EN.530.663: Robot Motion Planning
Graph Search Algorithms

Jin Seob Kim, Ph.D.
Senior Lecturer, ME Dept., LCSR, JHU

1 Dijkstra's Algorithm

1.1 Procedure

1. Initialize the distance to the initial node as 0 and initialize temporary distances for the remaining nodes to ∞. Additionally initialize the closest neighbor as NULL.

2. Set the initial node n_{init} as current and create a set of all unvisited nodes $U = \{n_1, n_2, \cdots, n_p\}$, where p is the total number of nodes.

3. For the current node, calculate the distance to each unvisited neighbor (if one exists) by summing the edge weights from the initial node to the unvisited neighbor. If the calculated distance to the unvisited neighbor is smaller than the current assigned distance, update the distance of the unvisited neighbor and change the closest neighbor to the current node.

4. When all unvisited neighbors have been visited, mark the current node as visited and, if applicable, remove it from the unvisited set.

5. If the goal node n_{goal} has been marked visited, stop and return the shortest path.

6. if n_{goal} has not been visited, mark the node with smallest tentative distance as current and repeat Steps 3 through 6.

1.2 Pseudocode

Note: loosely borrowed from: http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html

Use the weighted adjacency matrix as a representative of a graph in the function. Same as in A* algorithm.

function Dijkstra(graph, n_{init}, n_{goal})
  for each node n in *graph* **do**:
    dist[n] $\doteq \infty$
    prev[n] $\doteq$ NULL
  end for
  dist[n_{init}] $\doteq 0$
  $U \doteq$ set of all nodes
  while $n_{goal} \in U$ **do**:
    $C \doteq$ node in U with smallest distance
    remove C from U
    for each neighbor v of C **do**:
      alt $\doteq$ dist[C] + distance between C and v
      if alt $<$ dist[v] **then**:
        dist[v] $\doteq$ alt
        prev[v] $\doteq C$
      end if
    end for
  end while
  return dist[n_{goal}], prev[n_{goal}]

Note: You have to write an additional code for constructing a shortest path mainly based on "prev" (and "dist")

2 A* Algorithm

2.1 Procedure

1. Initialize the distance to n_{init} as 0 and the heuristic as its actual calculated value. Initialize temporary distances and heuristics for the remaining nodes to ∞. Calculate the cost for each node by adding the distance and heuristic. Additionally, initialize the closest neighbor for each node as NULL.

2. Create an open set, O, containing n_{init} and an empty closed set, C (contains visited nodes).

3. Select the node n_{best} with the smallest cost in O, and set it to active.

4. If the active node is n_{goal}, return the cost and the shortest path.

5. For each neighbor of the active node not in C, calculate the distance and the heuristic cost. If the new distance is smaller than the temporary distance, update the cost of the node and set the closest neighbor to the active node. If the neighbor is not in O, add it.

6. Once all neighbors have been visited, add the active node to C.

7. If n_{goal} is not the active node, repeat Steps 3 through 6.

2.2 Pseudocode

function Astar(G, n_{init}, n_{goal})
  $O = \{n_{init}\}$ `% open set, a priority queue`
  $C = \emptyset$ `% closed set`
  for each node n in graph G **do**:
    $prev[n] = NULL$ (set backpointer set as null set)
    $f[n] = \infty$
    $g[n] = \infty$
  end for
  $g[n_{init}] = 0$
  $f[n_{init}] = heuristic\_cost\_estimate(n_{init}, n_{goal})$
  while O is not empty **do**:
    Pick n_{best} from O such that $f(n_{\text{best}}) \leq f(n)$, $\forall n \in O$.
    Remove n_{best} from O
    Add n_{best} to C.
    if $n_{best} = n_{goal}$ **then**:
      exit
    end if
    for each neighbor, x, of n_{best} **do**:
      if $x \in C$ **then**:
        continue
      end if
      $g\_temp \doteq g[n_{best}] + dist\_between(n_{best}, x)$
      if $x \notin O$ **then**:
        $openset.add(x)$: add x to open set O
      else if $g\_temp \geq g[x]$ **then**:
        continue
      end if
      $prev[x] \doteq n_{best}$
      $g[x] \doteq g\_temp$
      $f[x] \doteq g[x] + heuristic\_cost\_estimate(x, n_{goal})$
    end for
  end while

Then using prev and others, reconstruct the shortest path.