Graphs
- Abstraction representation of a collection of items
  - A collection of vertices & edges

- Terminology
  - Adjacency
  - Paths and Cycles
  - Degree of a vertex
  - Fully connected graphs

Types of Graphs
- Directed vs. Undirected
- Weighted vs. Unweighted

Types of Graphs
- Cyclic vs. Acyclic
- Trees

Graph Representation: Example
Objective: We want to represent all possible ways to get from configuration A to B using a graph
HOW?
Graph Representation: Example

Search in Path Planning

- Find a path between two locations in an unknown, partially known, or known environment

Search Performance
- Completeness
- Optimality → Operating Costs
- Space Complexity
- Time Complexity

Search

- Uninformed Search
  - Use no information obtained from the environment
  - Blind search: BFS (Wavefront), DFS

- Informed Search
  - Use evaluation function
  - More efficient
  - Heuristic Searchers: A*, D*, etc.

Uninformed Search

- Graph search from A to N

Uninformed Search

- Stack
  - Push and Pop here
- Queue
  - Push here
  - Pop here
- BFS
Informed Search: A*

- Notation
  - $n \rightarrow$ node/state
  - $c(n_1, n_2) \rightarrow$ the length of an edge connecting $n_1$ and $n_2$
  - $b(n_1) = n_2 \rightarrow$ backpointer of a node $n_1$ to a node $n_2$

Informed Search: A*

- Evaluation/Cost Function $f(n) = g(n) + h(n)$
- Operating Cost Function $g(n)$
  - Actual operating cost of having been traversed
- Heuristic Function $h(n)$
  - Information used to find promising nodes to traverse
  - Admissible $\rightarrow$ never overestimate the actual path cost

Cost on Grid

A*: Algorithm

- The search requires 2 lists to store information about nodes
  1. Open list (O) stores nodes for expansion
  2. Closed list (C) stores nodes which we have explored

Start

- O is empty? Y End
- Pick $n_{best}$ from O s.t. $f(n_{best}) \leq f(n)$
- Remove $n_{best}$ from O & add it to C
- $n_{best} = \text{goal}$? Y End
- Expand all nodes $x$ who are nbrs of $n_{best}$ & not in C
- $x$ is not in O? Y Add x to O
- Update $b(x)$ on $n_{best}$ if $g(n_{best}) + c(n_{best}) < g(x)$
- $x$ is not in C? Y

Dijkstra's Search: $f(n) = g(n)$

1. $O = \{S\}$
2. $O = \{1,2,4,5\}; C = \{S\}$ (all point back to S)
3. $O = \{1,4,5\}; C = \{S,2\}$ (there are no adjacent nodes not in C)
4. $O = \{1,5,3\}; C = \{S,2,4\}$ (1,2,4, point to S; 5 points to 4)
5. $O = \{5,3\}; C = \{S,2,4,1\}$
6. $O = \{3,S\}; C = \{S, 2, 4, 1\}$ (goal points to 5 which points to 4 which points to S)
Two Examples of Running A*

Example (1/5)

Example (2/5)

Example 3/5
Example 4/5

There might be a shorter path, but assuming non-negative arc costs, nodes with a lower priority than the goal cannot yield a better path.

In this example, nodes with a priority greater than or equal to 5 can be pruned.

Why don't we expand nodes with an equivalent priority? (why not expand nodes D and F?)

Example 5/5

We can continue to throw away nodes with priority levels lower than the lowest goal found.

As we can see from this example, there was a shorter path through node K. To find the path, simply follow the back pointers.

Therefore the path would be:

Start = C — K — Goal

A*: Example (1/6)

Heuristics

A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6, H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

A*: Example (2/6)

Heuristics

A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6, H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

E(s) = E(π(n)) = g(n) + h(n) = 3 + 8 = 11

Closed List

Open List - Priority Queue

A(0)

E(11)

H(14)

A(x) = Alg(n)
A*: Example (3/6)

**Heuristics**
A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6
H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

Since A → B is smaller than A → E → B, the f-cost value of B in an open list needs not be updated.

A*: Example (4/6)

**Heuristics**
A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6
H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

A*: Example (5/6)

**Heuristics**
A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6
H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

A*: Example (6/6)

**Heuristics**
A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6
H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

Since the path to N from M is greater than that from J, the optimal path to N is the one traversed from J.
**A*: Example Result**

Generate the path from the goal node back to the start node through the back-pointer attribute.

**A*: Performance Analysis**

- Complete provided the finite boundary condition and that every path cost is greater than some positive constant $\delta$
- Optimal in terms of the path cost
- Memory inefficient
- Exponential growth of search space with respect to the length of solution