Graph Search
Howie Choset
16-311
Outline

• Overview of Search Techniques
• A* Search
Graphs

Collection of Edges and Nodes (Vertices)

A tree
Grids
Stacks and Queues

Stack: First in, Last out (FILO)

Queue: First in, First out (FIFO)
Depth First Search

algorithm dft(x)
  visit(x)
  FOR each y such that (x, y) is an edge
    IF y was not visited yet
      THEN dft(y)

Worst case performance

$O(|V| + |E|)$ for explicit graphs traversed without repetition,

Worst case space complexity

$O(|V|)$ if entire graph is traversed without repetition,
$O(\text{longest path length searched})$ for implicit graphs without elimination of duplicate nodes.

Copied from wikipedia
Breadth First Search

visit(start node)
queue <- start node
WHILE queue is not empty DO
  x <- queue
  FOR each y such that (x,y) is an edge and y has not been visited DO
    visit(y)
    queue <- y
  END
END

http://www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch14.html
Depth First and Breadth First
Wavefront Planner: A BFS
Search

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

• Informed Search
  – Use evaluation function
  – More efficient
  – Heuristic Search: A*, D*, etc.
Uninformed Search

Graph Search from A to N

BFS
Informed Search: A*

**Notation**

- $n \rightarrow$ node/state
- $c(n_1,n_2) \rightarrow$ the length of an edge connecting between $n_1$ and $n_2$
- $b(n_1) = n_2 \rightarrow$ backpointer of a node $n_1$ to a node $n_2$. 
Informed Search: A*

- Evaluation function, $f(n) = g(n) + h(n)$
- Operating cost function, $g(n)$
  - Actual operating cost having been already traversed
- Heuristic function, $h(n)$
  - Information used to find the promising node to traverse
    Admissible: never overestimate the actual path cost

Cost on a grid
A*: Algorithm

The search requires 2 lists to store information about nodes

1) **Open list (O)** stores nodes for expansions

2) **Closed list (C)** stores nodes which we have explored

```
Start

Pick n_best from O such that f(n_best) ≤ f(n)

Remove n_best from O and add it to C

n_best = goal?

End

Expand all nodes x that are neighbors of n_best and not in C

x is not in O?

Add x to O

update b(x)=n_best if (g(n_best)+c(n_best,x)<g(x))

O is empty?

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

1. $O = \{S\}$

2. $O = \{1, 2, 4, 5\}; C = \{S\}$ (1,2,4,5 all back point 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. $O = \{3, G\}; C = \{S, 2, 4, 1\}$ (goal points to 5 which points to 4 which points to S)

7. $O = \{3\}; C = \{S, 2, 4, 1\}$ (Path found because Goal was popped)
Two Examples Running A*
Example (1/5)

Legend

Priority = \( g(x) + h(x) \)

Note:

\[ g(x) = \text{sum of all previous arc costs, } c(x), \text{ from start to } x \]

Example: \( c(H) = 2 \)
Example (2/5)

First expand the start node

If goal not found, expand the first node in the priority queue (in this case, B)

Insert the newly expanded nodes into the priority queue and continue until the goal is found, or the priority queue is empty (in which case no path exists)

Note: for each expanded node, you also need a pointer to its respective parent. For example, nodes A, B and C point to Start
Example (3/5)

We’ve found a path to the goal:
Start => A => E => Goal
(from the pointers)

Are we done?
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 I?)
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
Monotonic

• never overestimates the cost of getting from a node to its neighbor.

• for all paths $x, y$ where $y$ is a successor of $x$, i.e.,

$$h(x) \leq g(y) - g(x) + h(y)$$

• $h(A) = 3 \quad g(A) = 1 \quad h(E) = 1 \quad g(E) = 2$

$$h(A) = 3 \neq g(E) - g(A) + h(E) = 2 - 1 + 1 = 2$$
A*: Example (1/6)

Heuristics

\[
\begin{align*}
A &= 14 & H &= 8 \\
B &= 10 & I &= 5 \\
C &= 8 & J &= 2 \\
D &= 6 & K &= 2 \\
E &= 8 & L &= 6 \\
F &= 7 & M &= 0
\end{align*}
\]

Legend

- Operating cost

The diagram illustrates a graph with nodes labeled A through N, and costs associated with the edges. The heuristics for each node are also provided to demonstrate the use of A* algorithm in pathfinding.
A*: Example (2/6)

Heuristics

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

Closed List

Open List - Priority Queue

\[ A(0) \]
\[ E(11) \]
\[ B(14) \]
\[ H(14) \]

Expand

\[ A(x) = A(g(n)) \]

\[ E(y) = E(f(n)) \]

where \( f(n) = g(n) + h(n) \)

\[ = 3 + 8 = 11 \]
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.
1. Put S on priority Q and expand it
2. Expand A because its priority value is 7
3. The goal is reached with priority value 8
4. This is less than B’s priority value which is 13