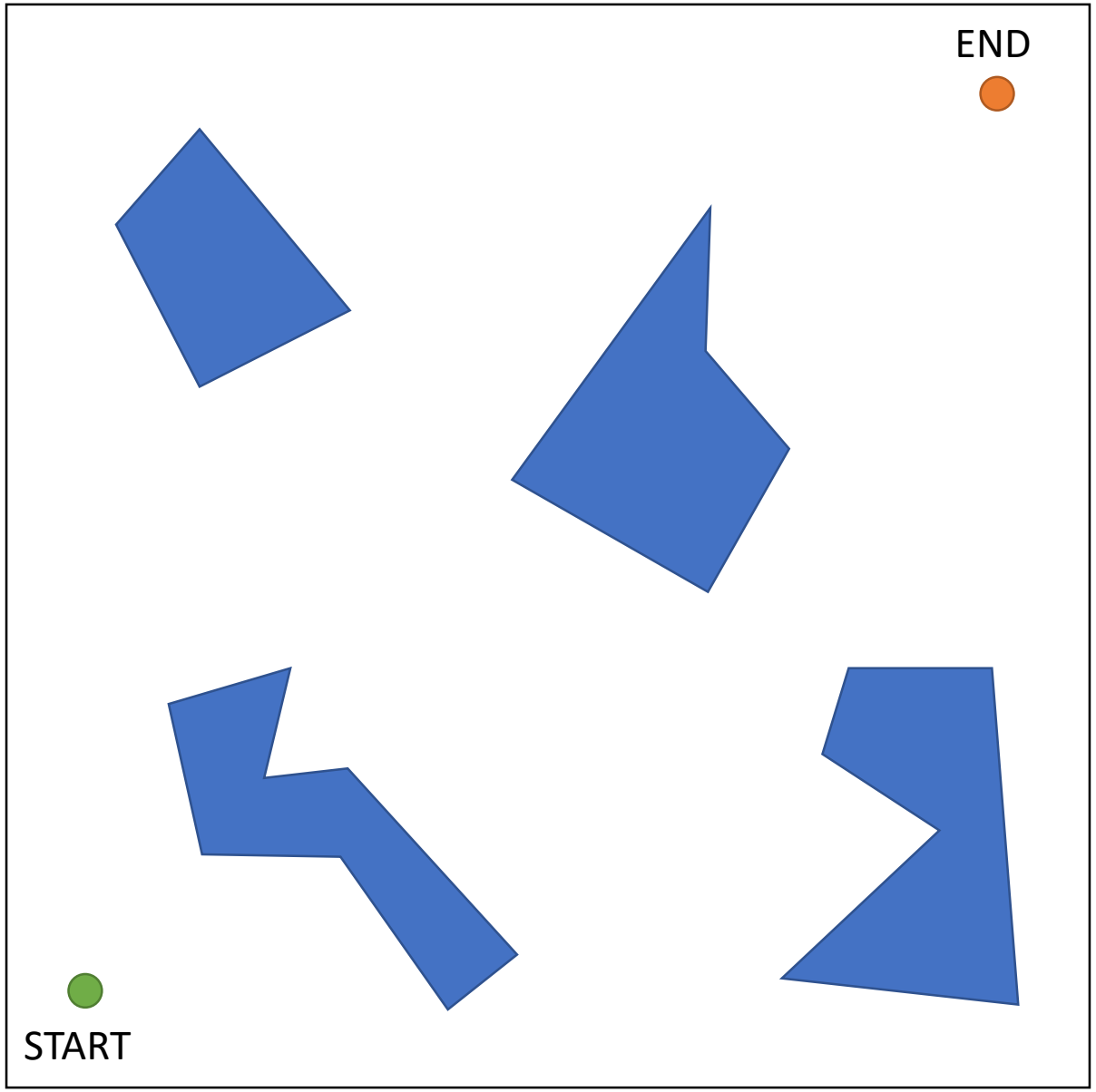
Amit Bouzaglo

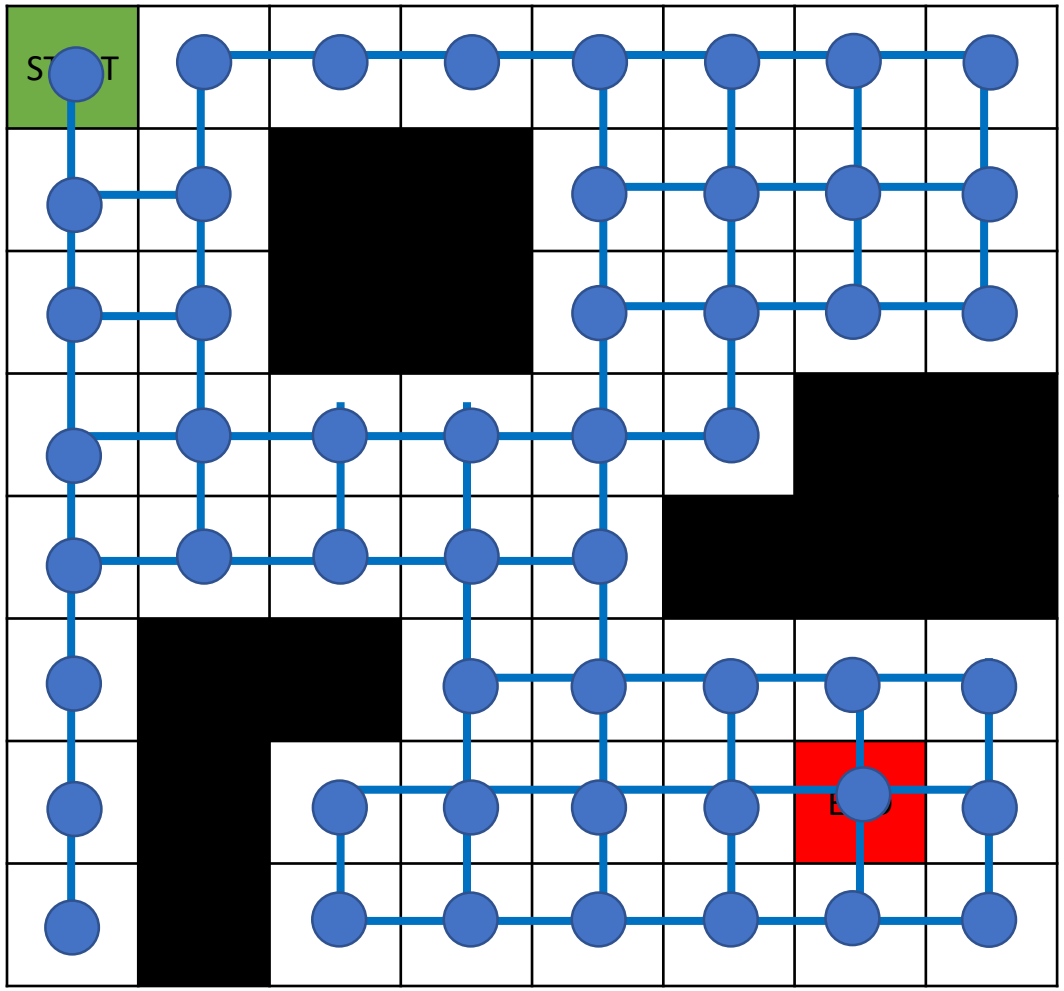

END
START

Autonomous Navigation

- Goal
  - Develop techniques that would allow a robot to automatically decide how to move from position one to another.
- Steps:
  - Localization (GPS, INR, sensors)
  - Mapping (obstacles, roads, POI)
  - **Path-planning**
  - Path-following

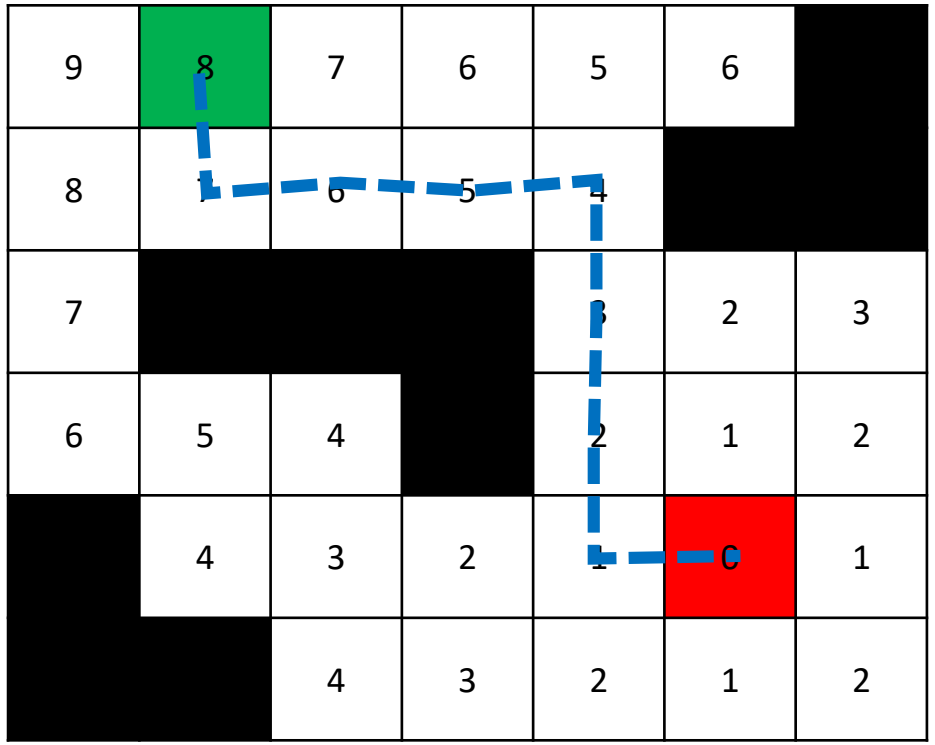
Mapping

- Grid to Graph
- Graph: $G(V, E)$
  - V – Vertexes (Nodes)
  - E – Edges



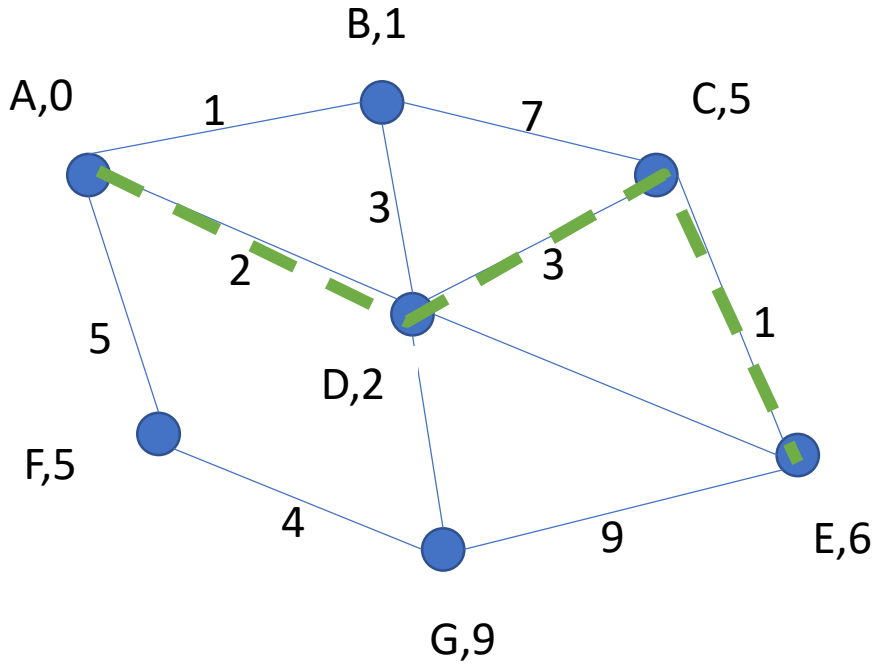
Grassfire Algorithm (BFS)

- For each node n in the graph:
  - $n.distance = \infty$
- Create empty list.
- $goal.distance = 0$, add goal to list.
- While list no empty:
  - Let current = first node in list.
  - Remove current from list.
  - For each node n, that is adjacent to current:
    - If $n.distance = \infty$:
      - $n.distance = current.distance + 1$.
      - Add n to the back of the list.
- **Trace a path:**
  - Start from Start.
  - Move to the neighbor with the smallest value.
  - Ties breaks arbitrarily.



Complexity = $O(|V|) = O(n * m)$

Dijkstra's Algorithm (BFS)

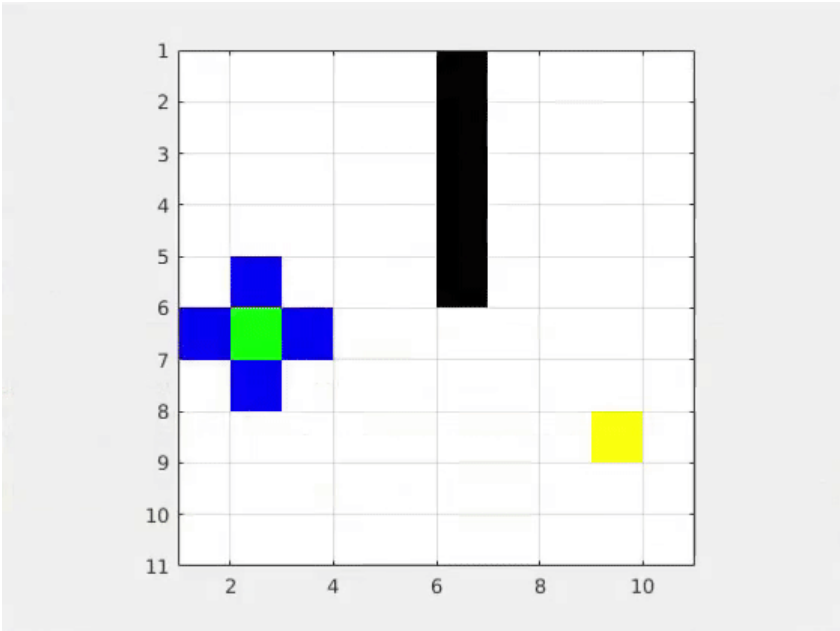


- For each node n in the graph:
  - $n.distance = \infty$
- Create an empty list.
- $start.distance = 0$, add start to list.
- While list no empty:
  - Let current = node n in the list with the smallest distance.
  - Remove current from list.
  - For each node n, that is adjacent to current:
    - If $n.distance > current.distance + length\ of\ edge\ from\ n\ to\ current$
      - $n.distance = current.distance + length\ of\ edge\ from\ n\ to\ current$
      - $n.parent = current$
      - Add n to list if it isn't there already
- **Trace a path:**
  - Start from destination.
  - Move from son to his parent.

Complexity = $\boldsymbol{O((|E| + |V|)log(|V|))}$

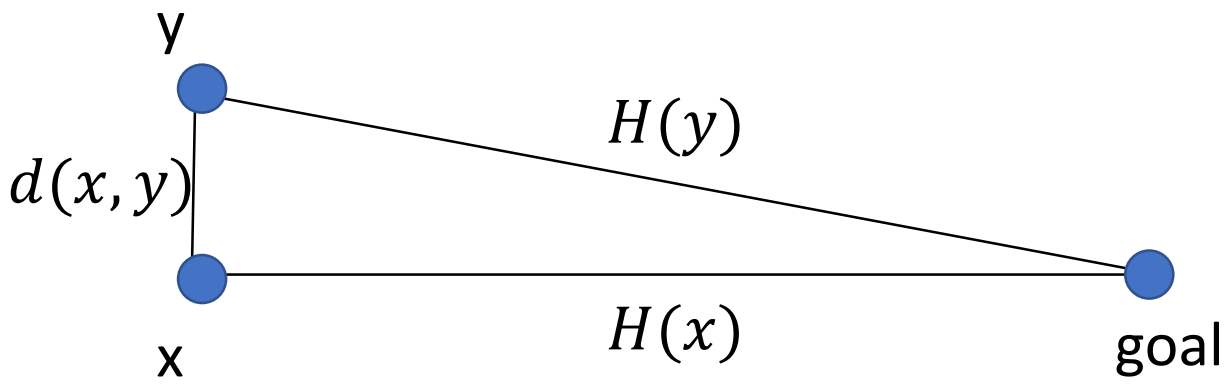
Quick summary

- Dijkstra/Grassfire Algorithm explores evenly in all directions until it finds the gold node.
- Could we do better?
  - YES!
  - Using the information we already have.
  - The destination…



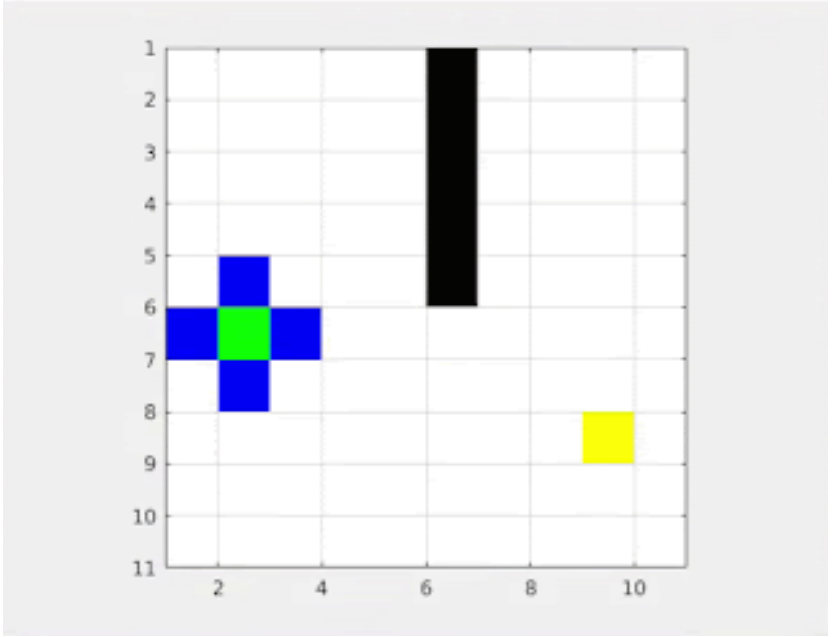
Heuristic function H(n)

- $H(n) =$ a non-negative value that is indicative of the distance from that node to the goal.
  - $H(goal) = 0$
  - $H(x) \leq H(y) + d(x, y)$
  - $d(x, y) = length\ of\ the\ x - y\ edge$
- Example of H function:
  - Euclidean distance - $H(x_n, y_n) = \sqrt{(x_n - x_g)^2 + (y_v - y_g)^2}$
  - Manhattan distance - $H(x_n, y_n) = |x_n - x_g| + |y_v - y_g|$



A* Algorithm (Best-FS)

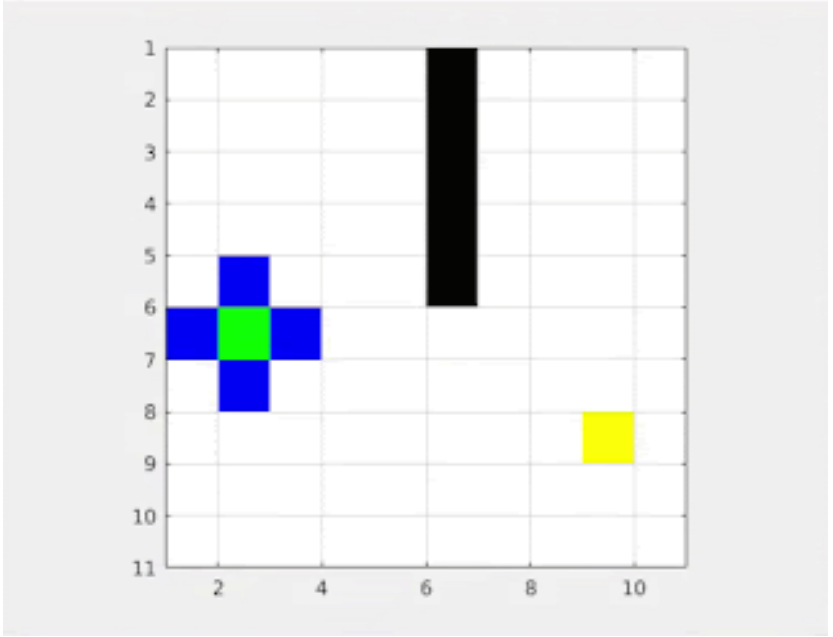
- For each node n in the graph:
  - $\mathrm{n.g} = \infty$ (the distance from n to start)
  - $n.f = \infty$ (g value + **estimate** distance to goal)
- Create an empty list.
- $start.g = 0$, start. f = H(start). add start to list.
- While list no empty:
  - Let current = node n in the list with the smallest f value.
  - Remove current from list.
  - If (current == goal node) – return success.
  - For each node n, that is adjacent to current:
    - If $(n.g > current.g + cost\ of\ edge\ from\ n\ to\ current.$
      - $n.g = current.g + cost\ of\ edge\ from\ n\ to\ current.$
      - $n.f = n.g + H(n)$
      - $n.parent = current$
      - Add n to list if it isn't there already
- **Trace a path:**
  - Start from destination.
  - Move from son to his parent.



Complexity = Dijkstra's complexity (worst case)

D* Algorithm

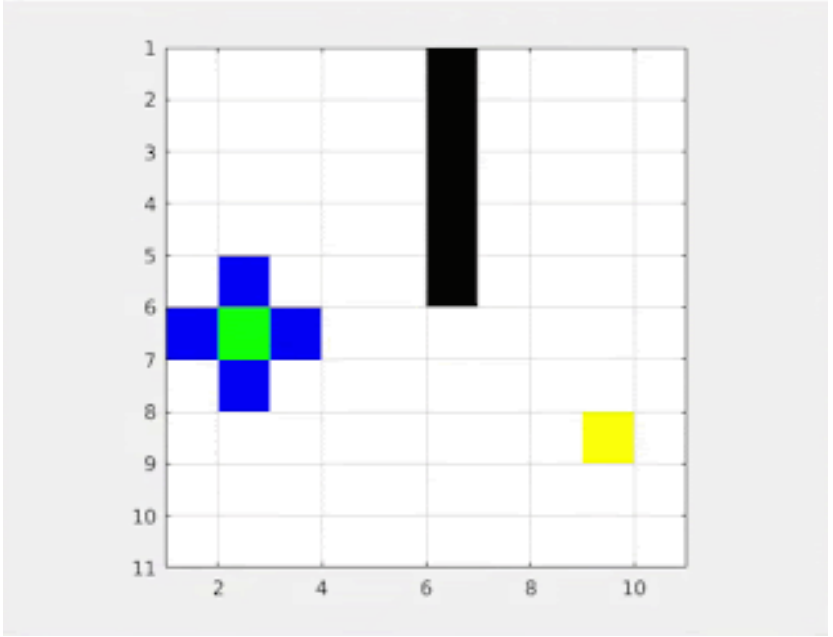
- For each node n in the graph:
  - $\mathrm{n.g} = \infty$ (the distance fro
  - $n.f = \infty$ (g value
- Create an empty list.
- $start.g = 0$, start
- While list no emp
  - Let current
  - Remove cu
  - If (current
  - For each no
    - If ($n.g$ $rent.$
      - $n.$ $rrent.$
      - $n.f$
      - $n.parent$
      - Add n to list if
- **Trace a path:**
  - Start from destination.
  - Move from son to his parent.



Complexity = Dijkstra's complexity (worst case)

D* lite Algorithm

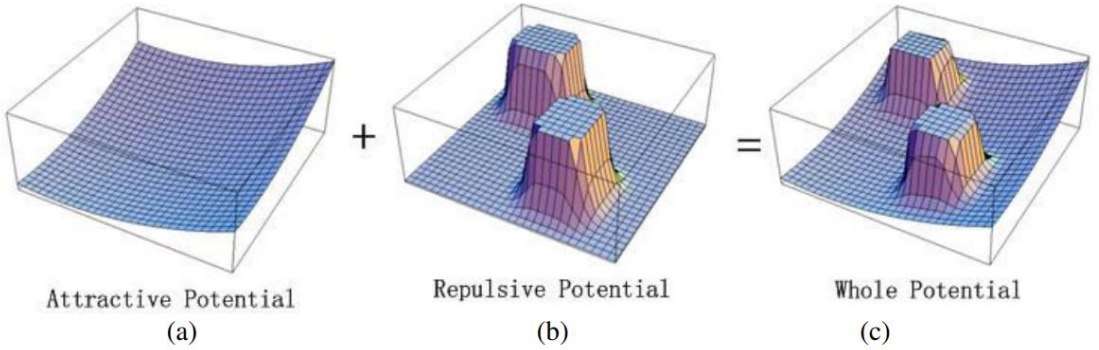
- For each node n in the graph:
  - $n.g = \infty$ (the distance fro
  - $n.f = \infty$ (g value
- Create an empty list.
- $start.g = 0$, start
- While list no emp
  - Let current
  - Remove cu
  - If (current
  - For each no
    - If ($n.g$ $rent.$
      - $n.$ $rrent.$
      - $n.f$
      - $n.parent$
      - Add n to list if
- **Trace a path:**
  - Start from destination.
  - Move from son to his parent.



Complexity = Dijkstra's complexity (worst case)

Potential Field Algorithm

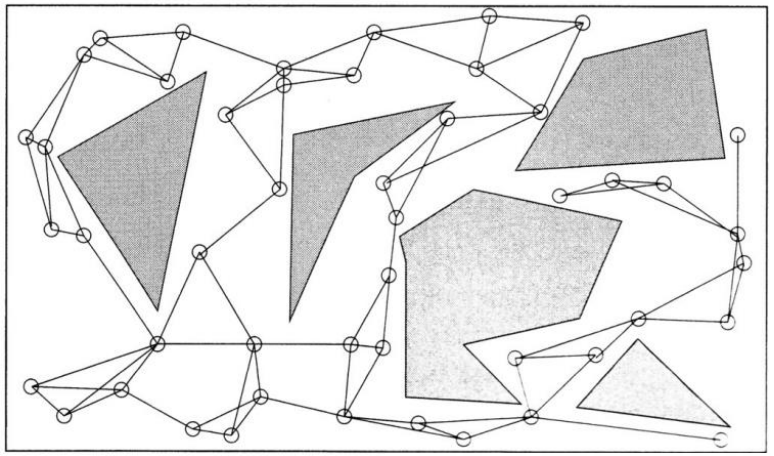
- Create APT model
- Potential field calculation (gradient descent)
- Start movement at starting point
  - If (goal reached)
    - Stop navigation
  - Else:
    - If obstacle is detected:
      - APF model
      - Potential field calculation
      - Turning angel to find safe position
      - if: (goal eached):
        - Stop navigation
      - Else if (obstacle):
        - Repeat APF
    - Else:
      - Keep navigate till goal is reached



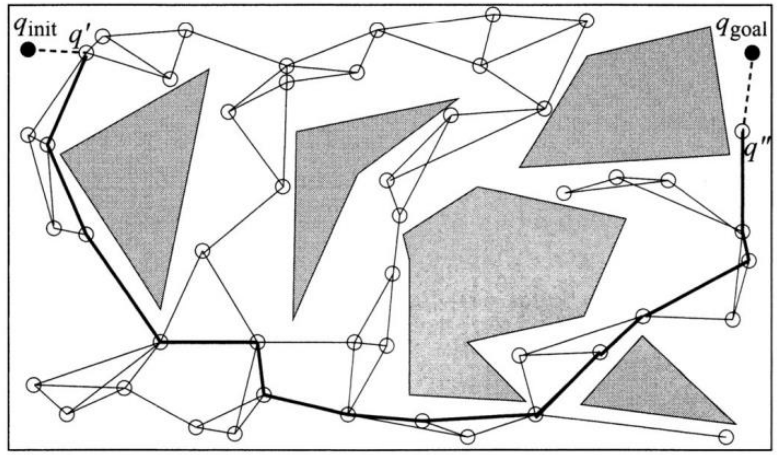
Complexity = $O(|V| * dimension)$

Random algorithms

Probabilistic Road-Map (PRM) Algorithm



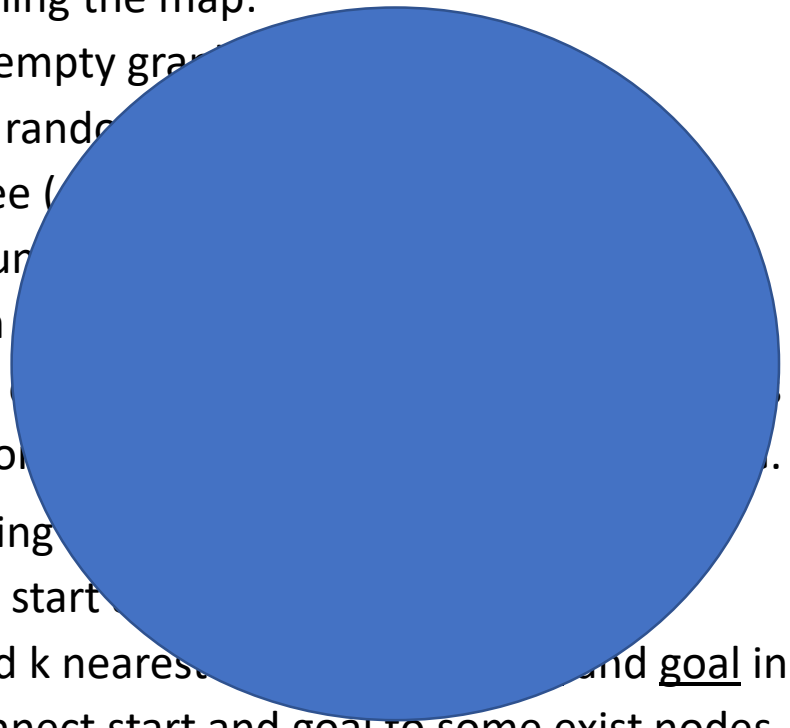
- **Step 1** – learning the map:
  - Initially empty graph – G (V,E).
  - Chose q randomly.
  - If q is free (collision detection) add to G.
  - Repeat until N nodes chosen.
  - For each q' select k closest neighbors:
    - For each neighbor – connect q to neighbors q'.
    - If connect successful – add edge (q,q') to G.
- **Step 2** – Finding a Path
  - Connect start and goal to exist G:
    - Find k nearest neighbors of start and goal in roadmap.
    - Connect start and goal to some exist nodes.
  - Use Dijkstra algorithm.



Complexity = Dijkstra's complexity (worst case)

RRT Algorithm

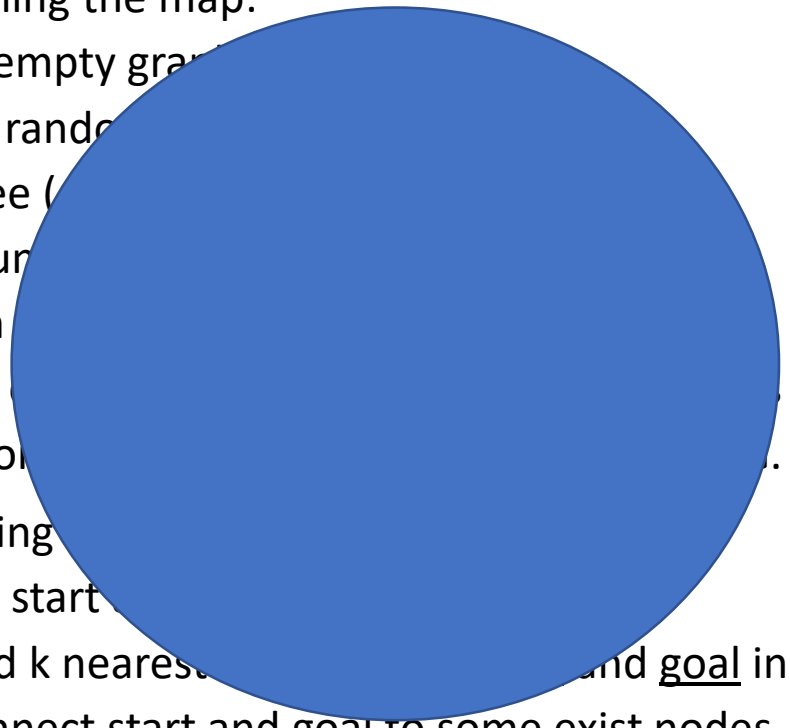
- **Step 1** – learning the map:
  - Initially empty gra
  - Chose q rando
  - If q is free (
  - Repeat un
  - For each
    - For q'.
    - If co .
- **Step 2** – Finding
  - Connect start
    - Find k neares nd goal in roadmap.
    - Connect start and goal to some exist nodes.
  - Use Dijkstra algorithm.



Complexity = Dijkstra's complexity (worst case)

RRT Algorithm

- **Step 1** – learning the map:
  - Initially empty gra
  - Chose q rando
  - If q is free (
  - Repeat un
  - For each
    - For q'.
    - If co .
- **Step 2** – Finding
  - Connect start
    - Find k neares nd goal in roadmap.
    - Connect start and goal to some exist nodes.
  - Use Dijkstra algorithm.



Complexity = Dijkstra's complexity (worst case)