Problem Solving and Search

Andrea Danyluk
February 10, 2017

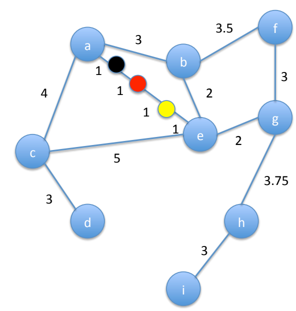
Announcements

- Programming Assignment 1: Search
  - Posted online

Today's Lecture

- Informed (Heuristic) search
  - Greedy best-first
  - A*
- Will talk a bit more about heuristics on Monday

Uniform Cost Search



Similar to another algorithm you know?

Fringe is a Priority Queue

Priority = cost so far

Evaluating Uniform Cost Search

- Complete?
  - Yes (if b is finite and step cost ≥ ε for positive ε)
- Optimal?
  - Yes
- Time Complexity?
  - $O(b^{1+C^*/\varepsilon})$ 🡐 Can't check for goal until coming out of PQ!
- Space Complexity?
  - $O(b^{1+C^*/\varepsilon})$

Generalized Search

```
Function SEARCH(problem, frontier)
        returns a solution, or failure
   expanded ← an empty set
   frontier ← Insert(Make-Node(Initial-State[problem]),
                     frontier)
   loop do
      if frontier is empty then return failure
      node ← Remove(frontier)
      if Goal-Test(problem, State[node]) then return node
      if State[node] is not in expanded then
         add State[node] to expanded
         frontier ← InsertAll(Expand(node, problem),
                              frontier)
   end
```

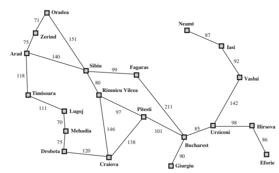
But be careful about slight (and yet significant) differences!

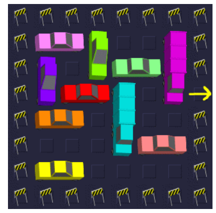
So we're done, right?

Our search spaces are big.

Informed (Heuristic) Search Strategies

- Use problem-specific knowledge to find solutions more efficiently





Best-first Search

- Choose a node from the frontier based on its "desirability"
  - Frontier is a priority queue
- Requires a search heuristic
  - Any estimate of how close a state is to a goal
  - Examples:
    - Euclidean distance
    - Manhattan distance
    - For the "rush hour" parking problem?

Greedy Best-first Search

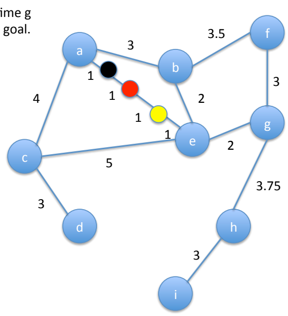
h(*n*) = estimate of cost from *n* to the closest goal

Expand the node with lowest h value

- i.e., the node that appears to be closest to the goal

Greedy Search with h_{SLD}

This time g is the goal.



a	6
b	2.5
c	6
d	3
e	1.5
f	3
g	0
h	3.5
i	6.5
black	4.5
red	3.5
yellow	2.5

Problems with Greedy Search?

Can be quite good with a high-quality heuristic, but

- Not optimal

- Time and space complexity: $O(b^m)$

Cost-Based Searches

- Uniform Cost Search
  - Expands leaf node on path with lowest cost so far
  - Good: Complete and Optimal
  - Bad: Explores "widely"; doesn't take into acct any info about the goal
- Greedy Search
  - Expands node that appears closest to a goal
  - Can take you (quickly) to the wrong goal

[Adapted from CS 188 UC Berkeley]

A* Search

- Uniform cost search
  - Orders nodes by *backward* cost g(*n*)
- Greedy search
  - Orders nodes by *forward* cost h(*n*)
- A* search
  - Orders nodes by the sum: f(*n*) = g(*n*) + h(*n*)

[Adapted from CS 188 UC Berkeley]

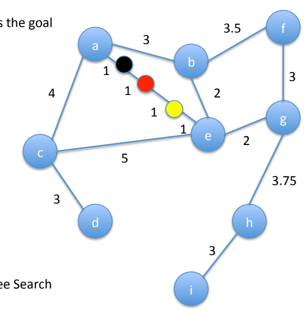
When should A* terminate?

- Should we perform the goal test when
  - Inserting a node into the priority queue?
  - Removing a node from the priority queue?

When removing from the priority queue

A* Search with h_{SLD}

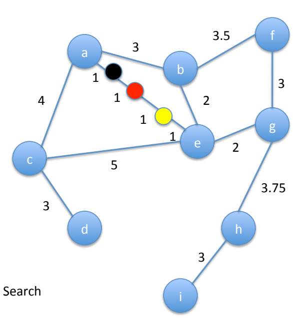
g is the goal



Tree Search

a	6
b	2.5
c	6
d	3
e	1.5
f	3
g	0
h	3.5
i	6.5
black	4.5
red	3.5
yellow	2.5

Is A* Search Optimal?



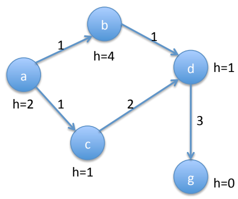
Tree Search

a	6
b	2.5
c	6
d	3
e	1.5
f	3
g	0
h	3.5
i	6.5
black	6
red	5
yellow	3.5

A* Conditions for Optimality

- Tree Search
  - Heuristic must be **admissible**
    - Never overestimates the cost to the goal

A* Graph Search Gone Wrong



What happens?

Heuristic admissible?

[Adapted from CS 188 UC Berkeley]

A* Conditions for Optimality

- Tree Search
  - Heuristic must be **admissible**
    - Never overestimates the cost to the goal
- Graph Search
  - Heuristic must be **consistent**
    - If n^1 is a successor of n generated by action a
      - $h(n) \le c(n, a, n^1) + h(n^1)$
      - if an action has cost *c, then taking that action can only cause a drop in heuristic of at most c*