Informed Search

CS 486/686
University of Waterloo
May 9

Outline

- Using knowledge
  - Heuristics
- Best-first search
  - Greedy best-first search
  - A* search
  - Other variations of A*
- Back to heuristics

Recall from last lecture

- Uninformed search methods expand nodes based on "distance" from start node
  - Never look ahead to the goal
  - E.g. in uniform cost search expand the cheapest path. We never consider the cost of getting to the goal
  - Advantage is that we have this information
- But, we often have some additional **knowledge** about the problem
  - E.g. in traveling around Romania we know the distances between cities so can measure the overhead of going in the wrong direction

Informed Search

- Our knowledge is often on the merit of nodes
  - Value of being at a node
- Different notions of merit
  - If we are concerned about the cost of the solution, we might want a notion of how expensive it is to get from a state to a goal
  - If we are concerned with minimizing computation, we might want a notion of how easy it is to get a state to a goal
  - We will focus on cost of solution

Informed search

- We need to develop a domain specific heuristic function, h(n)
- h(n) guesses the cost of reaching the goal from node n
  - The heuristic function must be domain specific
  - We often have some information about the problem that can be used in forming a heuristic function (i.e. heuristics are domain specific)

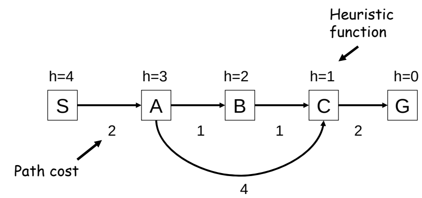
Informed search

- If h(n1)<h(n2) then we guess that it is cheaper to reach the goal from n1 than it is from n2
- We require
  - h(n)=0 when n is a goal node
  - h(n)>= 0 for all other nodes

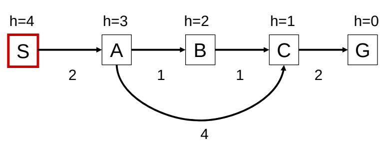
Greedy best-first search

- Use the heuristic function, h(n), to rank the nodes in the fringe
- Search strategy
  - Expand node with lowest h-value
- Greedily trying to find the least-cost solution

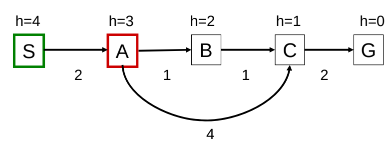
Greedy best-first search: Example



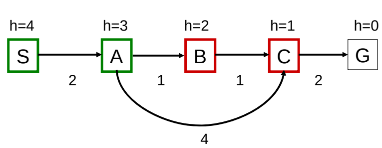
Example cont…



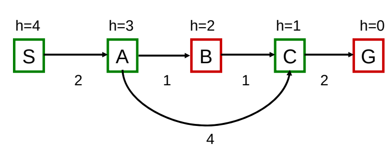
Example cont…



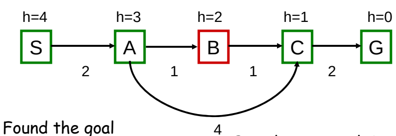
Example cont…



Example cont…



Example cont…



Found the goal

Path is S, A, C, G

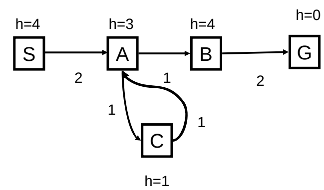
Cost of the path is 2+4+2=8

But cheaper path is S, A, B, C, G

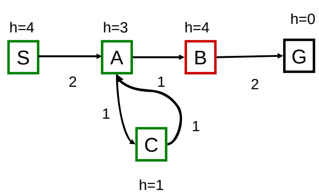
With cost **2+1+1+2=6**

Greedy best-first is not optimal

Another Example



Another Example



Greedy best-first can get stuck in loops

Not complete

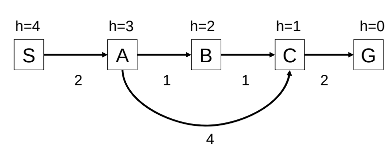
Properties of greedy search

- Not optimal!
- Not complete!
  - If we check for repeated states then we are ok
- Exponential space in worst case since need to keep all nodes in memory
- Exponential worst case time $O(b^m)$ where m is the maximum depth of the tree
  - If we choose a good heuristic then we can do much better

A* Search

- Greedy best-first search is too greedy
  - It does not take into account the cost of the path so far!
- Define
  - **f(n)=g(n)+h(n)**
  - g(n) is the cost of the path to node n
  - h(n) is the heuristic estimate of the cost of reaching the goal from node n
- A* search
  - Expand node in fringe (queue) with lowest f value

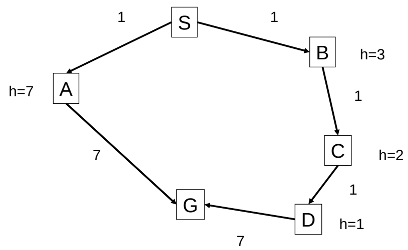
A* Example



1. Expand S
2. Expand A
3. Choose between B (f(B)=3+2=5) and C (f(C)=6+1=7)) expand B
4. Expand C
5. Expand G – recognize it is the goal

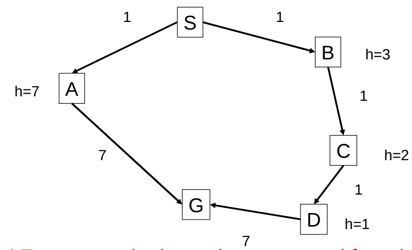
When should A* terminate?

- As soon as we find a goal state?



When should A* terminate?

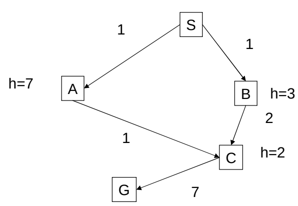
- As soon as we find a goal state?



A* Terminates only when goal state is popped from the queue

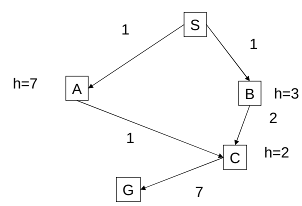
A* and revisiting states

What if we revisit a state that was already expanded?



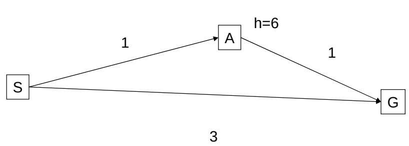
A* and revisiting states

What if we revisit a state that was already expanded?



If we allow states to be expanded again, we might get a better solution!

Is A* Optimal?



No. This example shows why not.

Admissible heuristics

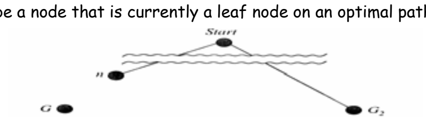
- Let $h^*(n)$ denote the true minimal cost to the goal from node n
- A heuristic, h, is admissible if
  - $h(n) \leq h^*(n)$ for all n
- Admissible heuristics never overestimate the cost to the goal
  - Optimistic

Optimality of A*

- If the heuristic is admissible then A* with tree-search is optimal

Let G be an optimal goal state, and $f(G) = f^* = g(G)$.
Let G_2 be a suboptimal goal state, i.e. $f(G_2) = g(G_2) > f^*$.
Assume for contradiction that A* has selected G_2 from the queue.
(This would terminate A* with a suboptimal solution)

Let n be a node that is currently a leaf node on an optimal path to G.



Because h is admissible, $f^* \geq f(n)$.

If n is not chosen for expansion over G_2, we must have $f(n) \geq f(G_2)$
So, $f^* \geq f(G_2)$. Because $h(G_2)=0$, we have $f^* \geq g(G_2)$, contradiction

Optimality of A*

- For searching graphs we require something stronger than admissibility
  - Consistency (monotonicity):
    - $h(n) \leq cost(n,n')+h(n')$
  - Almost any admissible heuristic function will also be consistent
- A* graph-search with a consistent heuristic is optimal

Properties of A*

- Complete if the heuristic is consistent
  - Along any path, f always increases) if a solution exists somewhere the f value will eventually get to its cost
- Exponential time complexity in worst case
  - A good heuristic will help a lot here
  - O(bm) if the heuristic is perfect
- Exponential space complexity

Memory-bounded heuristic search

- A* keeps most generated nodes in memory
  - On many problems A* will run out of memory
- Iterative deepening A* (IDA*)
  - Like IDS but change f-cost rather than depth at each iteration
- SMA* (Simplified Memory-Bounded A*)
  - Uses all available memory
  - Proceeds like A* but when it runs out of memory it drops the worst leaf node (one with highest f-value)
  - If all leaf nodes have the same f-value then it drops oldest and expands the newest
  - Optimal and complete if depth of shallowest goal node is less than memory size

Heuristic Functions

- A good heuristic function can make all the difference!
- How do we get heuristics?
  - One approach is to think of an easier problem and let h(n) be the cost of reaching the goal in the easier problem

8-puzzle

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- Relax the game
  1. Can move tile from position A to position B if A is next to B (ignore whether or not position is blank)
  2. Can move tile from position A to position B if B is blank (ignore adjacency)
  3. Can move tile from position A to position B

8-puzzle cont…

- 3 leads to misplaced tile heuristic
  - To solve this problem need to move each tile into its final position
  - Number of moves = number of misplaced tiles
  - Admissible
- 1 leads to manhattan distance heuristic
  - To solve the puzzle need to slide each tile into its final position
  - Admissible

8-puzzle cont…

- h1=misplaced tiles
- h2=manhattan distance
- Note h2 dominates h1
  - $h1(n) \leq h2(n)$ for all n
  - Which heuristic is best?

Designing heuristics

- Relaxing the problem (as just illustrated)
- Precomputing solution costs of subproblems and storing them in a pattern database
- Learning from experience with the problem class

Conclusion

- What you should now know
  - Thoroughly understand A* and IDA*
  - Be able to trace simple examples of A* and IDA* execution
  - Understand admissibility of heuristics
  - Proof of completeness, optimality
  - Criticize greedy best-first search

Next class

- Local search
  - Russell and Norvig Sections 4.3-4.5