Introduction to Cognitive Robotics

Module 3: Mobile Robots

Lecture 10: The A* Algorithm

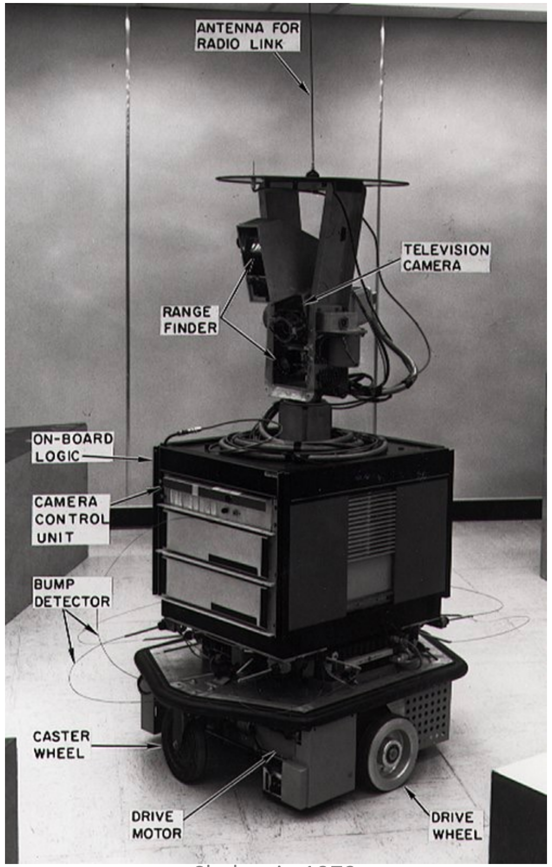
David Vernon
Carnegie Mellon University Africa

www.vernon.eu

Recall RPP01-02: History of Robotics

Some research results

- The A* search algorithm
- The Hough transform
- The visibility graph method
- Major impact on the development of robotics & AI
  (and computer science, generally)



Shakey in 1972
https://en.wikipedia.org/wiki/Shakey_the_robot

The A* Algorithm

- A* is a modification of Dijkstra's Algorithm that is optimized for a single destination
  - Dijkstra's Algorithm can find paths to all locations
  - A* finds paths to one location, or the closest of several locations
- A* prioritizes paths that seem to be leading closer to a goal

The A* Algorithm

- A* combines the information that Dijkstra's Algorithm uses
  - favoring vertices that are close to the starting point (i.e. shortest path from the starting point)
- and information that Greedy Best-First-Search uses
  - favoring vertices that are close to the goal based on a heuristic

The A* Algorithm

- Let $g(n)$ represent the exact cost of the path from the start vertex to any vertex n (the is the shortest path length as computed using Dijkstra's algorithm)

This can be as simple as the Euclidean distance from vertex n to the goal

- Let $h(n)$ represent the heuristic estimated cost from vertex n to the goal

- A* balances the two as it moves from the start vertex point to the goal vertex

- Each time through the main loop, it examines the vertex n that has the lowest

$$f(n) = g(n) + h(n)$$

This represents our current best guess as to how short a path from start to finish can be if it goes through n

The A* Algorithm

AStar(G, s, t)

Difference from Dijkstra in blue

This is $g(n)$ in the previous slides

A heuristic estimate of the distance from the candidate vertex x to the goal

For a discussion of different options when choosing the heuristic, see http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

path= {s}

for i = 1 to n, dist[i] = ∞, f[i] = ∞

for each edge (s, v), dist[v] = w(s, v), f[v] = w(s, v) + h[v])

last = s

while (last != t)

  select v_{next}, the unknown vertex minimizing f[v] (= dist[v]+ h[v])

  for each edge (v_{next}, x)

    if dist[x] > dist[v_{next}] + w(v_{next}, x)

      dist[x] = dist[v_{next}] + w(v_{next}, x)

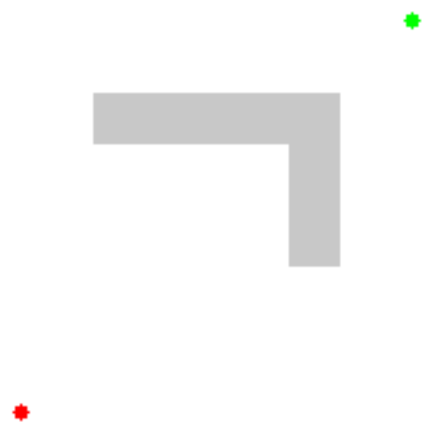
      parent[x] = v_{next}

      f[x] = dist[x] + h(x)

  last = v_{next}

  path = path ∪ {v_{next}}

The A* Algorithm



"Illustration of A* search for finding path from a start node to a goal node. The empty circles represent the nodes in the *open set*, i.e., those that remain to be explored, and the filled ones are in the closed set. Color on each closed node indicates the distance from the goal: the greener, the closer.

One can first see the A* moving in a straight line in the direction of the goal, then when hitting the obstacle, it explores alternative routes through the nodes from the open set."

https://en.wikipedia.org/wiki/A*_search_algorithm

The A* Algorithm

If $h(n) = 0$

- Only $g(n)$ plays a role
- A* turns into Dijkstra's Algorithm and is guaranteed to find a shortest path

If $h(n)$ is always lower than (or equal to) the cost of moving from n to the goal

- A* is guaranteed to find a shortest path
- The lower $h(n)$ is, the more nodes A* expands, making it slower

If $h(n)$ is exactly equal to the cost of moving from n to the goal

- A* will only follow the best path and never expand anything else, making it very fast

http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

The A* Algorithm

If $h(n)$ is sometimes greater than the cost of moving from n to the goal

- A* is not guaranteed to find a shortest path, but it can run faster

If $h(n)$ is very high relative to $g(n)$

- then only $h(n)$ plays a role, and A* turns into Greedy Best-First-Search

http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

The A* Algorithm

We can decide what we want to get out of A*

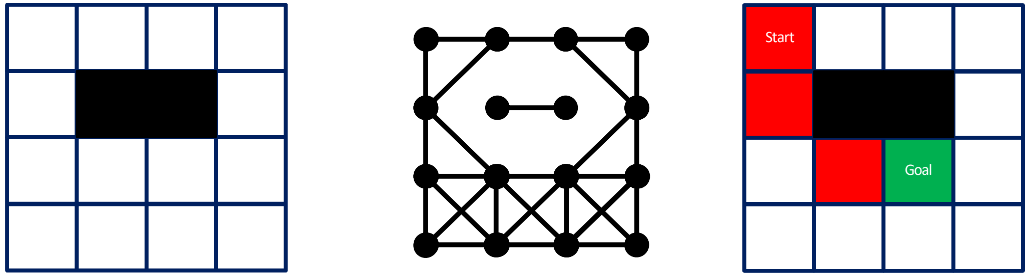
- With 100% accurate estimates, we'll get shortest paths really quickly
- If we're too low, then we'll continue to get shortest paths, but it'll slow down
- If we're too high, then we give up shortest paths, but A* will run faster

http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

Summary

- If you want a simple optimal path planner
  - use Breadth-First Search on an unweighted graph

- If you want an optimal path planner with diagonal paths on a grid
  - use a weighted graph and Dijkstra's shortest path algorithm (aka Uniform Cost Search)



Summary

- If you want an efficient path planner
  - Use more sophisticated form with heuristic search

    Iterative Deepening A*

    e.g. A*, IDA*, D*

    Dynamic A*

- If you want to model uncertainty in the robot environment
  - Use other AI techniques, e.g., Markov decision processes, POMDP

Reading

For an intuitive explanation of Breadth First Search, Dijkstra's Algorithm, and A*, see

https://www.redblobgames.com/pathfinding/a-star/introduction.html