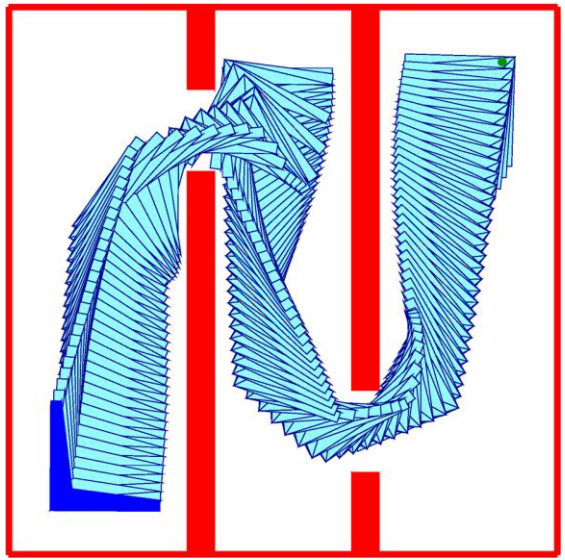
COMS W4733: Computational Aspects of Robotics

Lecture 16: Roadmap Path Planning



Instructor: Tony Dear

Logistics

- Midterm grades and solutions released over the break
- HW4 released after class; includes reading and programming components
  - Reading component completed individually
  - Programming (ROS) may be done in pairs
  - Reminder: Remaining homeworks are 15% each, no final exam!
- Today: Roadmap path planning methods
- C-space, visibility graphs, Voronoi diagrams

Path Planning So Far

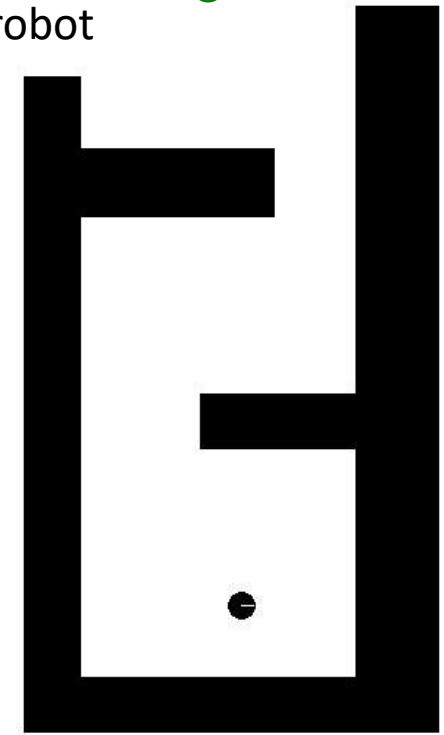
- If no obstacles, connect initial and final configurations using *continuous functions*
- May include velocities and acceleration conditions
- Planning may be done in joint space or workspace

- If obstacles, *bug algorithms* provide incremental and reactive approach
- Generally, head toward goal; make decisions to change course on the fly

- No knowledge of obstacles in advance
- Robot assumed to be a particle

Workspace of Finite-Sized Robots

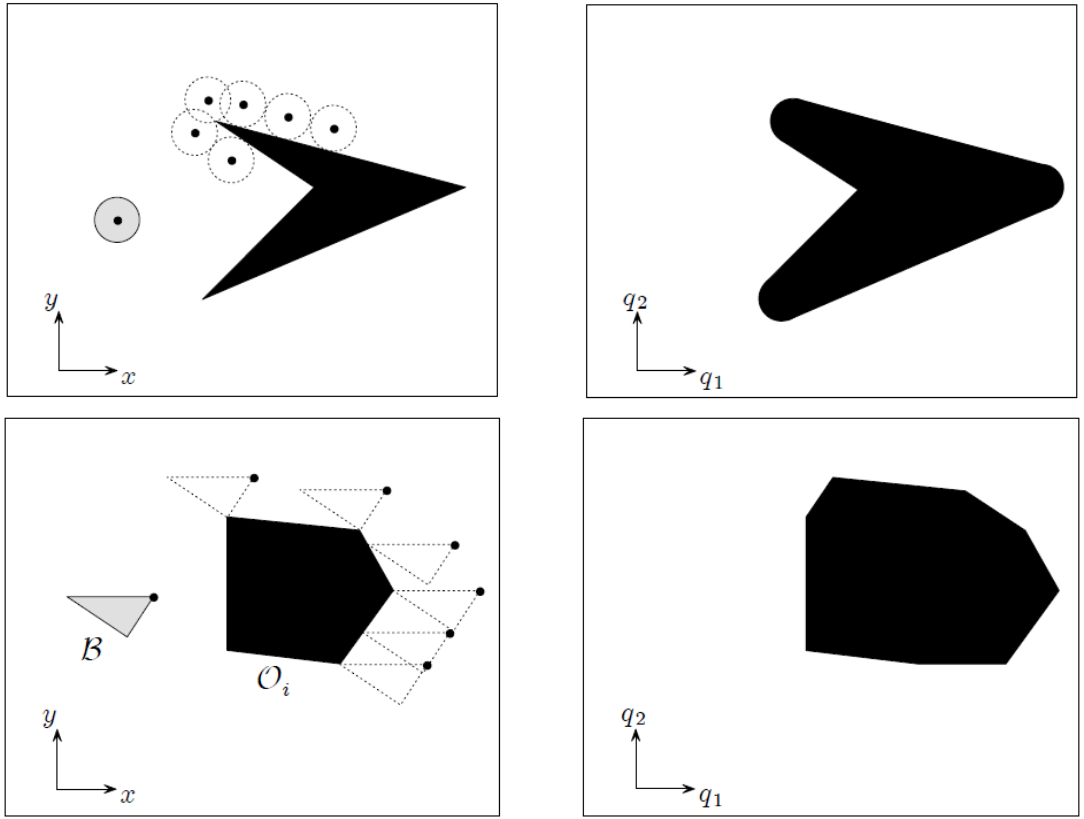
Circular robot



"Pad" obstacles and reduce robot to particle again

Same robot, now a particle!

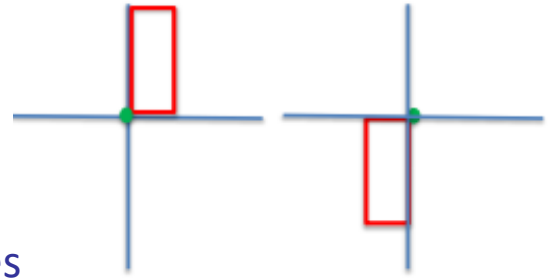
Workspace of Finite-Sized Robots

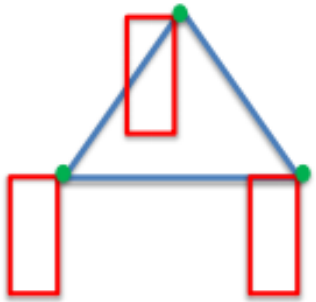


Growing Obstacles

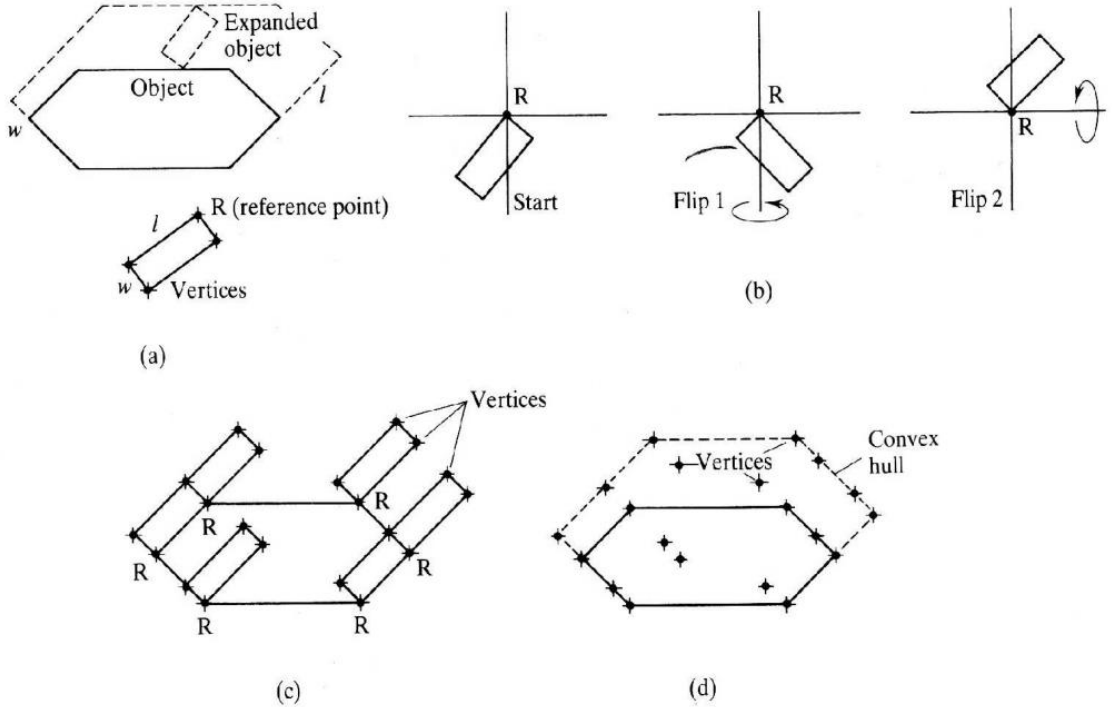
- We can “grow” any obstacle without actually moving our robot around it

1. Reflect the robot about its reference point
2. Place reference point of reflected robot at obstacle vertexes
3. Compute convex hull of all of the placed robots
4. Convex hull is the grown obstacle





Growing Obstacles



Configuration Space

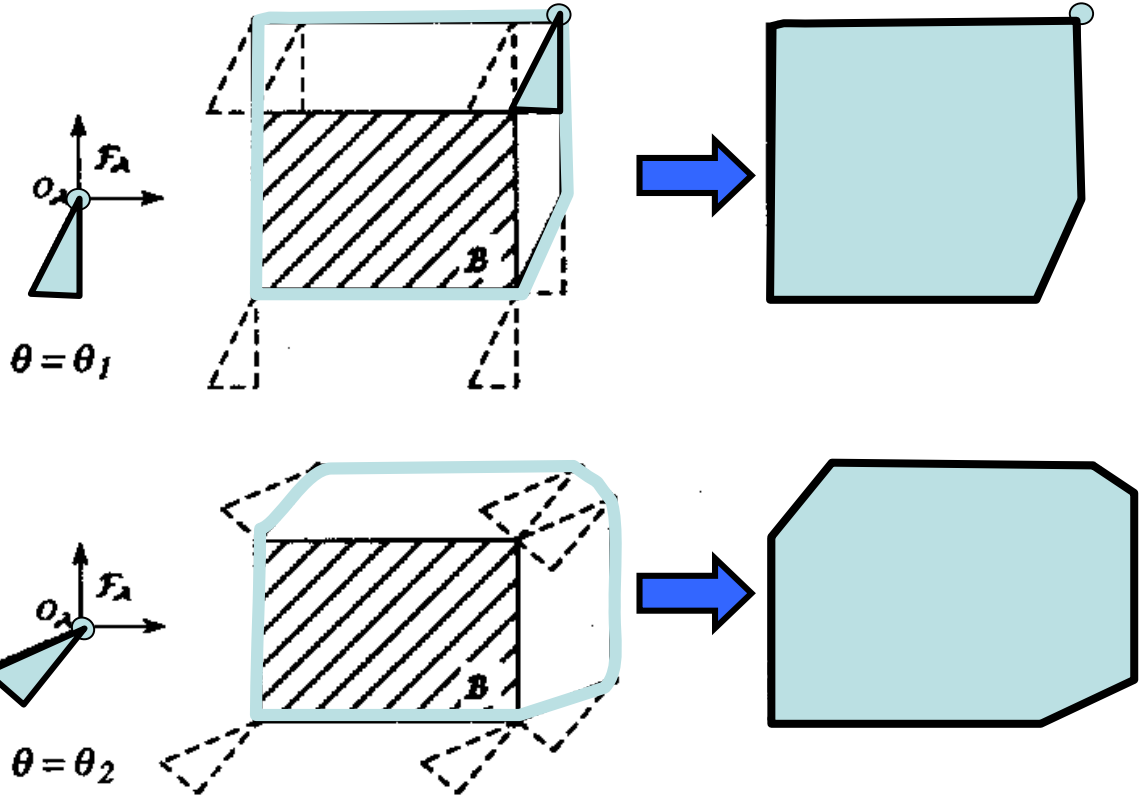
- In prior examples, robot had been reduced to a particle after constructing the padded **C-obstacles** by hugging robot around each original obstacle

- Robot's configuration space (**C-space**) very similar to its workspace

- What if robot can rotate in addition to translate?
- In the plane, C-space is now $\mathrm{SE}(2) = \mathbb{R}^2 \times S$ instead of just $\mathbb{R}^2$

- Orientation (θ) adds a third dimension to current workspace picture!

Robots in SE(2)



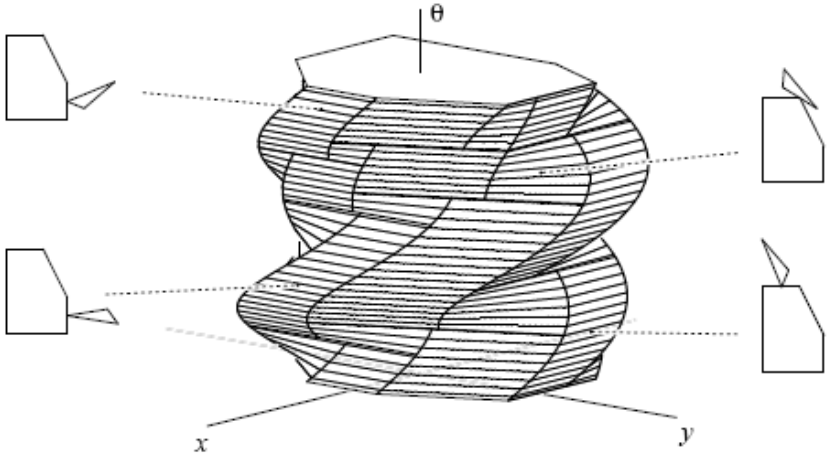
Robots in SE(2)

- Stacking together the Cartesian C-spaces at different orientations θ gives us the complete SE(2) C-space

- Path planning now amounts to finding an obstacle-free path for a particle between an initial and goal configuration

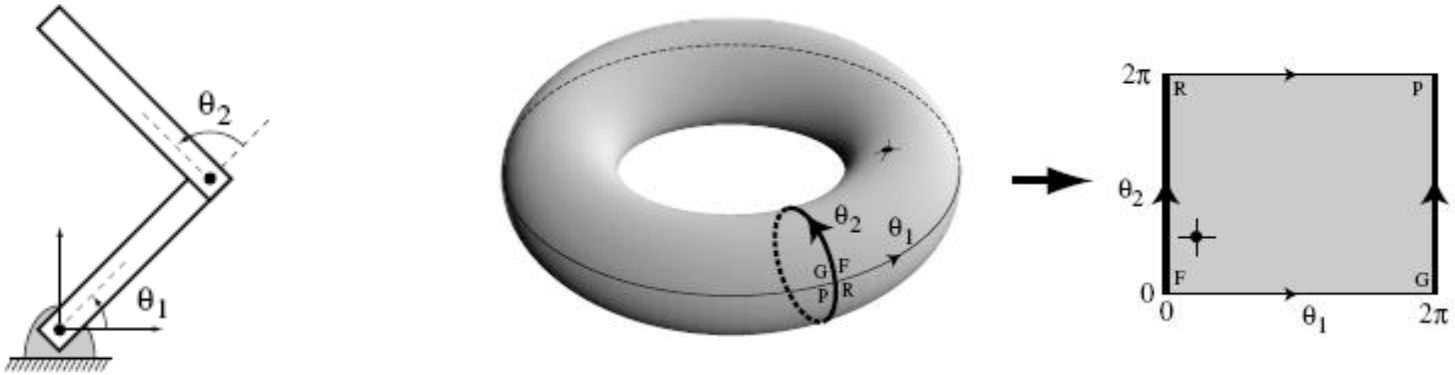
- Can combine all polygons into one

- Can try to plan in the higher-dim space



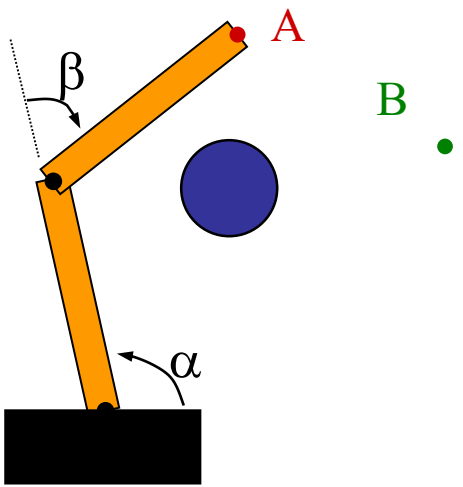
C-Space of Manipulators

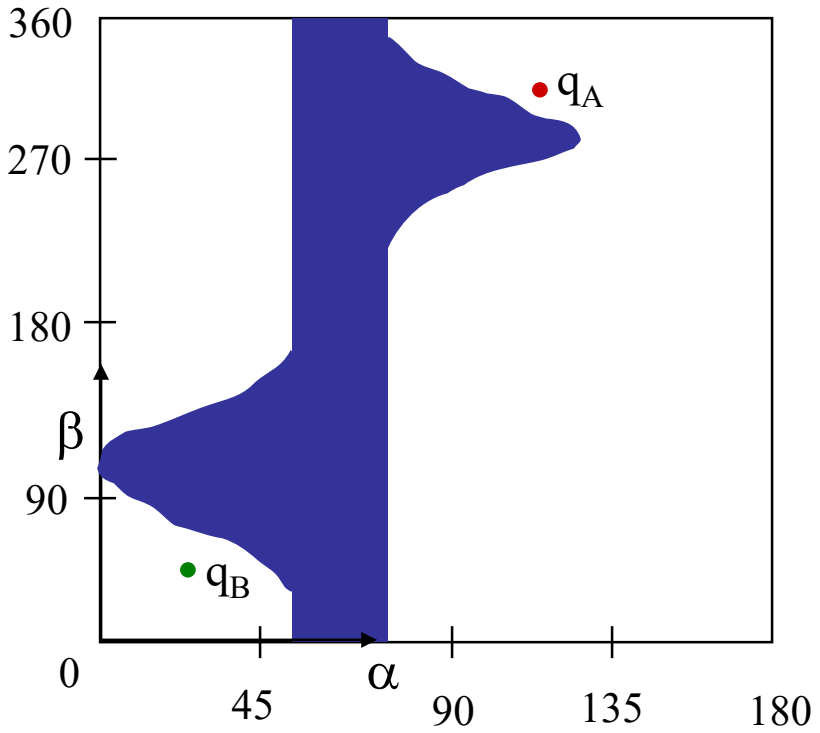
- If no joint limits, C-space of a RR arm is composed of $(\theta_1, \theta_2) \in S^1 \times S^1$
- Not Cartesian at all—in fact, true representation is a torus!
- We can also "flatten" it into a Cartesian representation



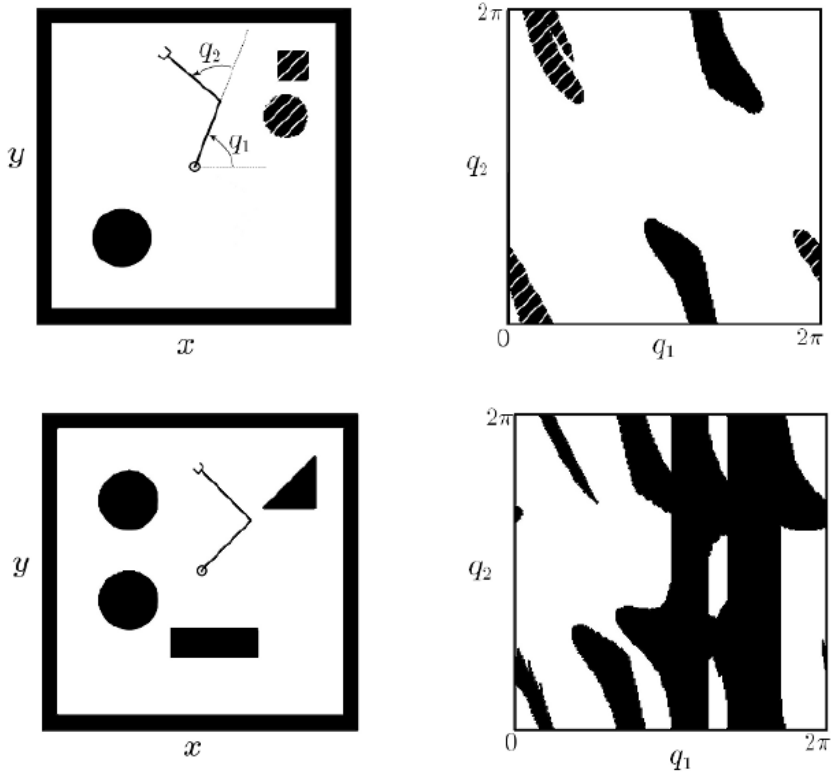
C-Space of Manipulators

- What happens if there is an obstacle in the manipulator's workspace?
- Obstacle has an equivalent representation in the C-space!





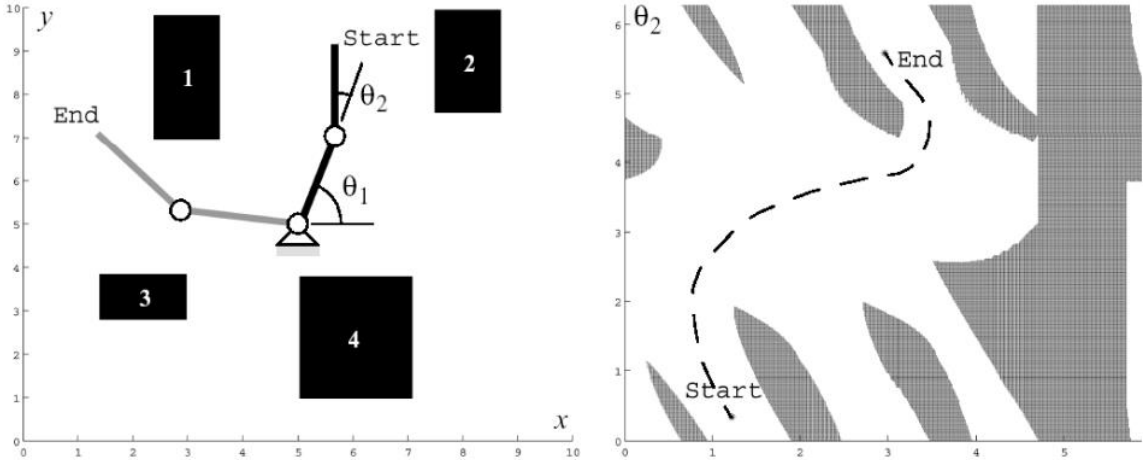
Manipulator C-Space Examples



http://www.cs.unc.edu/~jeffi/c-space/robot.xhtml

Path Planning in C-Space

- Once we have constructed the C-space, robot is just a particle again
- Place starting and ending configurations in the C-space
- Depending on the problem, desired configurations may be a set of points
- Use planning algorithm to find collision-free path

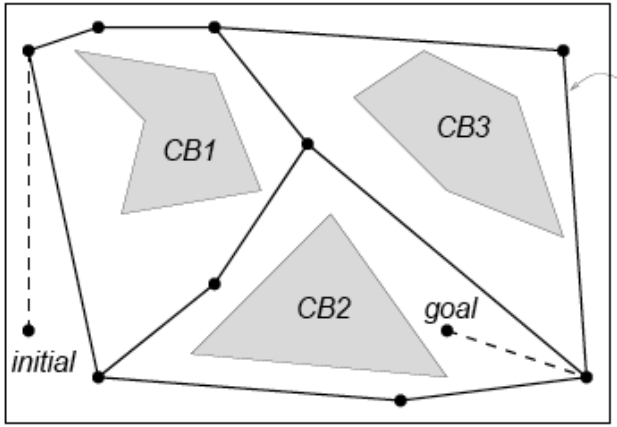


Roadmap Path Planning

- Since we have particle robots, we can again turn to reactive bug-like algorithms
- However, we now have knowledge of all obstacles in the space!
- We can do a lot better by planning in advance before we move

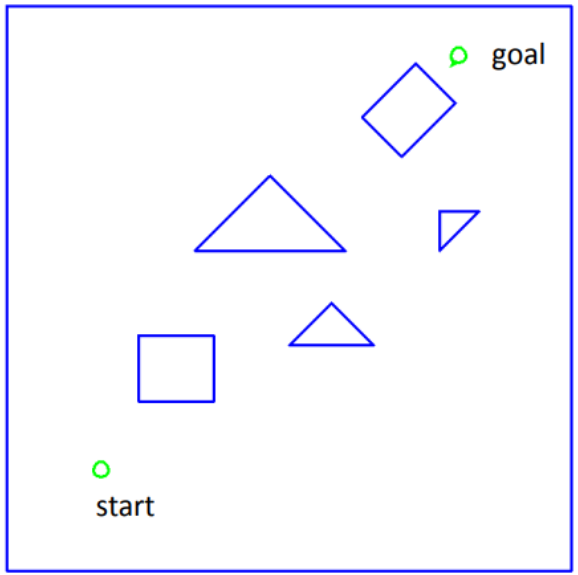
- Idea: Build a **roadmap**, or a network of paths in the free workspace connecting initial and goal configurations

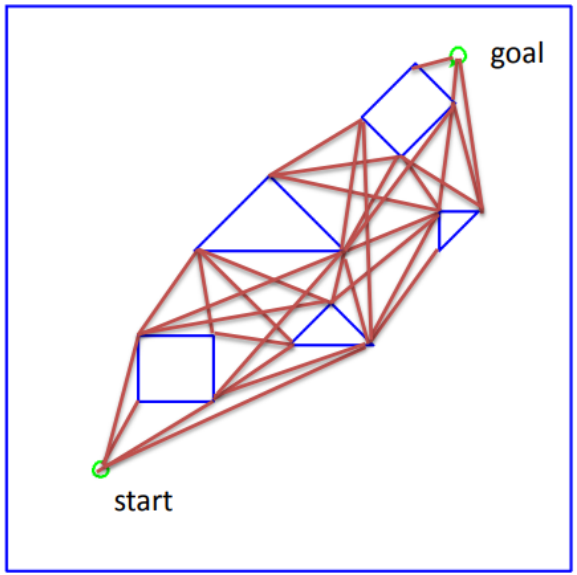
- How to find the set of nodes?
- How to find the set of edges?
- How to find a best path?



Visibility Graphs

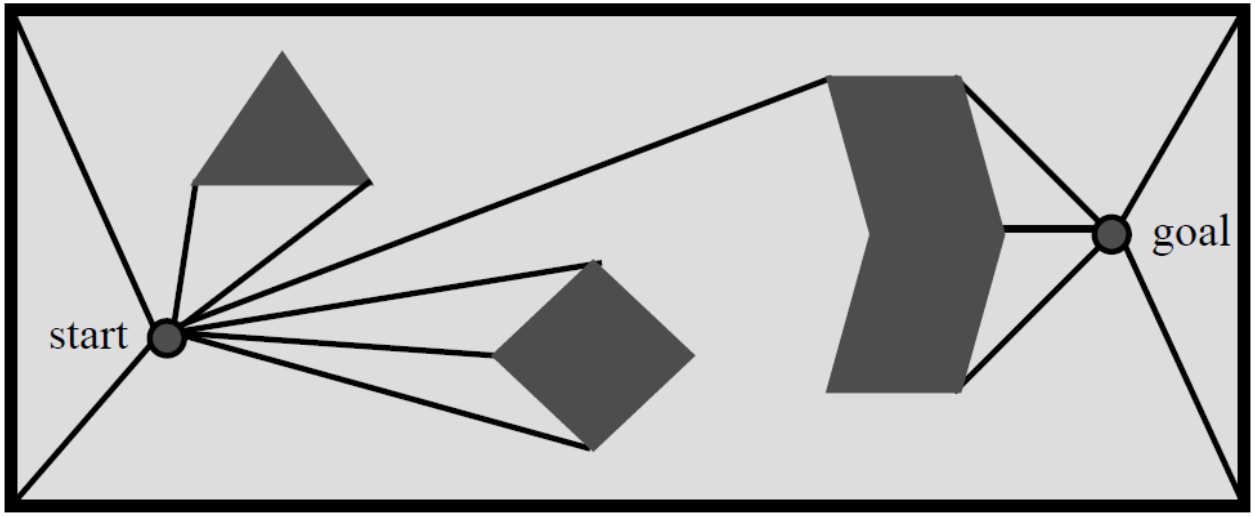
- If all obstacles are polygonal, their vertices are a natural choice for potential nodes
- Edges can be placed between nodes that are “visible” from each other
- A* or your favorite graph search algorithm can then be used to find a best path





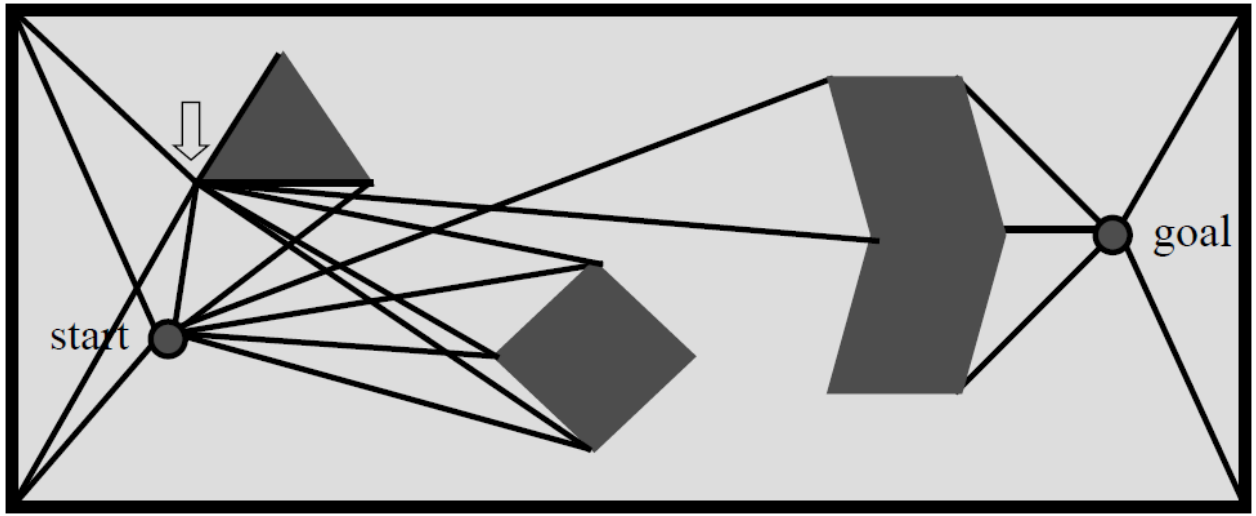
VGRAPH Algorithm

- To start, place edges from start and goal to all “visible” obstacle vertices and world corners



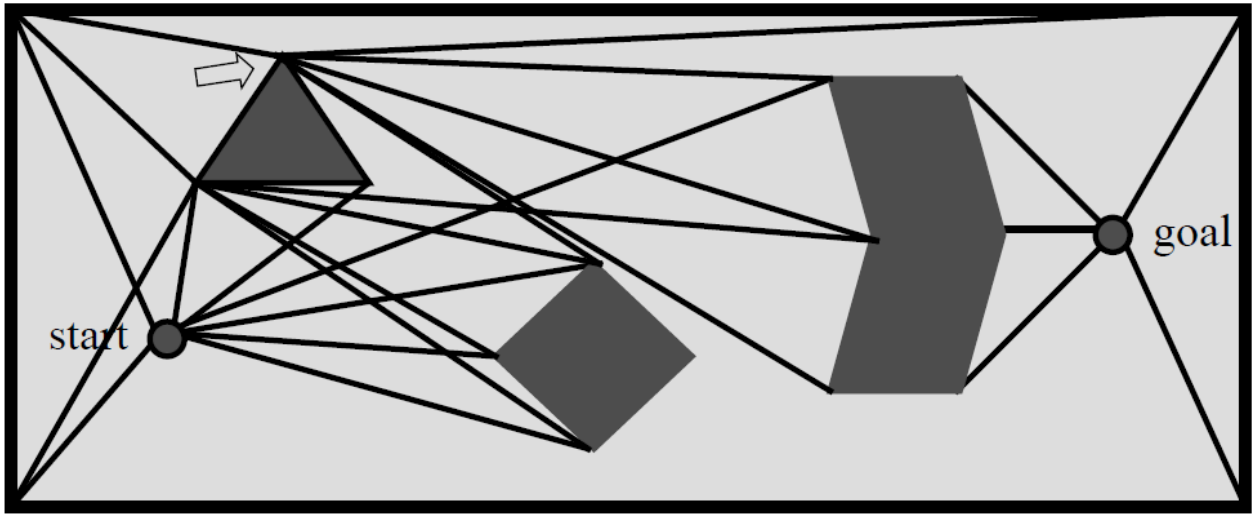
VGRAPH Algorithm

- From each polygonal vertex, insert edges to all other visible polygonal vertices, including those lying on the same polygon



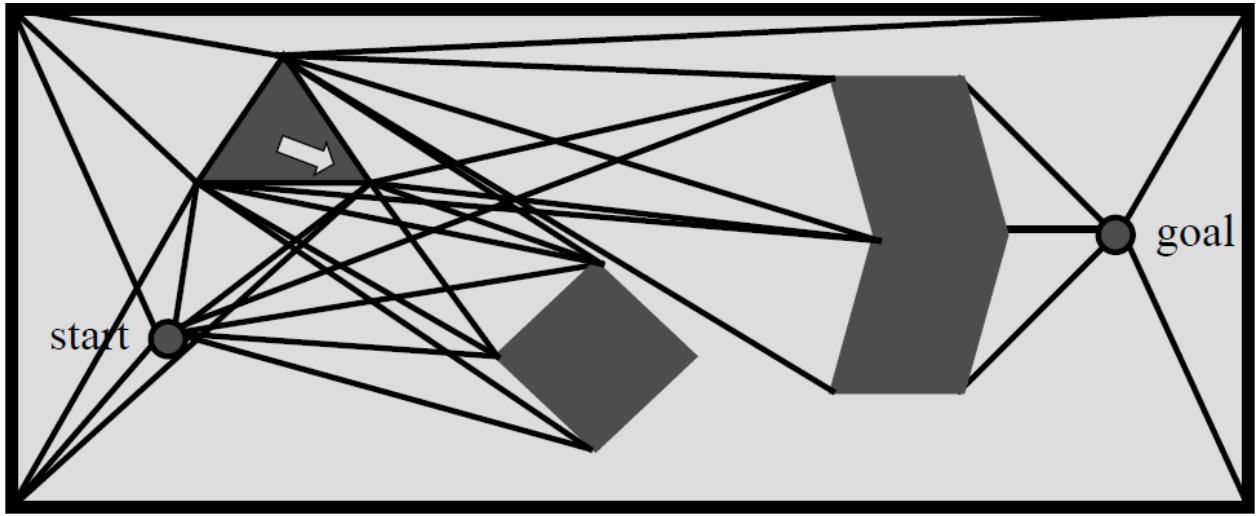
VGRAPH Algorithm

- From each polygonal vertex, insert edges to all other visible polygonal vertices, including those lying on the same polygon



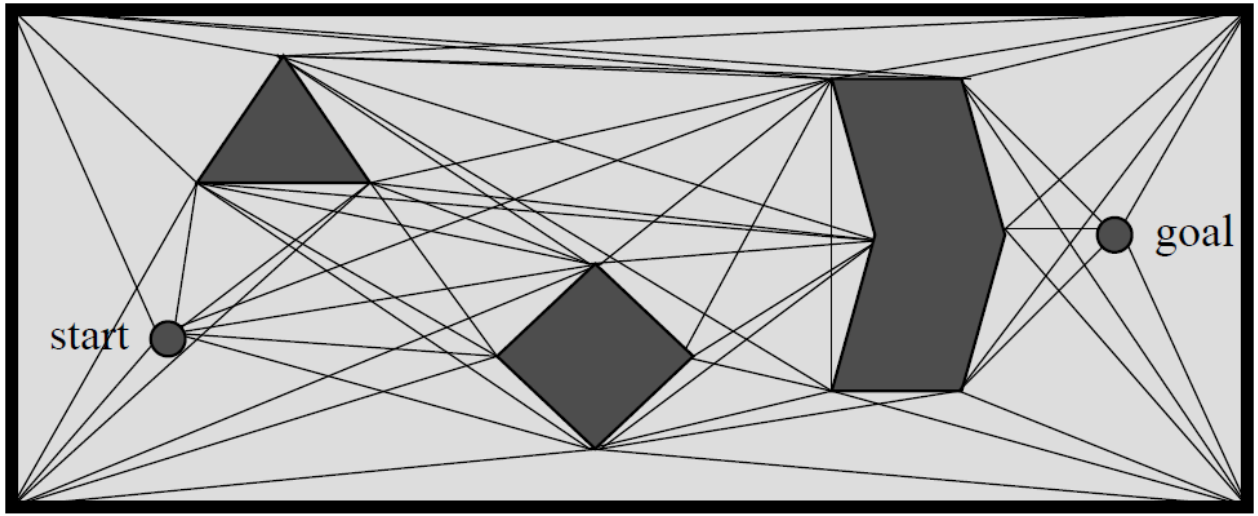
VGRAPH Algorithm

- From each polygonal vertex, insert edges to all other visible polygonal vertices, including those lying on the same polygon



VGRAPH Algorithm

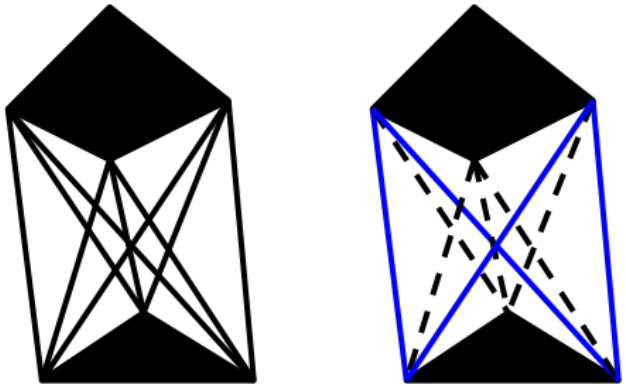
- Repeat until done
- Now graph search can be run on the resulting graph with assumed costs



Reduced Visibility Graphs

- Somewhat slow; algorithm runs in time ~ O(n^3)
- Shortest path found on the visibility graph is guaranteed to be shortest path possible
- But all paths still end up hugging closely to obstacles—no room for error

- We can potentially reduce the number of edges inserted into the visibility graph
  - Edges to concave vertices
  - Edges that go “into” the obstacle



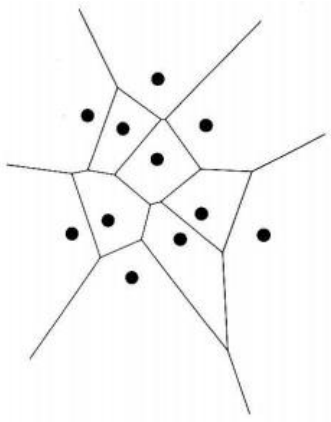
Voronoi Diagrams

- If robustness is preferred over optimality, then we want paths that keep the robot as far away from obstacles as possible

- **Voronoi diagrams** are a set of line segments that maximize distance from set of points

- One approach: Find perpendicular bisectors between all pairs
- Intersections of bisectors form vertices of the Voronoi diagram

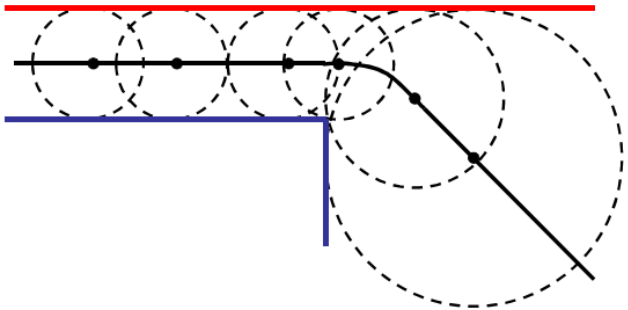
- More efficient algorithms can do this in time ~ O($n \log n$)

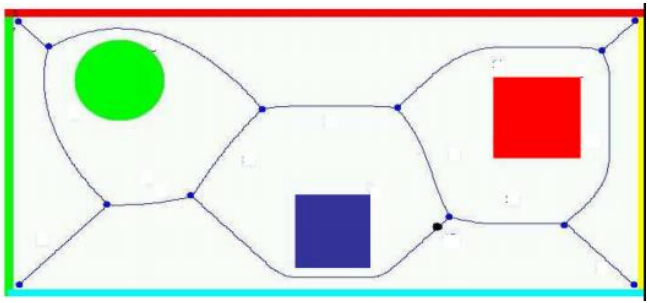


Generalized Voronoi Diagrams

- What about constructing Voronoi diagrams for polygons instead of points?
- One approach: Discretize polygon edges into sets of points and compute as before
  - Edges that lie inside polygons can be discarded in a second pass

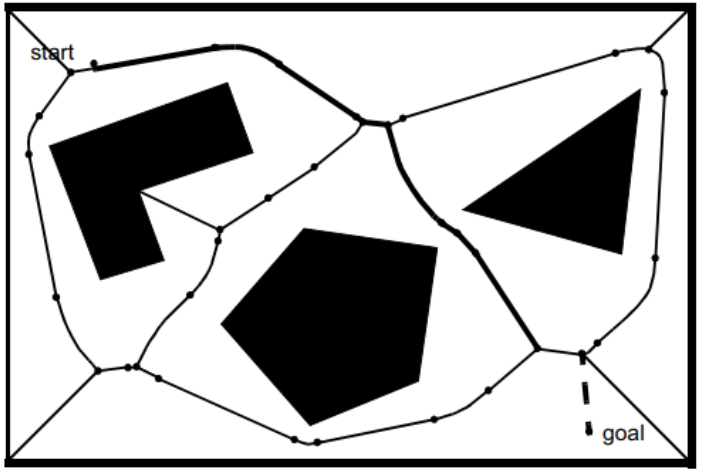
- Alternatively: Polygon boundaries are either straight lines or corners
- Points equidistant from two polygon edges lie on straight Voronoi edges
- Points equidistant from a polygon edge and a corner lie on a *curved* Voronoi edge



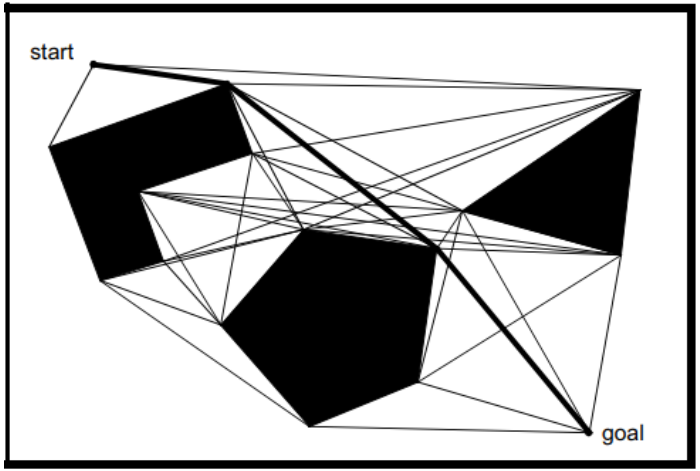


Planning on Generalized Voronoi Diagrams

- Once we have a diagram, the edges can be used to find a complete path
- Find *retractions* of start and goal locations on Voronoi edges
- Compute shortest path between the two corresponding points



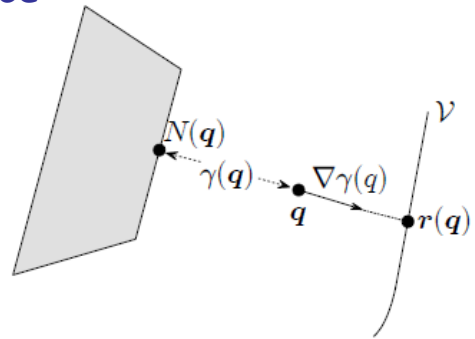
Voronoi planning



VGRAPH planning

Voronoi Planning Properties

- **Retraction** of a free point to Voronoi edge that increases distance from obstacles the fastest
- Mapping allows us to plan on Voronoi diagram only without worrying about free space

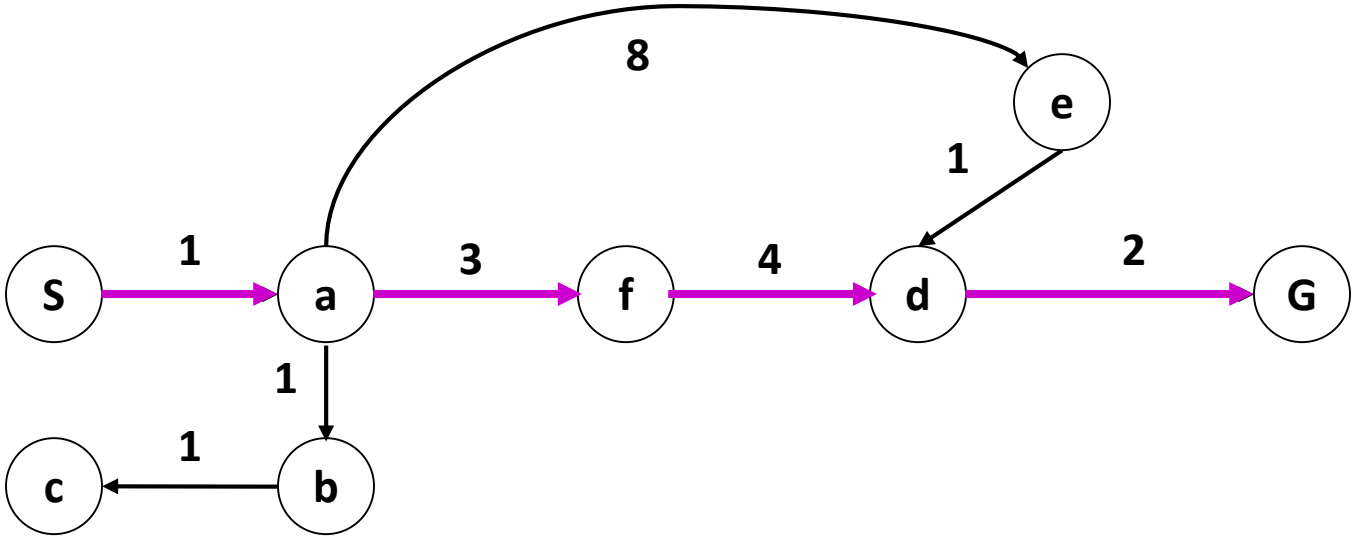


- Like VGRAPH, Voronoi planning is *complete*
- Can be faster than VGRAPH for repeated planning on same map

- Paths may be too conservative, leading to “sensor deprivation”
- Can be unstable even with small changes in obstacles

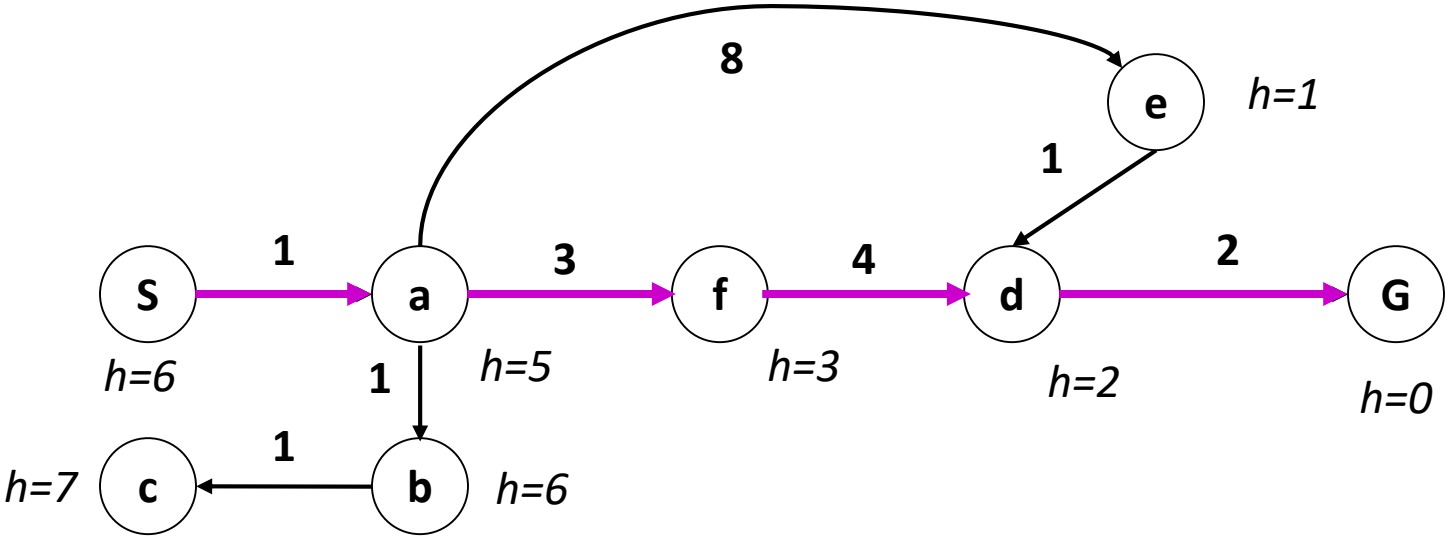
Review: Uninformed Search

- **Uninformed search** finds a path using only known information (no heuristics)
- Depth-first and breadth-first search do not use edge costs at all
- Uniform-cost search expands based on cheapest running cost up to current node, plus each possible new cost incurred upon expansion



Review: Informed Search

- For **informed search** we assume we have a *heuristic* as an estimate of cost-to-goal
- A* expands from current node based on cheapest sum of running (backward) cost and heuristic (forward) cost
- Guaranteed optimal if heuristic is admissible and consistent



Summary

- A robot's C-space may include (expanded) obstacles in different forms to indicate valid configurations for its particle representation
- Visibility graphs are formed by connecting obstacle vertices and then searched to find the optimal (shortest) paths
- Voronoi diagrams produces graphs that allow for maximal clearance between the robot and obstacles