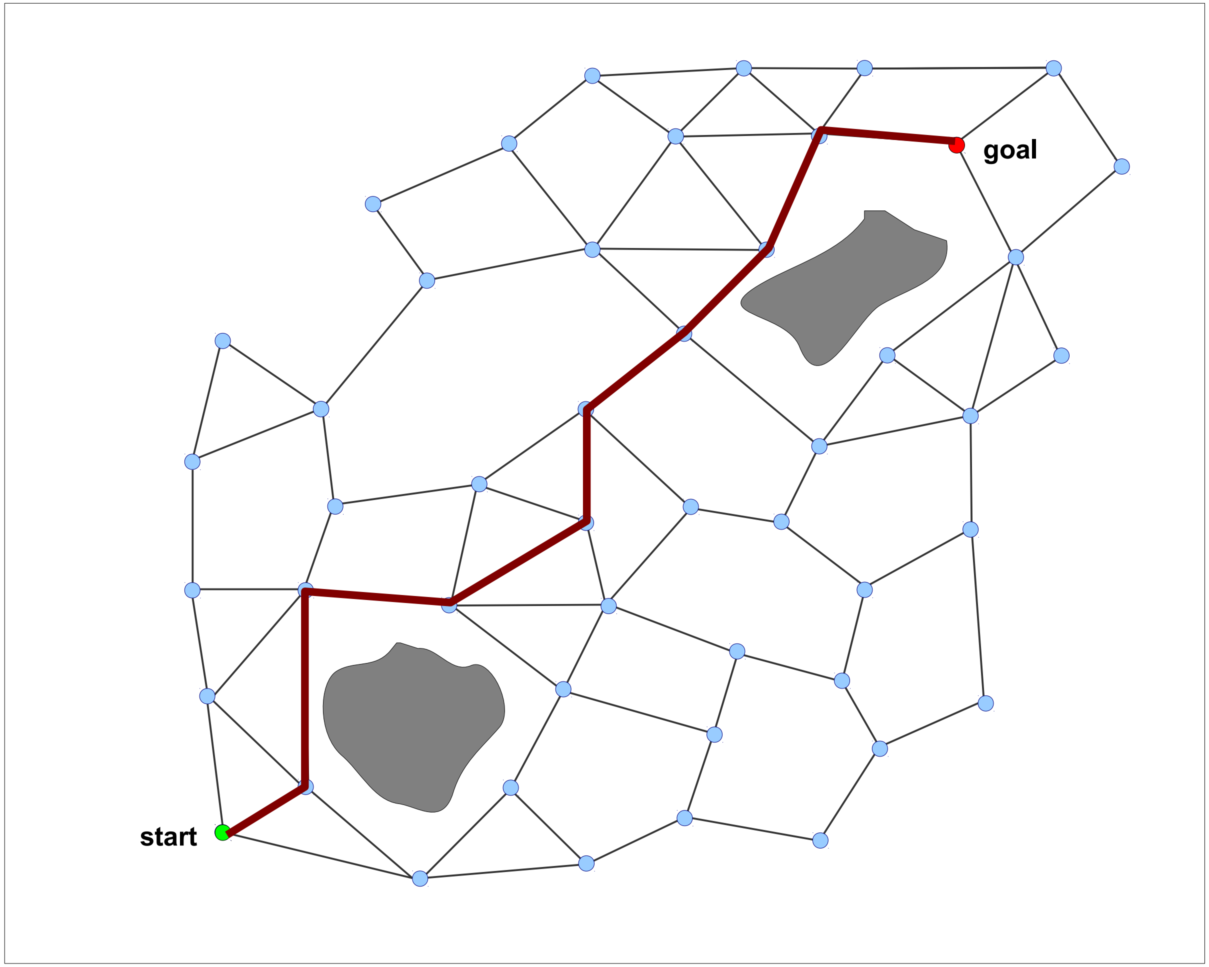
Introduction to Search-based Planning

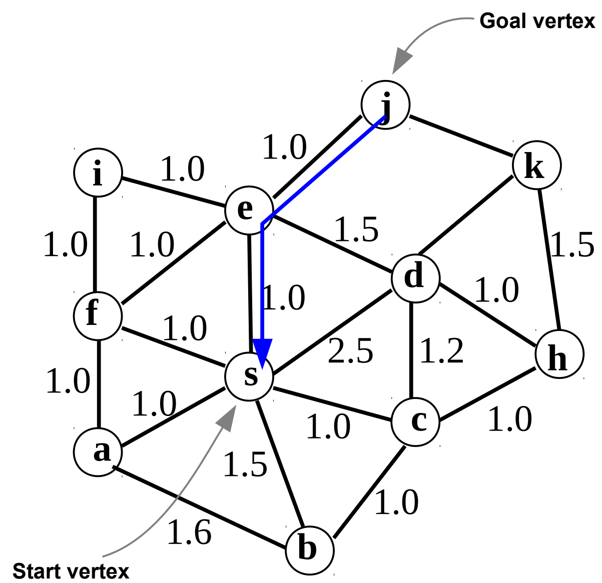
Subhrajit Bhattacharya
Lehigh University

Search-based Planning
(Trajectory Planning Using Graph Search)



Optimal Path Planning in a Graph

Dijkstra's search algorithm:



Open list

g-scores ($g: V \rightarrow \mathbb{R}_+$)

~~$g(\mathbf{s}) = 0.0$~~

~~$g(\mathbf{a}) = 1.0$~~

~~$g(\mathbf{b}) = 1.5$~~

~~$g(\mathbf{c}) = 1.0$~~

$g(\mathbf{d}) = 2.2$

~~$g(\mathbf{e}) = 1.0$~~

~~$g(\mathbf{f}) = 1.0$~~

~~$g(\mathbf{h}) = 2.0$~~

~~$g(\mathbf{i}) = 2.0$~~

~~$g(\mathbf{j}) = 2.0$~~

$g(\mathbf{k}) = 3.5$

Additional observations/details:

- At any instant there are 3 disjoint sets of vertices: ***Open list***, ***closed list***, and *the rest*.
- ***Heap data structure*** – efficient maintenance of an ordered list with fast insertion operations
  (for maintaining g-scores for the *open list*).
- Worst complexity: O(|E | + |V | log |V |)

Pseudocode (Dijkstra's)

$g =$ **Dijkstras** (G, p)

Inputs: a. Graph G

b. Start node $p \in \mathcal{V}(G)$

Outputs: a. The shortest distance map $g : \mathcal{V}(G) \to \mathbb{R}^+$

Initiate g: Set $g(v) := \infty$, for all $v \in \mathcal{V}(G)$ // Minimum distance
Set $g(p) = 0$
Set $Q := \{p\}$ // Open list
while ($Q \neq \emptyset$ AND *stopping criterion not satisfied*)
    $q := \text{argmin}_{q' \in Q} \; g(q')$ // Vertex to expand. Q is maintained by a heap data-structure.
    if $(g(q) == \infty)$
        break
    $Q = Q - q$ // Remove q from Q
    for each $(\{w \in \mathcal{N}_G(q) \mid w \in Q \;\text{ OR }\; g(w) == \infty\})$ // For each unexpanded neighbor of q
        Set $g' := g(q) + \mathcal{C}_G([q, w])$
        if $(g' < g(w))$
            Set $g(w) = g'$
            Set $Q = Q \cup \{w\}$ // Insert in open list if not already there.
return g

Pseudocode (Path Reconstruction)

$P =$ **Reconstruct_Path** (G, g, r)

Inputs: a. Graph G

b. The shortest distance map $g : \mathcal{V}(G) \to \mathbb{R}^+$

c. Vertex to which to find the shortest path, $r \in \mathcal{V}(G)$

Outputs: a. A path (ordered set of vertices) in the graph, $P = [\rho_1,\ \rho_2,\ \rho_3,\ \cdots,\ \rho_n = r]$

Initiate $P = [\ ]$
if $(g(r) == \infty)$ // r unreachable from the start node
    return P
Set $v := r$
while $(g(v) \neq 0)$
    $P = v \oplus P$ // Insert v at the beginning of P.
    $v = \operatorname{argmin}_{w \in \mathcal{P}_{G,g}(v)} g(w)$ // back-trace predecessor that led to v.
$P = v \oplus P$ // Insert the final vertex (the start node) at the beginning of P.
return P

Where, $\mathcal{P}_{G,g}(u) = \{w' \in \mathcal{V}(G) \mid [w', u] \in \mathcal{E}(G),\ g(v) = g(w') + \mathcal{C}_G([w', u])\}$

is the set of potential *predecessors* of u.

Pseudocode (A*)

$P =$ **A*** (G, p, r, h_r)

Inputs:
- a. Graph G
- b. Start node $p \in \mathcal{V}(G)$
- c. Goal node $r \in \mathcal{V}(G)$
- d. An admissible heuristic function, $h_r : \mathcal{V}(G) \rightarrow \mathbb{R}_+$

Outputs:
- a. A path connecting start vertex to goal vertex, $P = [p = \rho_1,\ \rho_2,\ \rho_3,\ \cdots,\ \rho_n = r]$

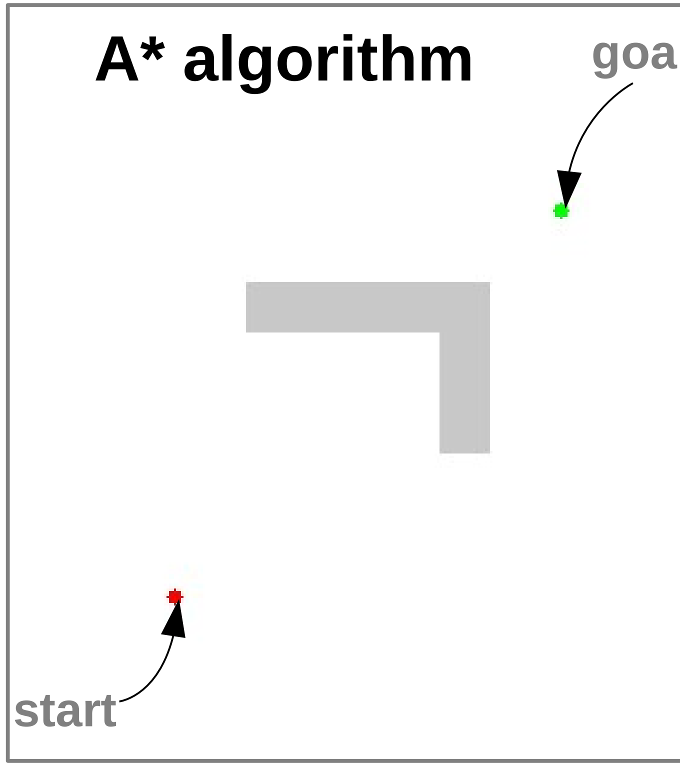
Initiate g, f: Set $g(v) := \infty$, $f(v) := \infty$, for all $v \in \mathcal{V}(G)$
Set $g(p) = 0$ and $f(p) = h(p)$
Set $Q := \{p\}$ // Open list
while $(Q \neq \emptyset$ AND *stopping criterion not satisfied*$)$
 $q := \text{argmin}_{q' \in Q}\ f(q')$ // Vertex to expand. Q is maintained by a heap data-structure.
 if $(q == r)$ // Goal vertex reached.
  return **Reconstruct_Path** (G, g, r)
 if $(g(q) == \infty)$
  break // No path found.
 $Q = Q - q$ // Remove q from Q
 for each $(\{w \in \mathcal{N}_G(q) \mid w \in Q$ OR $g(w) == \infty\})$ // For each unexpanded neighbor of q
  Set $g' := g(q) + \mathcal{C}_G([q, w])$
  if $(g' < g(w))$
   Set $g(w) = g'$
   Set $f(w) = g' + h_r(w)$
   Set $Q = Q \cup \{w\}$ // Insert in open list if not already there.
return []

Admissible heuristics function: $h_r(v)$ should be less than or equal to cost of shortest path from v to r.

Properties of A*

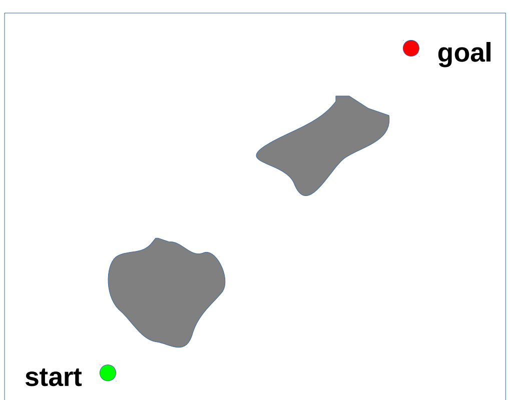
- If h_r is admissible, then A* is guaranteed to return the optimal path connecting the start and goal, r.
- Dijkstra's search is equivalent to A* search with zero heuristics.
- ***Weighted A*:*** If the heuristic function used, $\bar{h}_r$, is such that
  $\bar{h}_r \leq \varepsilon h_r$ (where, h_r is admissible, ε>1)
  then, the computed path is at most ε-suboptimal (the cost of the returned path is at most ε times the cost of the optimal path).

Optimal Path Planning in a Graph



Speed up search (expand less vertices) using an ***admissible heuristic function*** ($h: V \times V \rightarrow \mathbb{R}_+$) that underestimates the true least cost between vertices.
Still guarantees optimality.

Features of Dijkstra's and A*:



- The complete graph need not be available to start with.
- We only need to be able to query the list of neighboring vertices of vertex that we are ***expanding***, and cost of the edges connecting to them.
  (*Ex:* useful when, for example, the obstacles are given as semi-algebraic sets).
- The graph itself may be infinite/unbounded.

C++ Library for Graph Search

- DOSL (Discrete Optimal Search Library) : Available at https://github.com/subh83/DOSL