Heuristic Search

Sven Koenig, USC

Russell and Norvig, 3rd Edition, Sections 3.5-3.6

These slides are new and can contain mistakes and typos.
Please report them to Sven (skoenig@usc.edu).

1

Survey

- Please respond to Jiaoyang's survey!

2

Skeleton of Search Algorithms

1. Start with a tree that contains only one node, labeled with the start state.
2. If there are no unexpanded fringe nodes, stop unsuccessfully.
3. Pick an unexpanded fringe node n. Let s(n) be the state it is labeled with.
4. If s(n) is a goal state, stop successfully and return the path from the root node to n in the tree.
5. Expand n, that is, create a child node of n for each of the successor states of s(n), labeled with that successor state.
6. Go to 2.

3

Skeleton of Search Algorithms

- The search algorithms differ only in how they select the unexpanded fringe node.
  - If no knowledge other than the current tree is available to guide the decision, then a search algorithm is called uninformed (or blind).
  - Otherwise, a search algorithm is called informed. If the knowledge consists of estimates of the goal distances of the states, the informed search algorithm is called heuristic. By goal distance, we mean the minimum cost of any path (action sequence) from the start state to any goal state.

4

Best-First Search (for a given priority function f)

1. Start with a tree that contains only one node, labeled with the start state.
2. If there are no unexpanded fringe nodes, stop unsuccessfully.
3. Pick an unexpanded fringe node n with the smallest f(n). Let s(n) be the state that it is labeled with.
4. If s(n) is a goal state, stop successfully and return the path from the root node to n in the tree.
5. Expand n, that is, create a child node of n for each of the successor states of s(n), labeled with that successor state.
6. Go to 2.

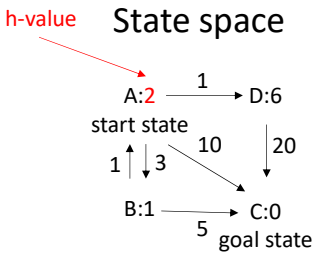
5

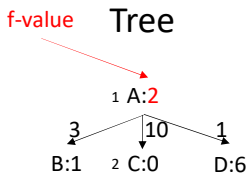
Greedy Best-First Search (operator cost = positive)

1. Start with a tree that contains only one node, labeled with the start state.
2. If there are no unexpanded fringe nodes, stop unsuccessfully.
3. Pick an unexpanded fringe node n with the smallest f(n) = h(s(n)), where s(n) is the state that node n is labeled with and h(s(n)) is an estimate of the goal distance gd(s(n)) of s(n).
4. If s is a goal state, stop successfully and return the path from the root node to n in the tree.
5. Expand n, that is, create a child node of n for each of the successor states of s, labeled with that successor state.
6. Go to 2.

6

Example: Greedy Best-First Search



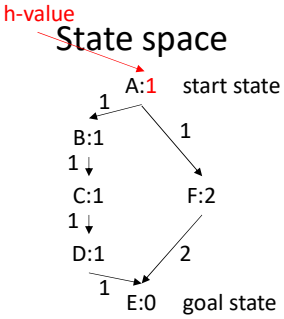


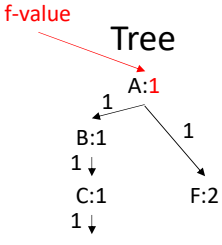
Path: A C
(non-optimal
but often finds paths with
few node expansions)

- Optional pruning rule: do not expand a node if a node labeled with the same state has already been expanded. Thus, we can say that states get expanded rather than nodes.
- Optional termination rule: terminate once a node labeled with a goal state has been generated.

7

Example: Greedy Best-First Search





Why is it a mistake to expand the node labeled with D next?

- Optional pruning rule: do not expand a node if a node labeled with the same state has already been expanded. Thus, we can say that states get expanded rather than nodes.
- Optional termination rule: terminate once a node labeled with a goal state has been generated.

8

A* (operator costs = positive)

1. Start with a tree that contains only one node, labeled with the start state.
2. If there are no unexpanded fringe nodes, stop unsuccessfully.
3. Pick an unexpanded fringe node n with the smallest f(n) = g(n) + h(s(n)), where s(n) is the state that node n is labeled with, g(n) is the cost from the root to n and h(s(n)) is an estimate of the goal distance gd(s(n)) of s(n).
4. If s is a goal state, stop successfully and return the path from the root node to n in the tree.
5. Expand n, that is, create a child node of n for each of the successor states of s, labeled with that successor state.
6. Go to 2.

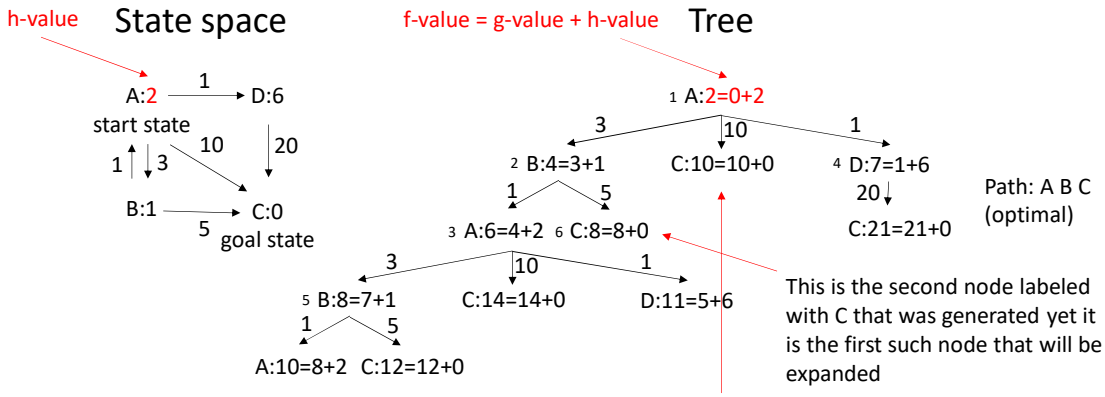
9

A* (operator costs = positive)

- f(n) is an estimate of the cost of a cost-minimal path from the root node (start state) along the tree to node n and from there to any goal state.

10

Example: A* (operator cost = positive)



- ~~Termination rule: terminate once a node labeled with a goal state has been generated.~~

11

Admissible H-Values

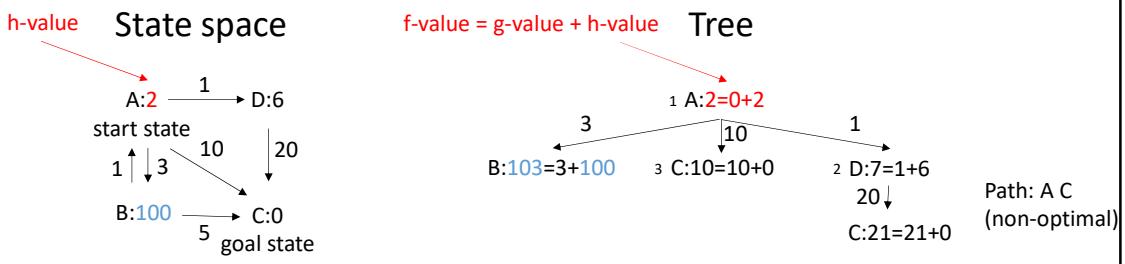
- The h-value (= heuristic value) of a state approximates its goal distance. It should be close to the goal distance without going over (i.e. "optimistic").
- h-values are called admissible if and only if the h-value h(s) of each state s is not larger than its goal distance gd(s):

  $0 \leq h(s) \leq gd(s)$ for all states s.

- We require the h-values to be admissible.
  Otherwise, A* won't be able to find minimum-cost paths.

12

Example: A* (operator cost = positive)

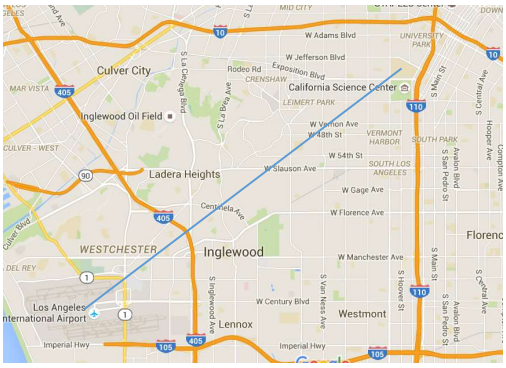


- ~~Termination rule: terminate once a node labeled with a goal state has been generated.~~

13

Admissible H-Values

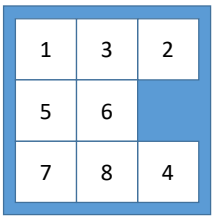
- Find a shortest (not: fastest) path from the USC main campus to the airport
- Straight-line-distance heuristic
  - h(location) = straight-line distance from the location to the airport



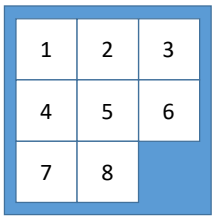
14

Admissible H-Values

- Find a shortest movement sequence that solves the eight-puzzle
- Tiles-out-of-order heuristic (5 for the example below)
  - h(tile configuration) = the number of tiles not at their correct place
- Manhattan-distance heuristic (1+1+3+1+1=7 for the example below)
  - h(tile configuration) = the sum of the x- and y-displacements of each tile from its correct place



current configuration



goal configuration

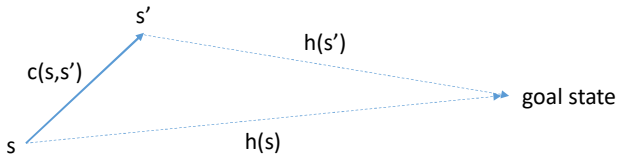
15

Consistent H-Values

- H-values are called consistent if and only if they satisfy the triangle inequality (c(s,s') is the action cost of moving from s to s'):

  h(s) = 0 for all goal states s, and
  $0 \leq h(s) \leq c(s,s') + h(s')$ for all non-goal states s and their successor states s'.

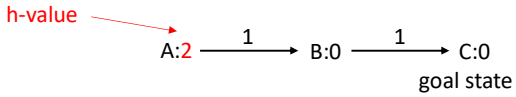
- From here on, we require the h-values to be consistent, not only admissible, for reasons that are explained on the following slides.



16

Consistent H-Values

- Admissible h-values are not necessarily consistent:

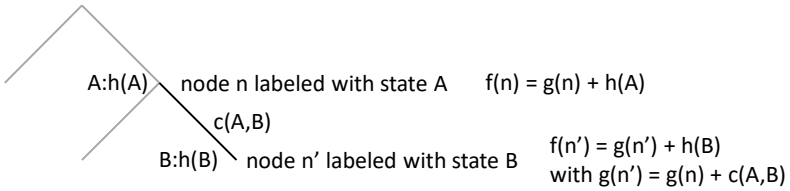


- Consistent h-values are admissible:
  Proof by induction:
  The statement is true for all states s with gd(s) = 0, i.e. all goal states.
  Now pick any non-goal state s.
  Assume that the statement is true for all states s' with gd(s') < gd(s).
  Pick a cost-minimal path from s to any goal state.
  Let s'' be the successor state of s on that path.
  Then, $0 \le h(s) \le c(s,s'') + h(s'') \le c(s,s'') + gd(s'') = gd(s)$. Qed.

17

Consistent H-Values

- Consider a search tree for consistent h-values



- Then, $f(n) = g(n) + h(A) \le g(n) + c(A,B) + h(B) = g(n') + h(B) = f(n')$.
- Thus, the f-values of the children of any expanded node are no smaller than the f-value of the expanded node itself.

18

Consistent H-Values

- Assume that A* picks node n for expansion and that the set of unexpanded fringe nodes at this point in time is OPEN. Then, the f-values of all nodes in OPEN are no smaller than the f-value of node n since A* always picks an unexpanded fringe node with the smallest f-value for expansion (Property A).
- Assume that the set of children of node n after its expansion is C. The f-values of the children of node n are no smaller than the f-value of node n (Property B), see the previous slide.
- (Our argument continues on the next slide…)

19

Consistent H-Values

- After the expansion of node n, the new set of unexpanded fringe nodes is OPEN' := (OPEN\{n})∪C since node n is no longer an unexpanded fringe node but the children of node n have become new unexpanded fringe nodes.
- A* must pick one of the nodes in OPEN' for the next expansion, and the f-values of all nodes in OPEN' are no smaller than the f-value of node n according to (Property A) and (Property B).
- Thus, A* expands nodes in order of non-decreasing f-values. That is, a node that A* expands later than some other node has an f-value that is no smaller than the f-value of the other node.

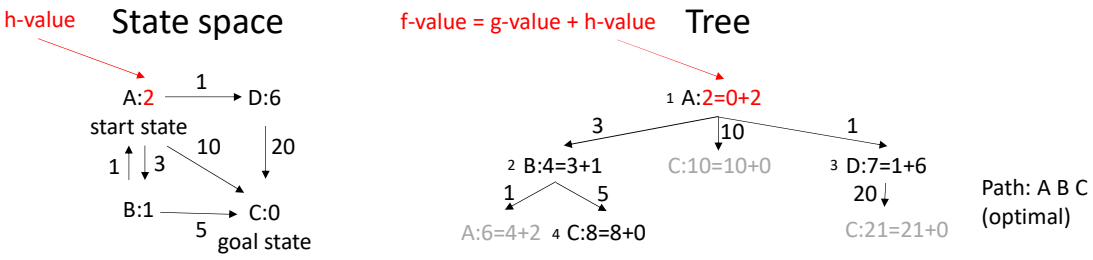
20

Consistent H-Values

- Now assume that A* expands a node n labeled with state s and later another node n' labeled with the same state s. Then,
  - f(n) ≤ f(n')
  - g(n) + h(s) ≤ g(n') + h(s)
  - g(n) ≤ g(n')
- Thus, the first node that A* expands has the smallest g-value among all nodes labeled with the same state that A* expands. Remember that the g-value of a node corresponds to the length of the path in the tree from the root node to the node, that is, the length of a path found in the state space from the start state to the state that labels the node. A* thus does not need to expand any nodes labeled with the same state as a node that it has already expanded!

21

Example: A* (operator cost = positive)



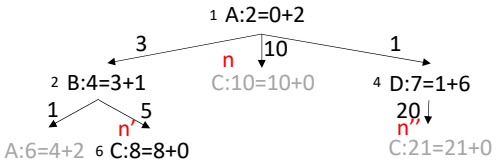
- Optional pruning rule: do not expand a node if a node labeled with the same state has already been expanded. Thus, we can say that states get expanded rather than nodes.
- ~~Termination rule: terminate once a node labeled with a goal state has been generated.~~

22

Implementation of A*

In this case (if the optional pruning rule is used), every node in the search tree is labeled with a different state – and one can now talk about "states" in the search tree instead of "nodes."

- For each state, maintain at most one node labeled with it, namely the one with the smallest f-value so far (the node might or might not have been expanded already).



- Maintain the unexpanded fringe nodes in a heap (often called OPEN list) with their f-values as keys. Always choose the "top" of the heap (a node in the heap with a smallest f-value) for expansion. Break ties among nodes with the smallest f-value in favor of nodes with larger g-values.

23

Problem Relaxation

- Obtain a new planning problem by relaxing constraints of the actions (e.g. by deleting preconditions of operator schemata), which can add states and actions to the state space.
- Typically, this is done in a way so that the goal distances for the new planning problem can be computed without search.
- Use the goal distance of a state for the new planning problem as the h-value of the state for the original planning problem.
- The resulting h-values are consistent and thus also admissible.
- Many human-created admissible h-values can be explained as resulting from this process. Thus, in practice, many human-created admissible h-values are consistent!

24

Problem Relaxation

- Find a shortest (not: fastest) path from the USC main campus to the airport
- Straight-line-distance heuristic
  - h(location) = straight-line distance from the location to the airport
  - Relaxation: one can drive on- and off-roads

25

Problem Relaxation

- Find a shortest movement sequence that solves the eight-puzzle
- Tiles-out-of-order heuristic (5 for the example below)
  - h(tile configuration) = the number of tiles not at their correct place
  - Relaxation: one can move any tile to any place in one move, even if that place is already occupied by another tile
- Manhattan-distance heuristic (0+1+1+3+1+1=7 for the example below)
  - h(tile configuration) = the sum of the x- and y-displacements of each tile from its correct place
  - Relaxation: one can move any tile from its current place to any adjacent place, even if that place is already occupied by another tile

26

Consistent H-Values

- To verify that h-values are consistent,
  - either prove that the triangle inequality holds or
  - show that they can result from a problem relaxation.
- To create consistent h-values,
  - create admissible h-values and verify that they are consistent.

27

Dominating H-Values

- A* expands nodes in order of non-decreasing f-values. Let gd* be the goal distance of the start state or, equivalently, the g-value and f-value of the node labeled with a goal state that A* is about to expand when it terminates. Then, A* expands
  - all nodes n with f(n) < gd*, and
  - no nodes n with f(n) > gd*.

28

Dominating H-Values

- H-values h(s) dominate h-values h'(s) if and only if, for all states s, $h(s) \geq h'(s)$.
- Consider consistent h-values h(s) and h'(s) where the h-values h(s) dominate the h-values h'(s). Then, A* with h'(s) expands at least all nodes n that A* with h(s) expands, except perhaps for some states n whose f-values under both searches equal their goal distances.
  Proof: Consider any state n expanded by A* with h(s). Then, $g(n) + h(s(n)) = f(n) \leq gd^*$, which implies that $h'(s(n)) \leq h(s(n)) \leq gd^* - g(n)$. Thus, either $h'(s(n)) = h(s(n)) = gd^* - g(n)$, i.e. $f'(n) = f(n) = gd^*$, or $h'(s(n)) < gd^* - g(n)$, i.e. $f'(n) = g(n) + h'(s(n)) < gd^*$ and A* with h'(s) expands n as well. Qed.

29

Dominating H-Values

- Given consistent h-values h(s) and h'(s) where the h-values h(s) dominate the h-values h'(s). Then, A* with h'(s) and A* with h(s) both find cost-minimal paths but A* with h(s) runs at least as fast (in terms of node expansions) as A* with h'(s), perhaps up to tie-breaking among nodes whose f-values equal their goal distances.
- Note: This does not take into account that calculating the h-values h(s) and h'(s) can take different amounts of time!

30

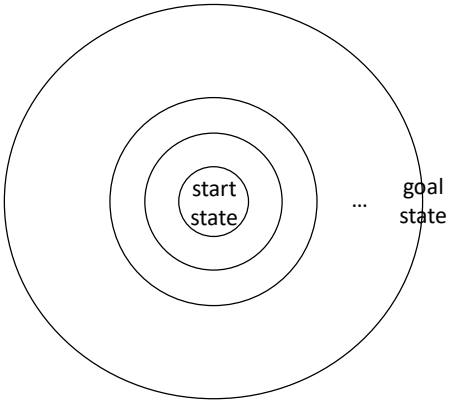
Examples: Dominating H-Values

- The tiles-out-of-order h-values and the Manhattan-distance h-values are both consistent (since they result from problem relaxations), and the Manhattan-distance h-values dominate the tiles-out-of-order h-values. Thus, you want to use A* with the Manhattan-distance h-values rather than A* with the tiles-out-of-order h-values.
- Given two consistent h-values h(s) and h'(s), the h-values max(h(s),h'(s)) are consistent and dominate both h(s) and h'(s) (prove it yourself). Thus, you want to use A* with max(h(s),h'(s)) rather than A* with h(s) or A* with h'(s).

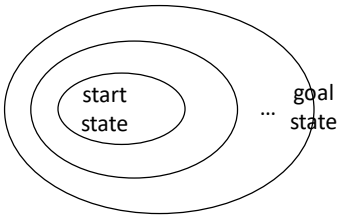
31

Uninformed Search vs. Informed Search

Uniform cost search (A* with h(s) = 0)



A*

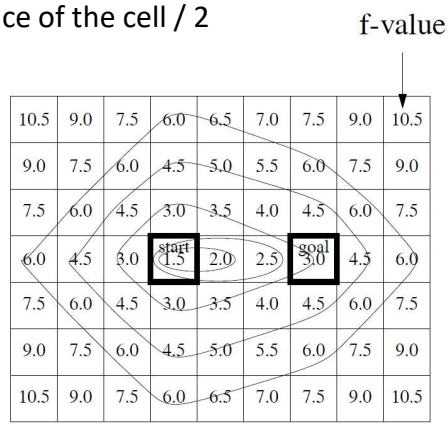


iso f-value contours

32

Uninformed Search vs. Informed Search

- Example
  - Grid world in which one can move N, E, S and W with cost 1
  - h(cell) = goal distance of the cell / 2



33

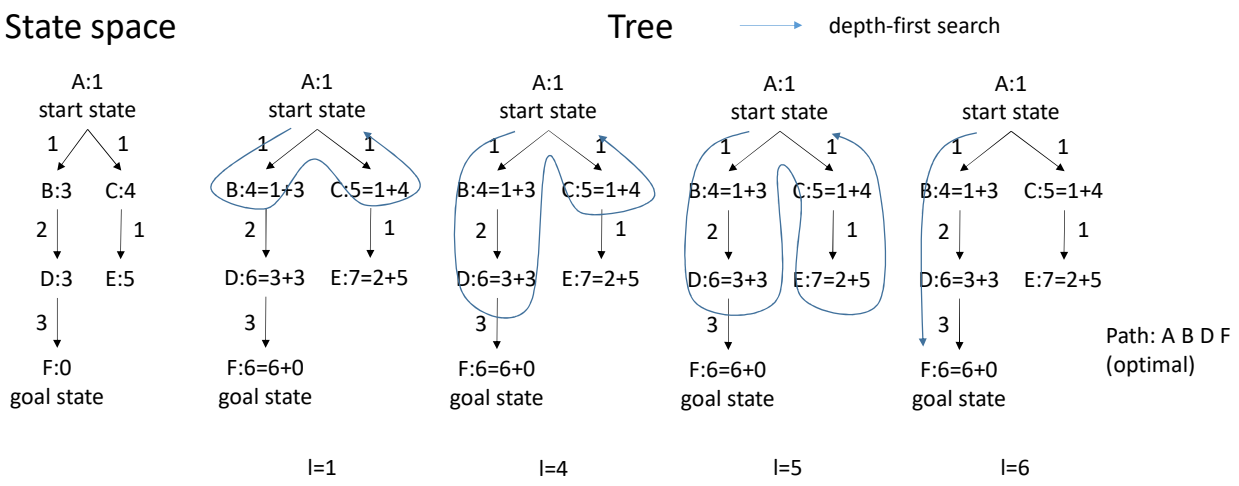
Iterative Deepening A* (operator cost = positive)

- Combine the best properties of A* and depth-first searches, which can be necessary since A* still needs an exponential amount of memory
- Implement an A* search with a series of depth-first searches with increasing f-value limits (that is, depth-first searches that assume that nodes whose f-values are larger than the depth limit have no children).

1. l := h(start).
2. Perform a depth-first search with f-value limit l.
3. If a node n with f-value l and labeled with a goal state was expanded, stop successfully and return the path from the root node to n in the tree.
4. If no node with f-value larger than l was expanded, stop unsuccessfully.
5. l := the smallest f-value of any expanded node whose f-value is larger than l.
6. Go to 2.

34

Example: Iterative Deepening A* (= IDA*)



35

Example: Iterative Deepening A* (= IDA*)

- The overhead of Iterative Deepening over Breadth-First Search (i.e. the percentage of additional node expansions) is often smaller than the overhead of Iterative Deepening A* over A*.
- The reason is that there are often more nodes with the same g-value [= all of them get expanded for the first time during the same Depth-First Search of Iterative Deepening] when all action costs are one than there are nodes with the same f-value [= all of them get expanded for the first time during the same Depth-First Search of Iterative Deepening A*] (especially when all action costs are different).

36

Heuristic Search

- Want to play around with heuristic search algorithms?
- Go here: http://aispace.org/search/

37