CISS450 Lecture 4: Informed Search

Yihsiang Liow

October 13, 2020

Table of contents I

Informed search I

- **Informed search** = **heuristic search**
- Search uses more state information
- Recall: For uninformed search, state used only in:
  - expansion
  - goal test
- Let n be a search node. If actions have costs or weights, let

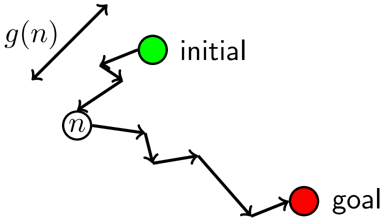
$$\begin{aligned} g(n) &= \text{path cost from initial node to } n \\ &= \text{sum of weights on actions from initial node to } n \end{aligned}$$

(Also written $\texttt{path\_cost}(n)$.) If weights are not indicated, assume cost of each edge/action is 1

Informed search II

- Note that path cost depends on the path and therefore depends on the search node n. Remember that in during a search, the path cost to a state can change, i.e., path cost depends on the path of a search node (of the state) going back to the initial node. A state can appear in many search nodes.
- (WARNING: The book AIMA's notation is wrong. It uses $g(s)$ where s is a state. The cost is path dependent. The correct notation is $g(n)$, where n is a search node, and not $g(s)$.)
- IMPORTANT. $g(n)$ does not include the cost to go from n to a node containing a goal state:

Informed search III



- NOTATION: I'll be talking about heuristic functions. These functions are functions on states. For convenience, if f is a function on states, if n is a search node that contains state s, I will also write $f(n)$ to mean $f(s)$:

$$f(n) = f(s)$$

Informed search IV

- It's really important to distinguish between $g(n)$ (i.e. `path_cost`(n)) which depends on search nodes and heuristic functions $f(s)$ which depend on states.
- Recall that uniform cost search (UCS) uses $g(n)$ to prioritize the fringe. I am now going to consider other ways to prioritize the fringe.

Best first search I

- **Best first search**:
  - create function f an **evaluation function**
  - $f(n) =$ measures approximately the cost from initial node to n and then from s (the state in n) to a goal state.
- Implement fringe as priority queue with search nodes ordered by f, i.e., this is just uniform cost search using this function f instead of path cost. Note that smallest f–value means higher priority.
- Note: This is a generalization of uniform cost search in the sense that best first search with $f(n) = g(n)$ is the uniform cost search.

Best first search II

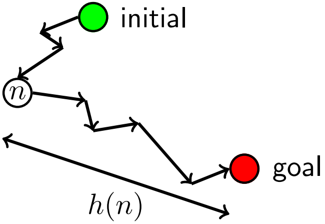
- Usually f involves a heuristic function:

$$f(n) = g(n) + \text{(some heuristic function on state of } n\text{)}$$

  See later.
- Not complete, not optimal in general – depends on quality of heuristic function.
- Worse time and space $= O(b^m)$ where m is max depth of search space.

Greedy Best first search I

- **Heuristic function** $h(n)$ = measures "distance" from node n to goal such that $h(\text{goal state or node with goal state}) = 0$
- Note: $h(n)$ is usually an approximation!



- **Greedy best-first search** = best-first search with $f = h$

Greedy Best first search II

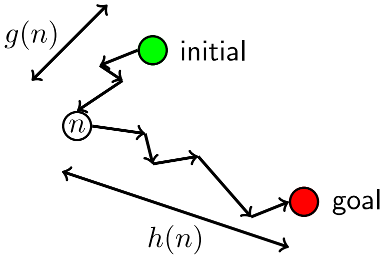
- Examples of h:
  - Path finding in real world: straight line distance from a point to the goal
  - $n^2 - 1$ puzzle: $h(n) =$ number of misplaced tiles for state corresponding to node
  - $n^2 - 1$ puzzle: $h(n) =$ sum of Manhattan distances of all tiles
- Properties of greedy best-first search:
  - Not optimal. Can you find an example?

A* search I

- **A* search**: use

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

where h is a heuristic function.



A* search II

- Note: $g(n)$ is the actual (real) cost to go from initial state to n (along some path). $h(n)$ is an approximate cost from n to a goal state.

Admissibility and consistency I

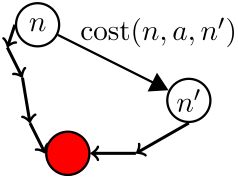
- A heuristic function h is **admissible** if it never overestimates the cost to reach a goal.
- Question: Is the straight line distance function an admissible heuristic function?

Admissibility and consistency II

- h is **consistent** if for any n with successor n' via action a:

$$h(n) \leq \text{cost}(n, a, n') + h(n')$$

where $\text{cost}(n, a, n')$ is the cost of action a from n to n'. The above inequality is called the **triangle inequality**.



triangle inequality

Admissibility and consistency III

- Facts:
  - A* using tree search algorithm is optimial if h is admissible
  - A* using graph search is optimal if h is consistent.

Admissibility and consistency IV

- Fact 1: A* using tree search is optimal if h is admissible

  Proof: Let G be an optimal goal node and G' be a suboptimal goal node in the fringe. Let C^* be cost of an optimal path.

$$\begin{aligned} G' \text{ suboptimal} &\implies g(G') > C^* \\ G \text{ optimal} &\implies f(G) = C^* \end{aligned}$$

Admissibility and consistency V

So

$$
\begin{aligned}
f(G) &= C^* \\
&< g(G') = g(G') + 0 \\
&= g(G') + h(G') \\
&= f(G')
\end{aligned}
$$

So G goes before G' in the fringe. $\square$

Admissibility and consistency VI

- Fact 3: If h is consistent, then the f-values along any path are nondecreasing.

  Proof: Let n' be the successor of n via action a. Then

  $$g(n') = g(n) + c(n, a, n')$$

  So

  $$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) = f(n) \end{aligned}$$

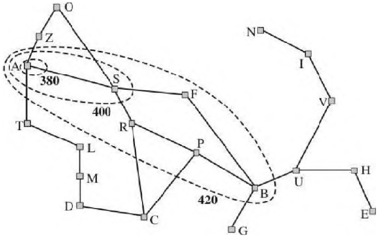
Admissibility and consistency VII

- Fact 2: A* using GRAPH SEARCH is optimal if h is consistent

  Proof: Let G be the first goal node found. By Fact 2, the cost to reach any later goal nodes must be at least as expensive. $\square$

Consistency I

- f-contour: Line with cost C labeled and enclosing states with path costs $\leq C$ Study Figure 3.25 page 97.



- Let C^* be cost of optimal path.
  - A* expands all nodes with $f(n) < C^*$

Consistency II

- Then A* might expand some nodes with $f(node) = C^*$ before selecting a goal node.
- A* is complete.
- Note that A* does not expand nodes with $f(n) > C^*$ – these nodes are pruned.
- A* is good: complete, optimal, optimally efficient
  - Optimally efficient = any other optimal search cannot guarantee that to expand fewer nodes
- Problem: Space may be exponential in length of solution

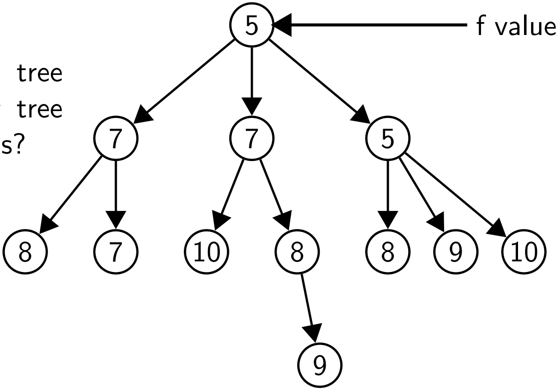
Iterative deeping A* search (IDA*) I

- Recall Iterative Deepening DFS (IDDFS): depth limited search for depth $= 0, 1, 2, 3, 4, \ldots$
- **Iterative deepening A* (IDA*)**: Same as Iterative Deepening DFS except that f-values are used. The cut-off value $=$ smallest of the f-values of any node that exceeded cutoff of **previous** iteration

Iterative deeping A* search (IDA*) II

Cut-off = 5

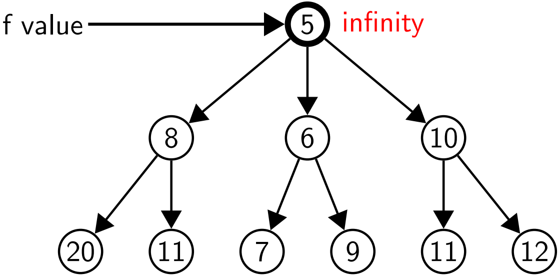
What is the search tree given the following tree of states and actions?



What is the cut-off for the **next iteration**?

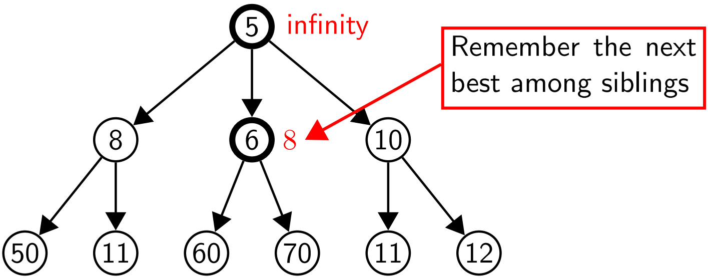
Recursive best first search (RBFS) I

- Keeps track of f-value of **best alternative path** from any **ancestor** of current node



Recursive best first search (RBFS) II

- If f-value of current node exceeds limit, go backward and choose alternative.



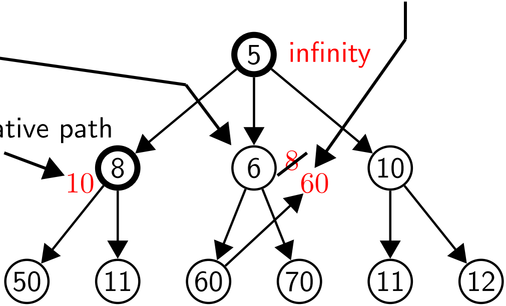
Recursive best first search (RBFS) III

60 is worse than 10. So:

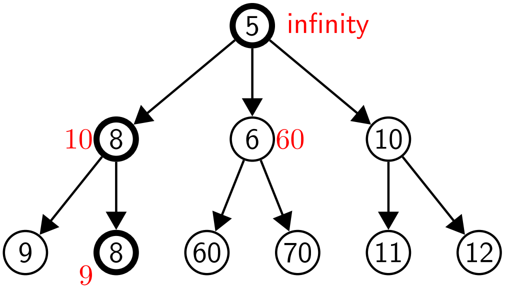
2. Update min path cost to 60

1. Go back up

3. Choose alternative path



Recursive best first search (RBFS) IV



Recursive best first search (RBFS) V

- RBFS more efficient than IDA* but there is still a lot of nodes generated
  - Algorithm can take a wrong path and change it's mind and go backward
  - Optimal if h is admissible
- See algorithm on page 99.

MA* and SMA* I

- A* uses too much memory; IDA* and RBFS uses too little
- MA* and SMA* are variations using more memory
- MA* = memory bound A*
- SMA* = simplified MA*

Effective Branching Factor I

- **Effective branching factor** b^* of a heuristic function:
  - Let N be number of nodes generated by A*
  - Suppose solution depth is d
  - Define b^* by:

$$N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

- Good heuristic function: $b^* \approx 1$

Domination I

- Let h_2, h_1 be heuristic functions.
- h_2 **dominates** h_1 if

$$h_2(n) \geq h_1(n)$$

  for all nodes n.
- If h_2 dominates h_1, then using A*, h_2 will not expand more nodes than h_1
  - except possibly for some nodes n with $f(n) = C^*$
- Heuristic function with higher values are better (unless if it takes a long time to compute)

Generating heuristic functions I

- There are two basic methods for generating heuristic functions, by considering
  - Relaxed problems
  - Subproblems
- When you have more than one possible heuristic function, you can take the max.

Relaxed problems I

- When you relax conditions/rules on a problem, you get a **relaxed problem**.
- $n^2 - 1$: Tile can move from square A to square B if
  - A is horizontally or vertically adjacent to B, and
  - B is blank
- *Relaxed* $n^2 - 1$: Tile can move from square A to square B if
  - B is blank
- Another relaxed $n^2 - 1$: Tile can move from square A to square B if ...
  - NO CONDITION!!!

Relaxed problems II

- Note that for $n^2 - 1$ when we use the last relaxed problem, we get the heuristic function that gives the number of misplaced tiles
- FACT: The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.
  Proof: Exercise.

Subproblems I

- **Subproblem** = remove some goal(s) from a problem
- Example: Subproblem for $n^2 - 1$ problem. Move the tiles so that the *first half* of the tiles are in their correct positions. See diagram on page 106.
- Precompute for each state the number of moves to get first half of tiles to right place and save in database.
- Instead of first half, can also consider bottom half, or even numbered tiles, etc. Combine all using max. (See page 106-107)

Combining heuristic functions I

- But suppose now you have generated more than one heuristic function ... which one to use?!?
- If $h_1, ..., h_m$ are all admissible and you can't decide which is the best, then use

$$h(n) = \max(h_1(n), ..., h_m(n))$$

- FACT: If $h_1, ..., h_m$ are all admissible then so is

$$\max(h_1(n), ..., h_m(n))$$

Proof: Exercise.