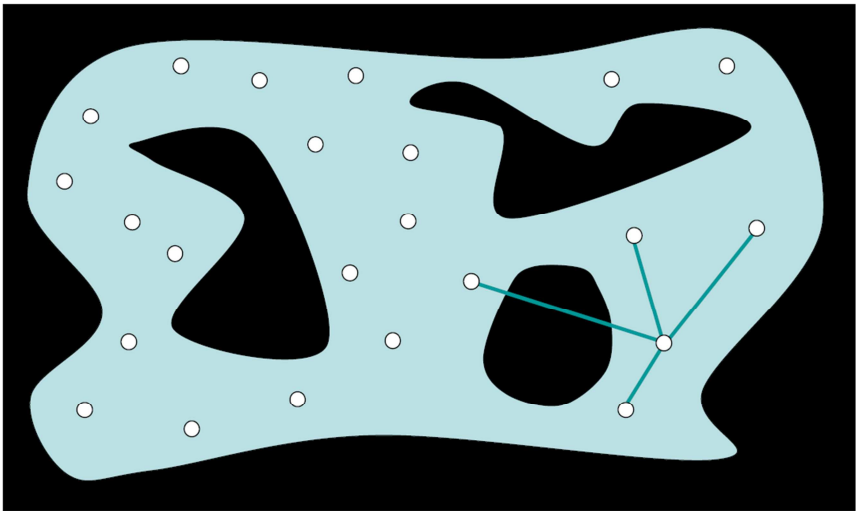
Probabilistic Road Mapping

Remove the samples in the forbidden regions



Probabilistic Road Mapping

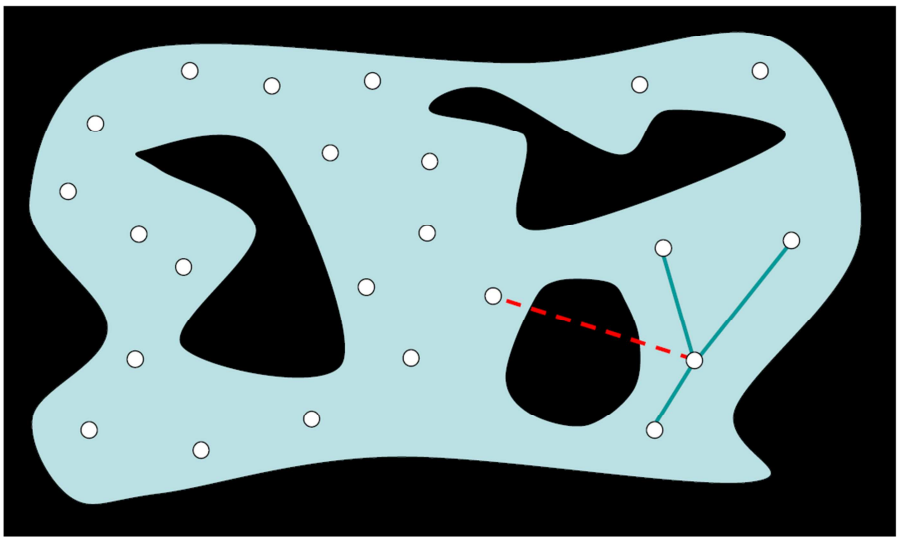
Link each sample to its *K* nearest neighbors



Here K = 4

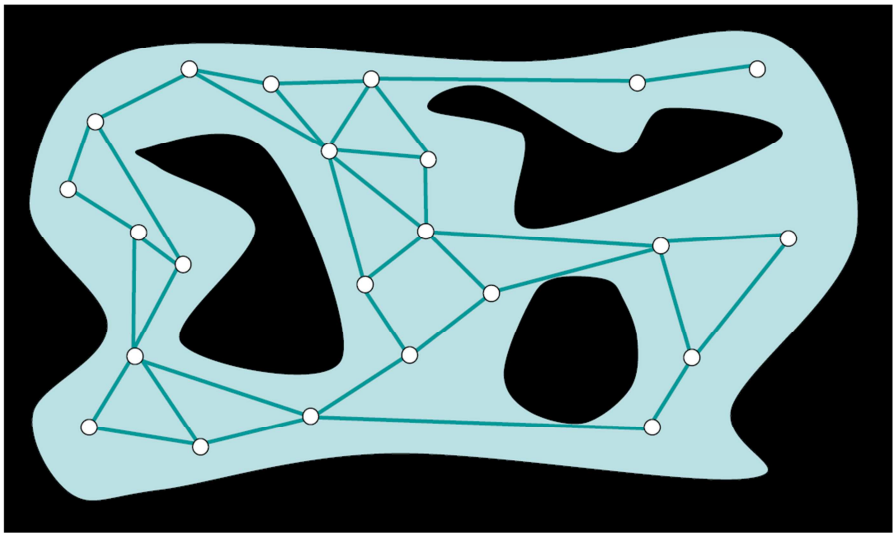
Probabilistic Road Mapping

Remove the links that cross forbidden regions



Probabilistic Road Mapping

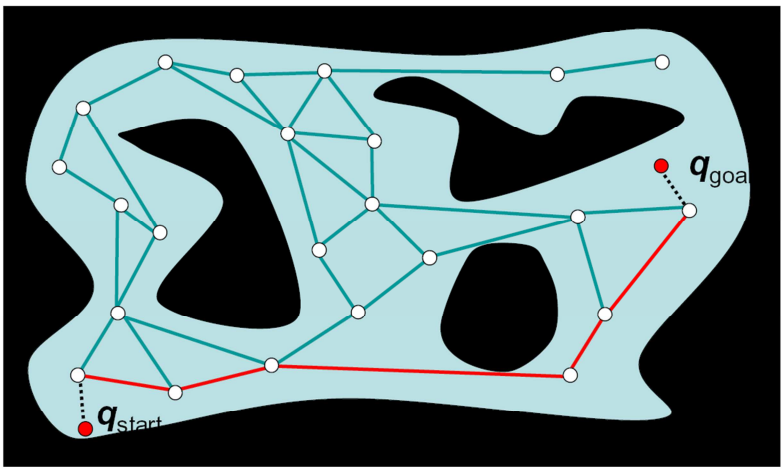
Remove the links that cross forbidden regions



The resulting graph is a *probabilistic roadmap (PRM)*

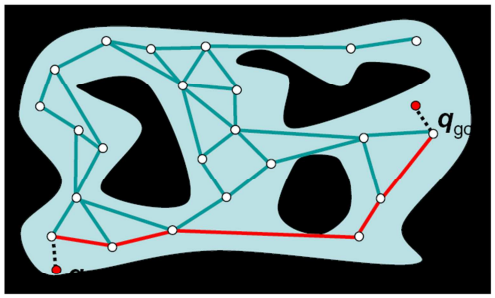
Probabilistic Road Mapping

Link the start and goal to the PRM and search using A*



Probabilistic Road Mapping

Continuous Space



Discretization

A* Search

- "Good" sampling strategies are important:
  - Uniform sampling
  - Sample more near points with few neighbors
  - Sample more close to the obstacles
  - Use pre-computed sequence of samples

Is this complete? We get a good approximation, IF we do the sampling well. You can't just sample the left side of the map or else you can't plan on the right…you need to sample uniformly.

How do we do that?

Fewer links (successors) implies obstacles, so we should do more sampling of points.

Sampling Techniques

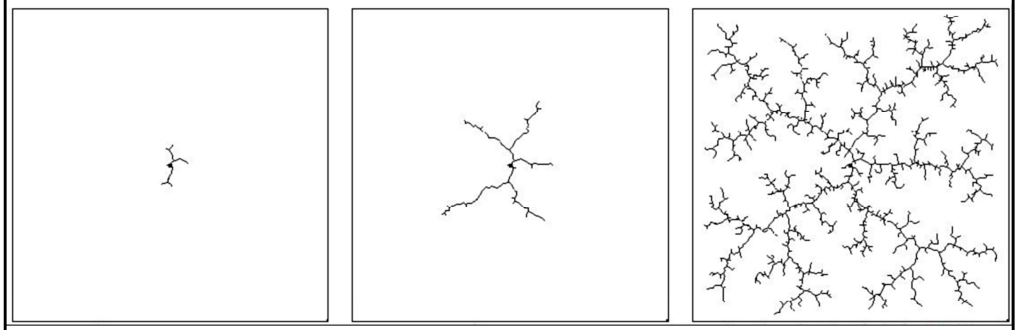
- Remarkably, we can find a solution by using *relatively few randomly* sampled points.
- In most problems, a relatively small number of samples is sufficient to cover most of the feasible space with probability 1
- For a large class of problems:
  - Prob(finding a path) → 1 exponentially with the number of samples
- *But*, cannot detect that a path does not exist

The more you sample, the more you're covering the space. So you ought to be able to get epsilon close to optimal.

You can't say there doesn't exist a path since in continuous space you can't really cover the entire space.

Searching for a path through the PRM gets more time consuming as the number of samples goes to infinity

RRT Properties



- Tends to explore the space rapidly in all directions
- Does not require extensive pre-processing
- Single query/multiple query problems
- Needs only collision detection test → No need to represent/pre-compute the entire C-space

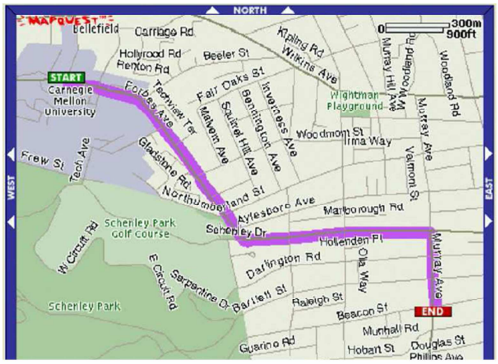
ERRT – RRT with Replanning

1) Start with initial state as root of tree
2) Pick a random target state
   - Goal configuration with probability p
   - Random item from waypoint cache with probability q
   - Random configuration with probability 1-q-p
3) Find the closest node in the tree
4) Extend the closest node toward the target
5) Goto step 2

Robot Motion Planning

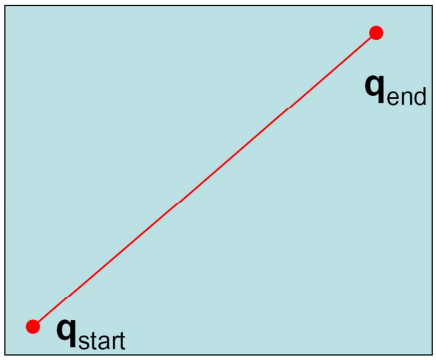
- Probabilistic approaches:
  - PRM
  - RRT, ERRT
- Non-probabilistic approaches:
  - Visibility graphs
  - Voronoi diagrams
  - Cell decomposition
  - Potential fields
- Path planning by analogy

Roadmaps



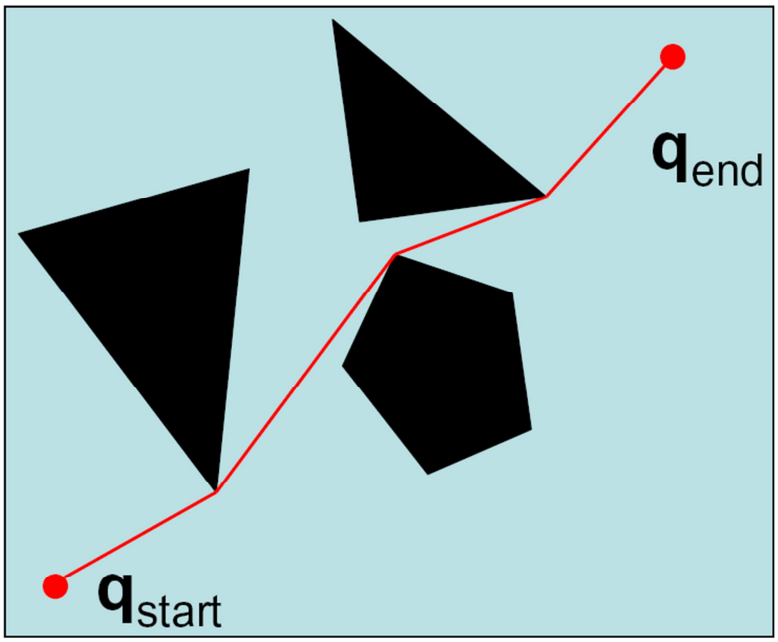
- General idea:
  - Avoid searching the entire space
  - Pre-compute a (hopefully small) graph (the roadmap) such that staying on the "roads" is guaranteed to avoid the obstacles
  - Find a path between $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ by using the roadmap

Visibility Graphs



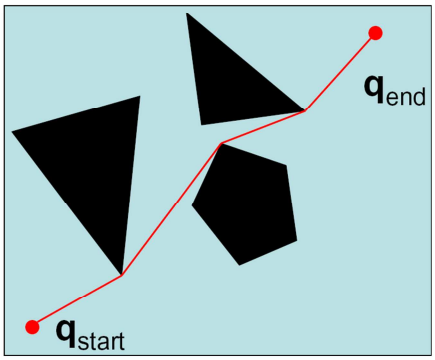
In the absence of obstacles, the best path is the straight line between $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$

Visibility Graphs



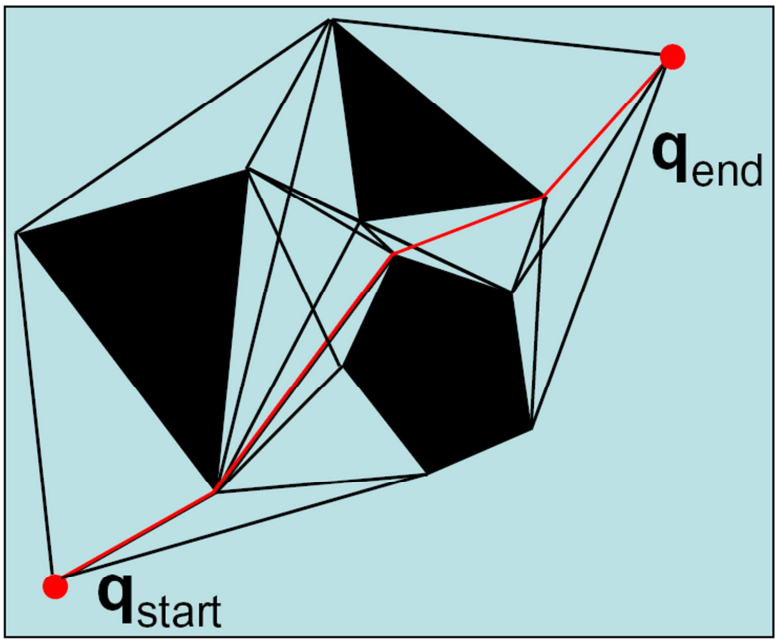
Shortest path includes vertices of the shapes.

Visibility Graphs



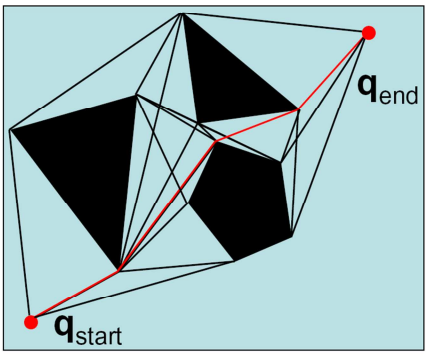
- Assuming polygonal obstacles: It looks like the shortest path is a sequence of straight lines joining the vertices of the obstacles.
- Is this always true?

Visibility Graphs

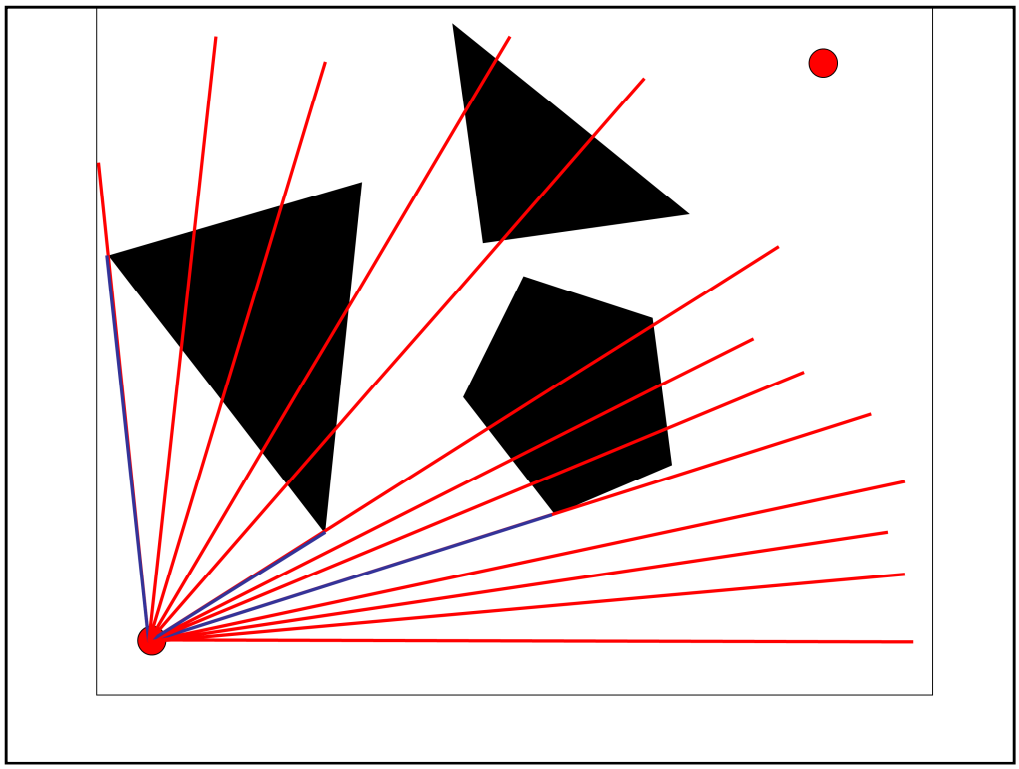


When the world is static, you can do a lot of geometrical computation like this. Dynamic environments depend more on probability, since as the world changes your geometric computations become invalid.

Visibility Graphs

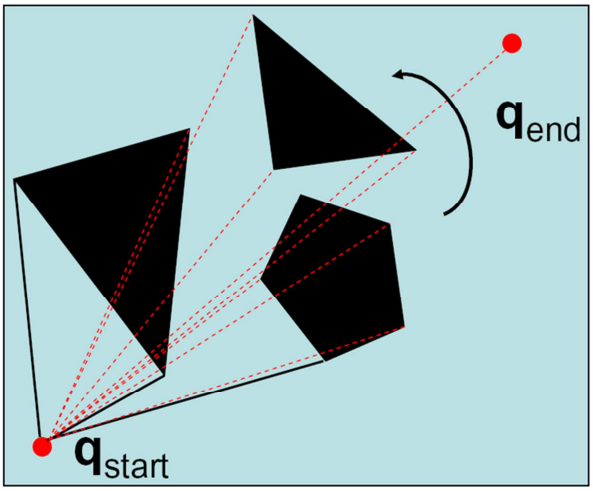


- Visibility graph *G* = set of unblocked lines between vertices of the obstacles + $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$
- A node *P* is linked to a node *P'* if *P'* is visible from *P*
- Solution = Shortest path in the visibility graph



Sweep your ray until you hit a vertex of a shape.

Construction: Sweep Algorithm



- Sweep a line originating at each vertex
- Record those lines that end at visible vertices

Visibility Graphs: Weaknesses

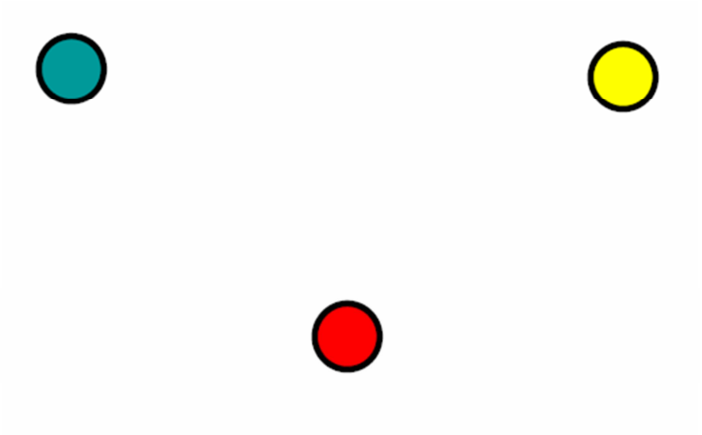
- Shortest path but:
  - Tries to stay as close as possible to obstacles
  - Any execution error will lead to a collision
  - Complicated in >> 2 dimensions
- We may not care about strict optimality so long as we find a safe path. Staying away from obstacles is more important than finding the shortest path
- Need to define other types of “roadmaps”

You can increase the size of the obstacles to avoid collisions (if you pretend a rock is bigger than it is, then you will stay further away from the rock).

Note we assume polygons. Round obstacles will probably screw you up.

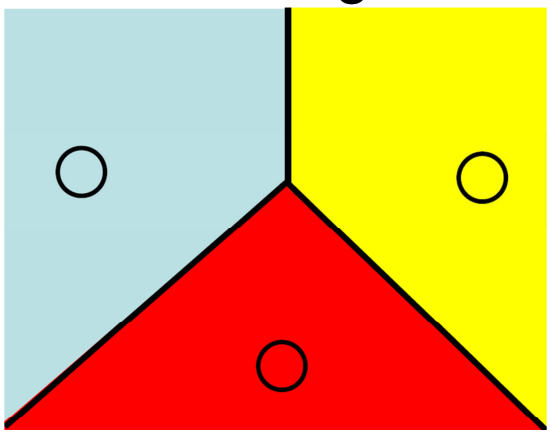
These algorithms generally have trouble in the real world, where you don’t have perfect knowledge of obstacles, and you have noisy observations of these obstacles.

Voronoi Diagrams



- Given a set of data points in the plane:
  - Color the entire plane such that the color of any point in the plane is the same as the color of its nearest neighbor

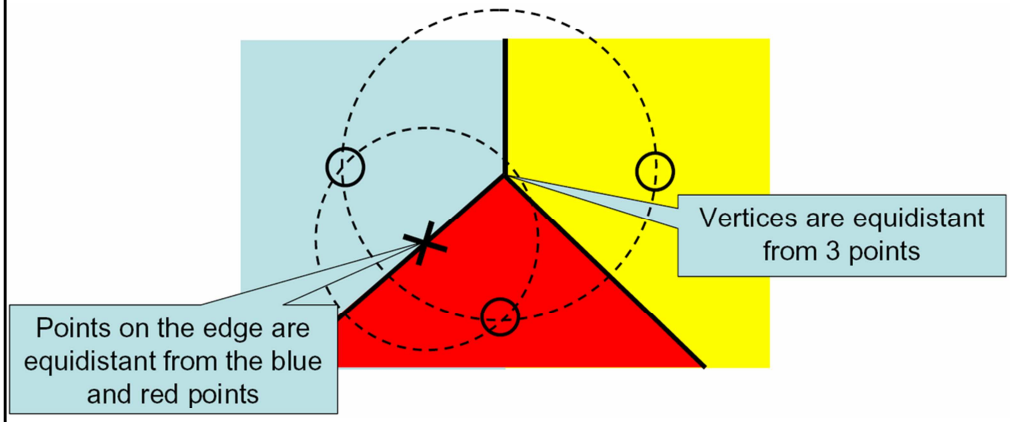
Voronoi Diagrams



- Voronoi diagram = The set of line segments separating the regions corresponding to different colors
  - Line segment = points equidistant from 2 data points
  - Vertices = points equidistant from > 2 data points

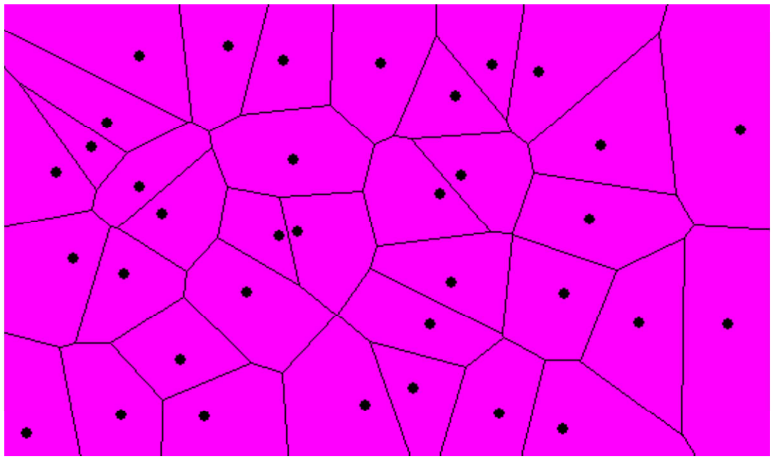
The space is colored by the nearest obstacle. The boundaries between colored regions are the points equidistant from two or more obstacles. Here the wall (the edge of the state space) is not considered to be an obstacle, though sometimes it is.

Voronoi Diagrams



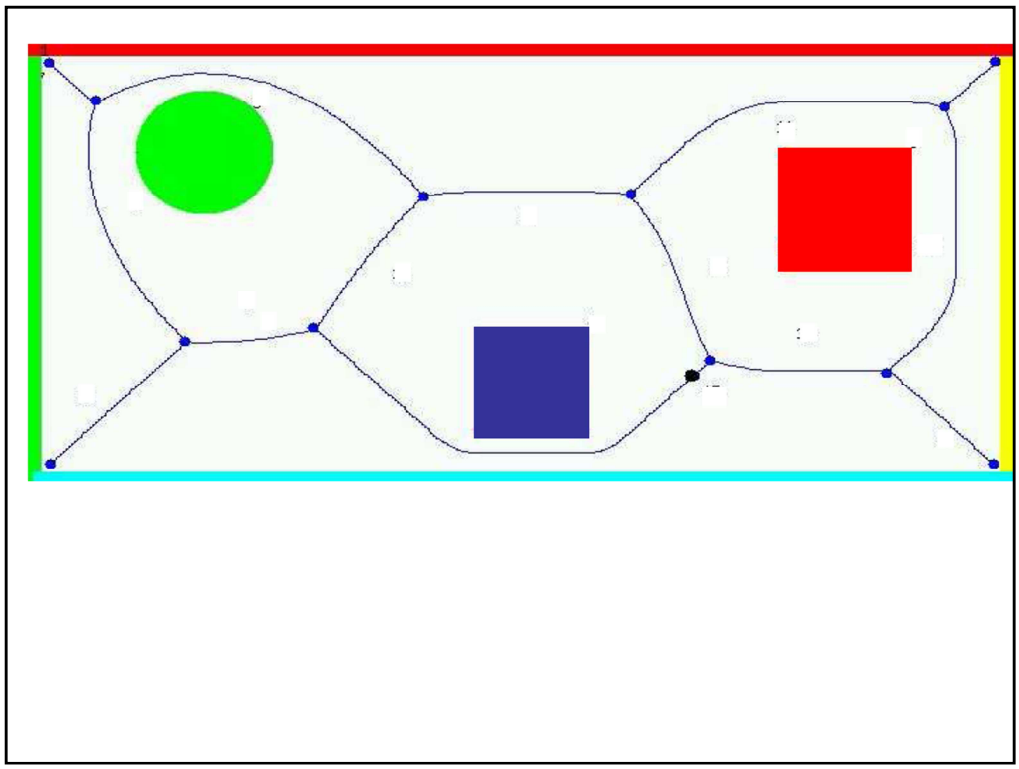
- Voronoi diagram = The set of line segments separating the regions corresponding to different colors
  - Line segment = points equidistant from 2 data points
  - Vertices = points equidistant from > 2 data points

Voronoi Diagrams

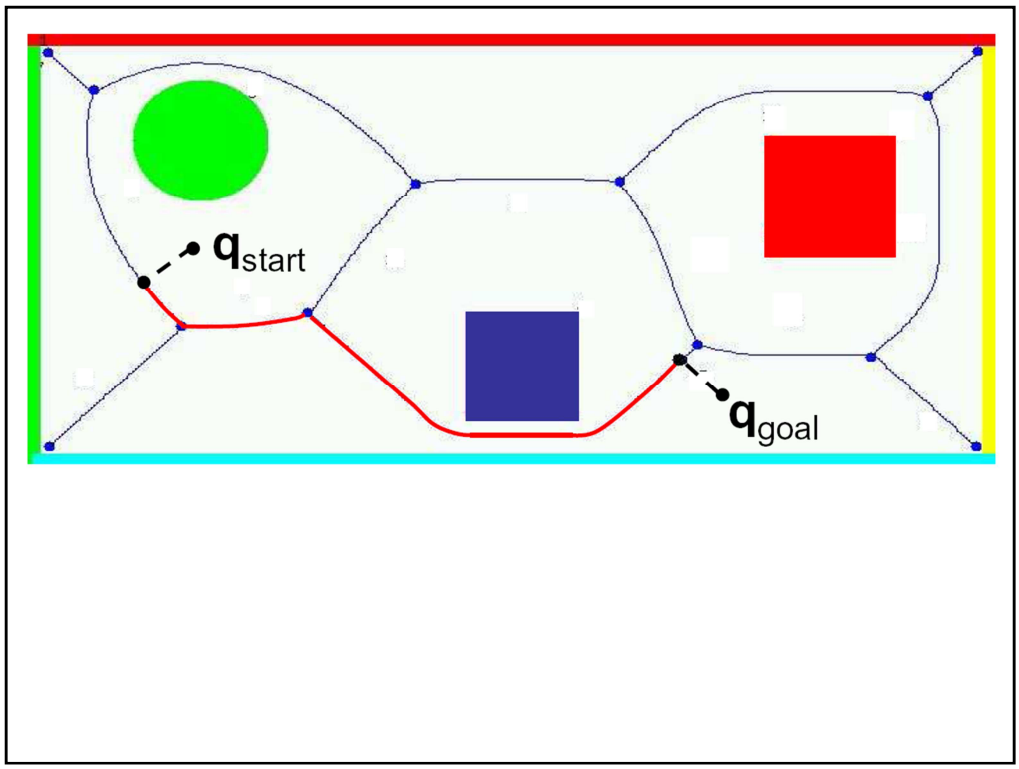


- (See for example http://www.cs.cornell.edu/Info/People/chew/Delaunay.html for an interactive demo)

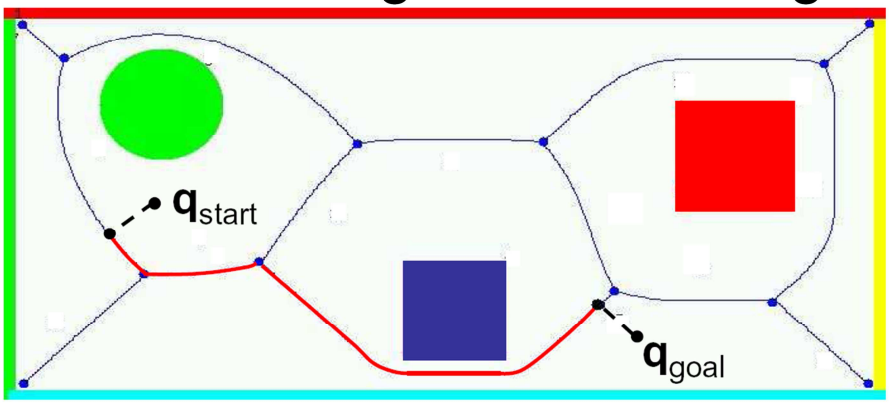
Black dots are obstacles.



Here the walls are obstacles.


q_{start}
q_{goal}

Voronoi Diagrams: Planning



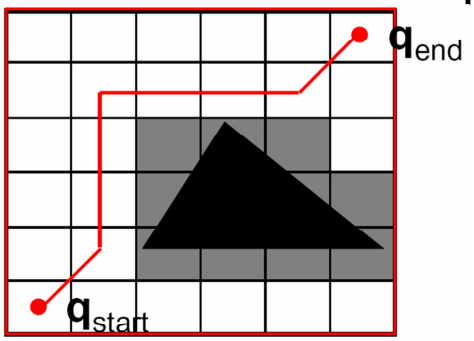
- Find the point $\mathbf{q}^*_{start}$ of the Voronoi diagram closest to $\mathbf{q}_{start}$
- Find the point $\mathbf{q}^*_{goal}$ of the Voronoi diagram closest to $\mathbf{q}_{goal}$
- Compute shortest path from $\mathbf{q}^*_{start}$ to $\mathbf{q}^*_{goal}$ on the Voronoi diagram

You navigate on the edges of the Vonoroi Diagram. The edges are the safest ways to get around the obstacles. On a plane with point obstacles, you can generate the diagram in O(nlogn) for n points. This gets harder for weird shapes and higher dimensions.

Voronoi: Weaknesses

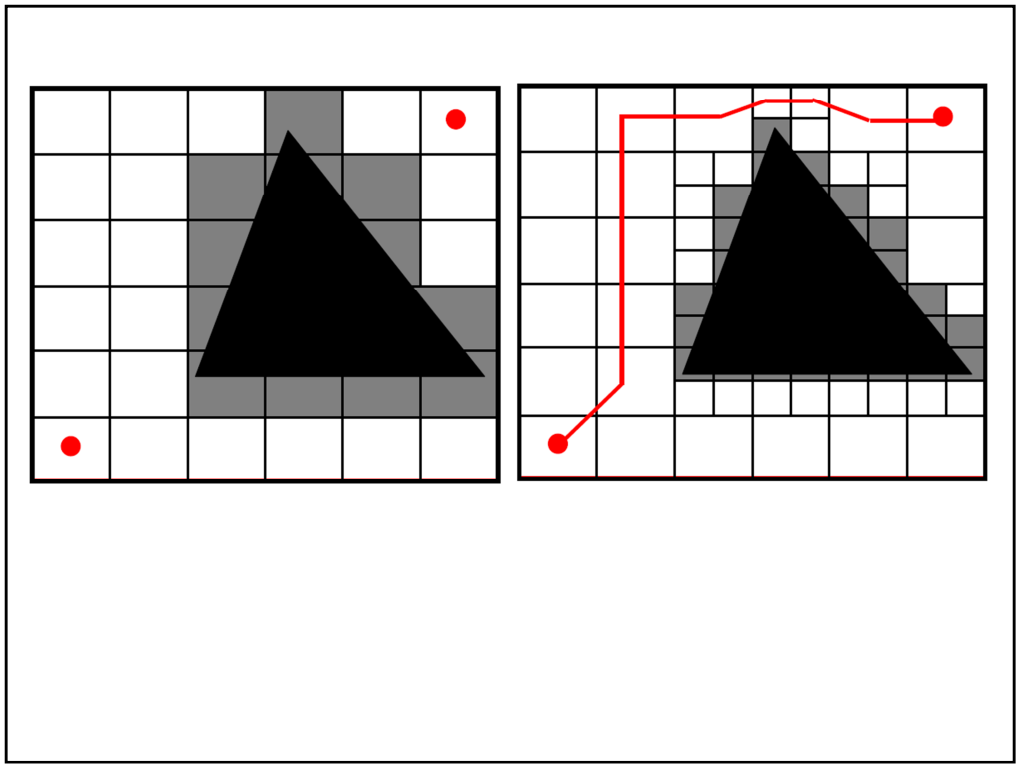
- Difficult to compute in higher dimensions or nonpolygonal worlds
- Approximate algorithms exist
- Use of Voronoi is not necessarily the best heuristic ("stay away from obstacles") Can lead to paths that are much too conservative, or lead to "ranging sensor deprivation"
- Can be unstable → Small changes in obstacle configuration can lead to large changes in the diagrams

Approximate Cell Decomposition



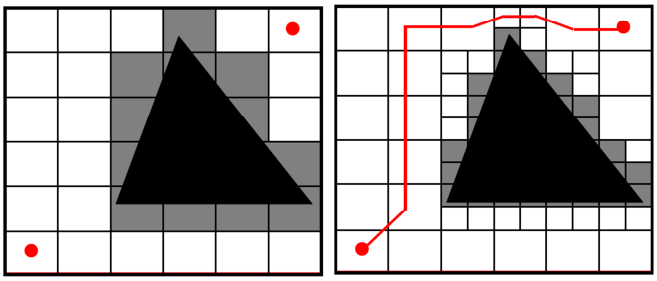
- Define a discrete grid in C-Space
- Mark any cell of the grid that intersects $\mathcal{C}_{obs}$ as blocked
- Find path through remaining cells by using (for example) A* (e.g., use Euclidean distance as heuristic)
- Cannot be *complete* as described so far. Why?

Not complete because we mark some open space as blocked. Therefore we can potentially mark the only path to the goal as blocked, even when it is open.



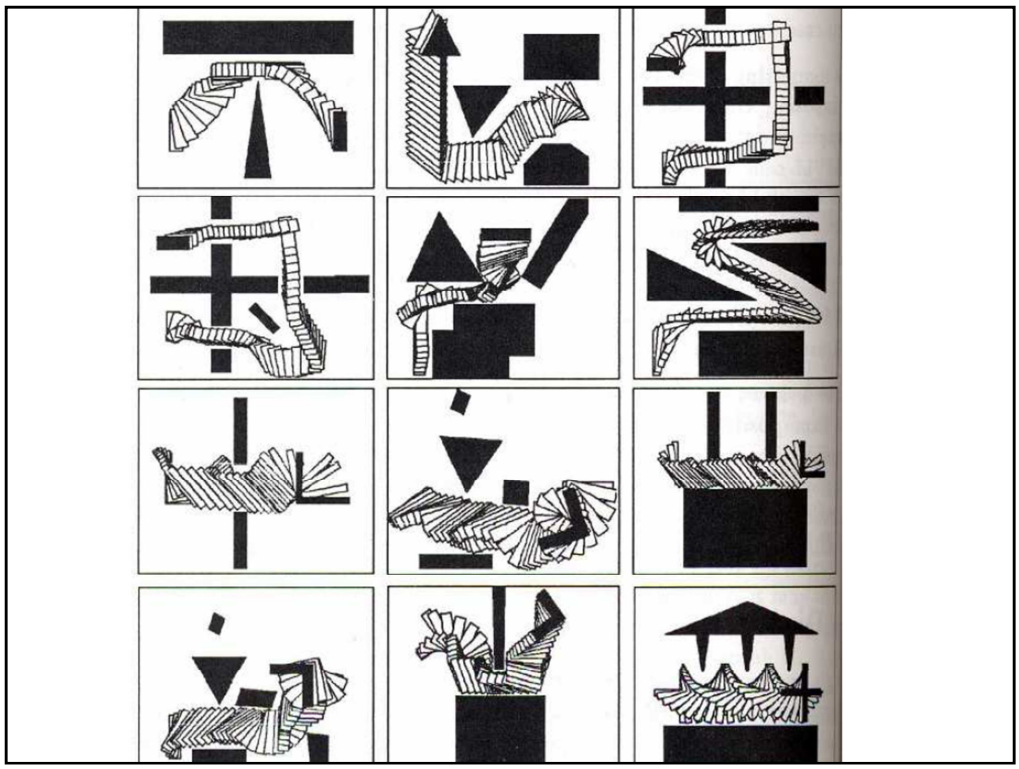
Subdivide cells that are only partially covered by an obstacle. No matter how small you make the grid, it still won't be complete, because there's a tiny continuous spot you're missing.

Approximate Cell Decomposition



- Cannot find a path in this case even though one exists
- Solution:
- Distinguish between
  - Cells that are entirely contained in $\mathcal{C}_{obs}$ (*FULL*) and
  - Cells that partially intersect $\mathcal{C}_{obs}$ (*MIXED*)
- Try to find a path using the current set of cells
- If no path found:
  - Subdivide the *MIXED* cells and try again with the new set of cells

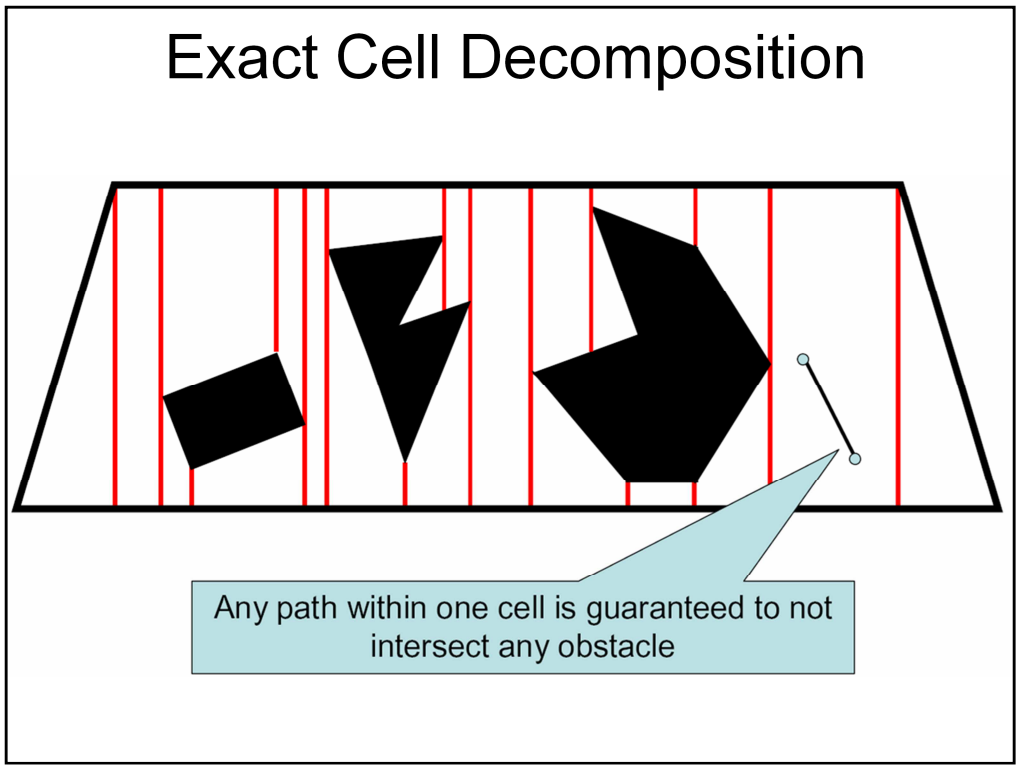
You only need to refine granularity in the gray mixed cells. Probably need some stopping condition for when there is no path possible. Easy to define when you have a robot with a radius...your robot can't go through a gap smaller than its diameter.



Approximate Cell Decomposition: Limitations

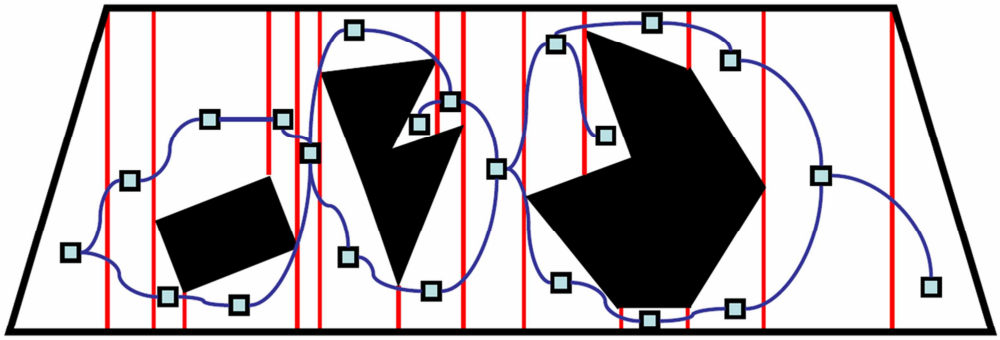
- Good:
  - Limited assumptions on obstacle configuration
  - Approach used in practice
  - Find obvious solutions quickly
- Bad:
  - Trade-off completeness/computation
  - Still difficult to use in high dimensions

In 3d subdivide into 8, in 4d subdivide into 16, etc.



Create strangely shaped cells that don't intersect with any obstacles.

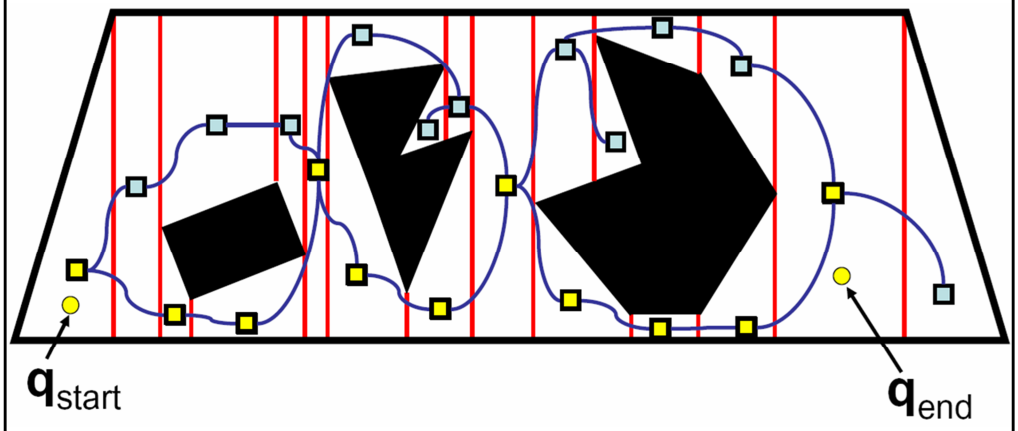
Exact Cell Decomposition



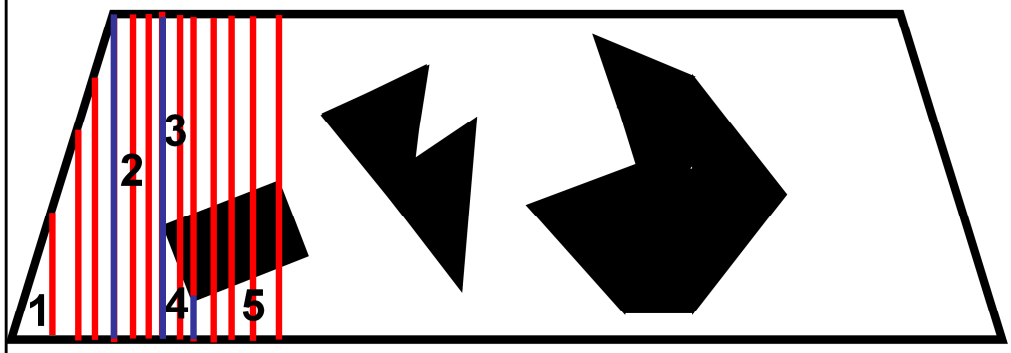
- Define a graph – roadmap – using the cells

Nodes in cells are chosen such that they can connect to their neighboring cells' nodes.

Exact Cell Decomposition



- The graph can be used to find a path between any two configurations



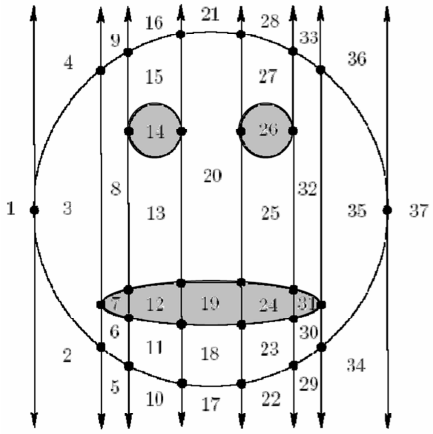
Critical event:
Create new cell

Critical event:
Split cell

Plane Sweep algorithm

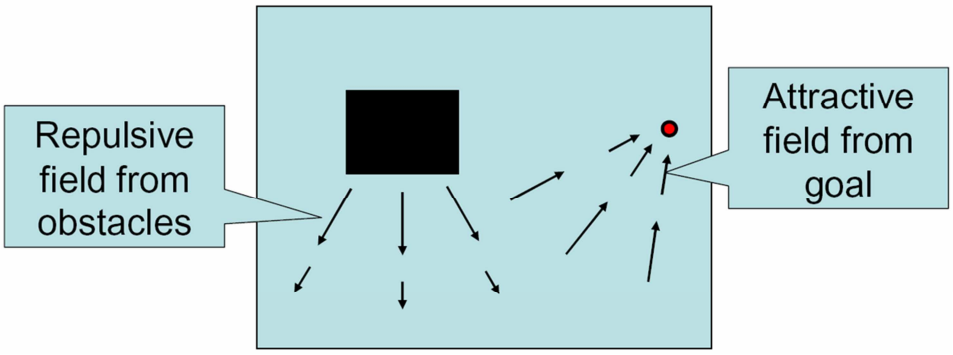
- Initialize current list of cells to empty
- Order the vertices of $\mathcal{C}_{obs}$ along the *x* direction
- For every vertex:
  - Construct the plane at the corresponding *x* location
  - Depending on the type of event:
    - Split a current cell into 2 new cells OR
    - Merge two of the current cells
  - Create a new cell
- Complexity (in 2-D):
  - Time: O(*N* log *N*)
  - Space: O(*N*)

Exact Cell Decomposition



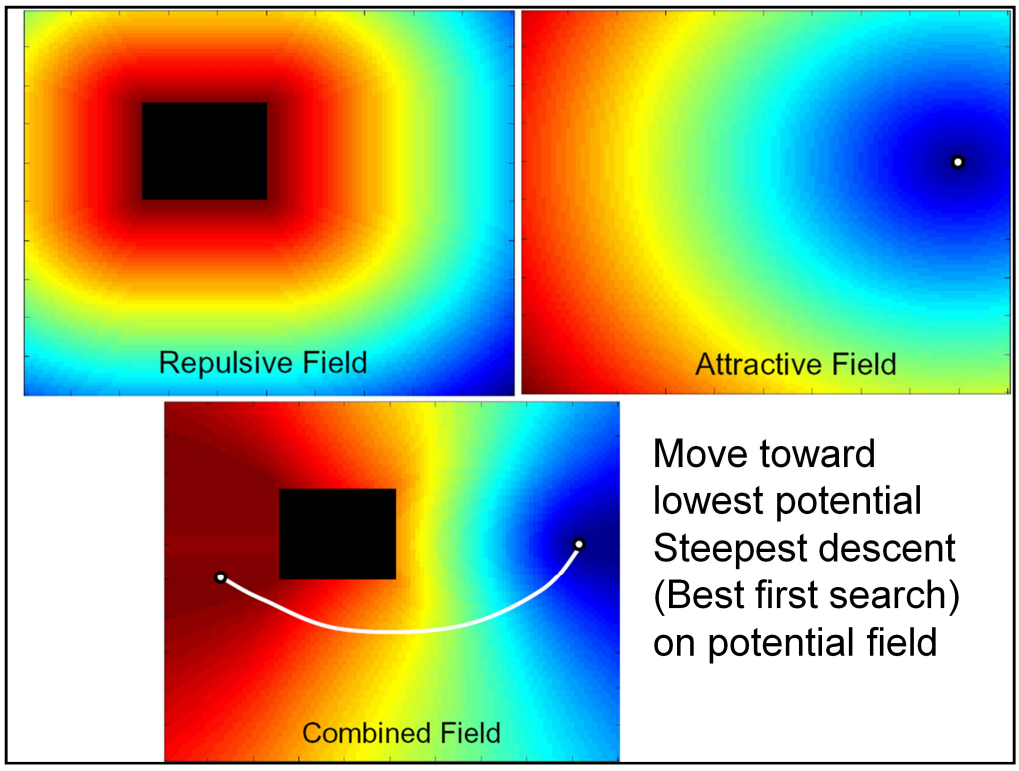
- A version of exact cell decomposition can be extended to higher dimensions and non-polygonal boundaries ("cylindrical cell decomposition")
- Provides exact solution → completeness
- Expensive and difficult to implement in higher dimensions

Potential Fields



- Stay away from obstacles: obstacles generate a *repulsive* field
- Move closer to the goal: goal location is a particle that generates an *attractive* field

Good for dynamic environments


Repulsive Field
Attractive Field
Combined Field
Move toward
lowest potential
Steepest descent
(Best first search)
on potential field

$$U_g(\mathbf{q}) = d^2(\mathbf{q}, \mathbf{q}_{goal})$$

Distance to goal state

$$U_o(\mathbf{q}) = \frac{1}{d^2(\mathbf{q}, Obstacles)}$$

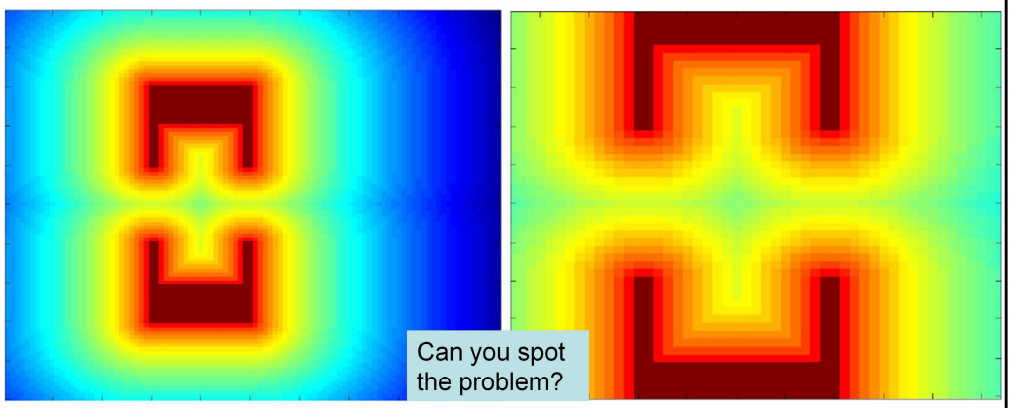
Distance to nearest obstacle point.
Note: Can be computed efficiently by using the *distance transform*

$$U(\mathbf{q}) = U_g(\mathbf{q}) + \lambda U_o(\mathbf{q})$$

λ controls how far we stay from the obstacles

Trying to tweak these parameters is hard– you get some dramatic changes (hooray for nonlinearity)

Potential Fields: Limitations



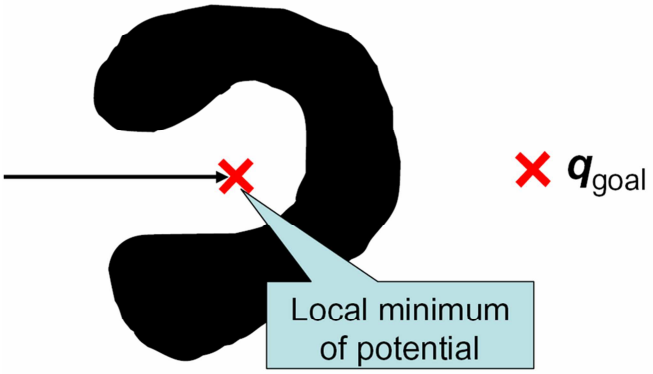
Potential field

Zoomed in view

- Completeness?
- Problems in higher dimensions

You can get stuck in the middle of the obstacles. Can't get to the blue from the center of this obstacle.

Local Minimum Problem



- Potential fields in general exhibit local minima
- Special case: Navigation function
  - $U(\boldsymbol{q}_{goal}) = 0$
  - For any $\boldsymbol{q}$ different from $\boldsymbol{q}_{goal}$, there exists a neighbor $\boldsymbol{q}'$ such that $U(\boldsymbol{q}') < U(\boldsymbol{q})$

Less of a problem in dynamic world. If you get stuck in a local minimum, the world changes and you are no longer in a local minimum.

Getting out of Local Minima I

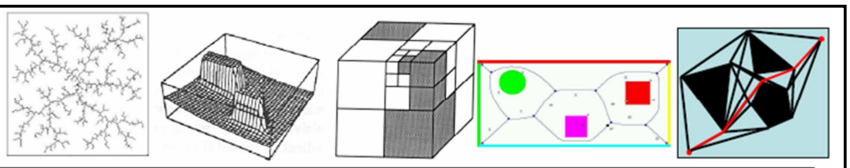
- Repeat
  - If $U(\boldsymbol{q}) = 0$ return Success
  - If too many iterations return Failure
  - Else:
    - Find neighbor $\boldsymbol{q}_n$ of $\boldsymbol{q}$ with smallest $U(\boldsymbol{q}_n)$
    - If $U(\boldsymbol{q}_n) < U(\boldsymbol{q})$ OR $\boldsymbol{q}_n$ has not yet been visited
      - Move to $\boldsymbol{q}_n$ ($\boldsymbol{q} \leftarrow \boldsymbol{q}_n$)
      - Remember $\boldsymbol{q}_n$

May take a long time to explore region "around" local minima

Getting out of Local Minima II

- Repeat
  - If $U(\boldsymbol{q}) = 0$ return Success
  - If too many iterations return Failure
  - Else:
    - Find neighbor $\boldsymbol{q}_n$ of $\boldsymbol{q}$ with smallest $U(\boldsymbol{q}_n)$
    - If $U(\boldsymbol{q}_n) < U(\boldsymbol{q})$
      - Move to $\boldsymbol{q}_n$ ($\boldsymbol{q} \leftarrow \boldsymbol{q}_n$)
    - Else
      - Take a random walk for T steps starting at $\boldsymbol{q}_n$
      - Set $\boldsymbol{q}$ to the configuration reached at the end of the random walk

Similar to stochastic search and simulated annealing: We escape local minima faster



	Sampling	Potential Fields	Approx. Cell Decomposition	Voronoi	Visibility
Practical in ~2-D or 3-D	Y	Y	Y	Y	Y
Practical in >> 2-D or 3-D	Y	Y (using randomized version)	??	N	N
Fast	Y	Y	Y	In low dim.	In 2-D
Online Extensions					N
Complete?	Probabilistically complete	Probabilistically-resolution complete	Resolution-Complete	Y	Y

More exact/Complete

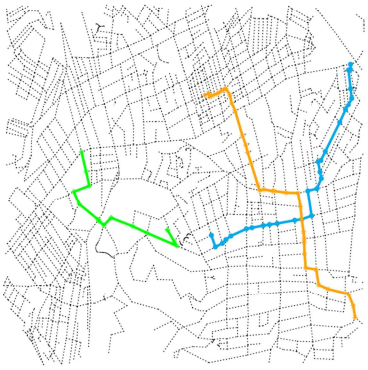
Faster/More practical in high dim.

Humans?

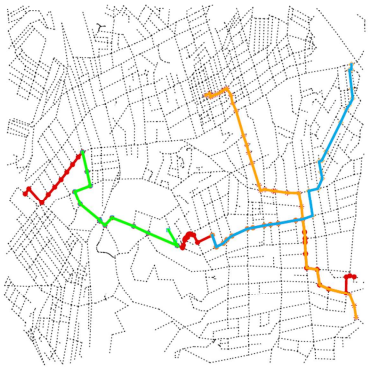
What do you think humans do?

Planning by analogy….

Example - Route Planning



A particular time/day situation:
Relevant cases for b-m0



b-m0 solved by Analogy:
red shows the extra planning done;
the three cases are merged with extra planning;
irrelevant parts of the cases are not used.

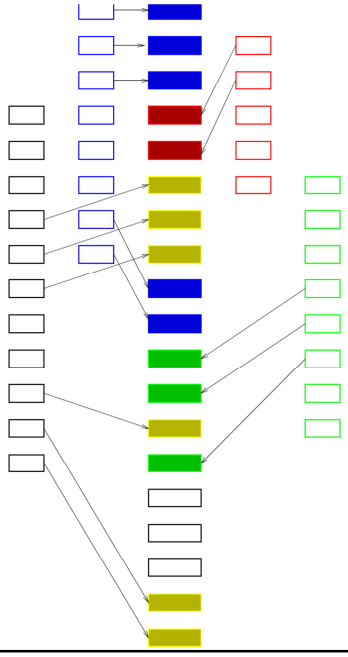
(*Haigh & Veloso - ICCBR'95, Haigh, Shewchuk & Veloso - JETAI'97*)

The subway system, for example, is a discretization.

Route Planning

- Routes are accumulated in a case library.
- Routes are abstracted and **indexed** according to situational parameters, such as: time of the day, day of the week, and driver.
- Geometric features are used by the **similarity metric** used at **retrieval** time.
- **Multiple routes are merged** at planning time.
- Planning cases are integrated with generative planning.
- Relevant parts of the cases are validated, pursued and merged.
- Generative planner does **any extra planning** work needed to merge the planning cases.

Replaying Multiple Cases



Merge together different parts of different paths. Blank spots show additional planning you need to do.

Summary – Points to Know

- Visibility graphs
- Voronoi diagrams
- Cell decomposition
- Potential fields
- Route planning by analogy