Assignment 2

Benjamin Sorenson

October 8, 2015

Question 1: (a) Formulate the problem:

As a complete-state

State Space

All possible arrangements of N guests around the table.

State Representation

Each state could be represented as an array A of integers in $[0, N]$—0 would represent and empty seat at the table, and any number $k \geq 1$ in position i (i.e., $A[i] = k$) would mean that person k is seated at position i.

Initial State

The initial state could be any arrangement of N guests at the table

Goal Test

All guests are seated, and the total group satisfaction cannot be improved by changing the position of any one of the guests.

Actions

$INSERT - RIGHT(S, i, j)$ given a state S, assign guest in position i to position j, and shift the guests to the right of j right one position.

$INSERT - LEFT(S, i, j)$ Similar to $INSERT - RIGHT$, but guests to the left of j are shifted left one position.

$SWAP(i, j)$ Swap guest in position i with guest in position j

As an incremental formulation

State Space

All possible arrangements of $0 \leq n \leq N$ guests around the table

State Representation

Same as above

Initial State

No guests seated at the table.

Actions

All the same actions as above with the addition of

$SEAT(S, k, i)$ given a state S seat person k in open position i

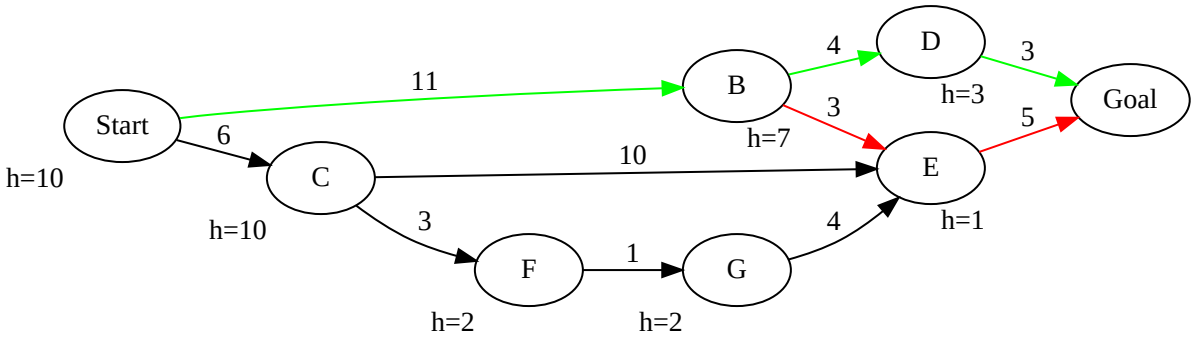
Goal Test

Same as above

(b) The advantage of the complete-state formulation is that the state space is smaller than the incremental formulation. There are "only" $(n-1)!$ possible state in the complete-state formulation, and there are $\sum_{i=0}^{N-1} i!$ states in the incremental formulation. The incremental solution might be easier to solve since it's essentially the same problem as the complete-state formulation, but we can construct the initial state.

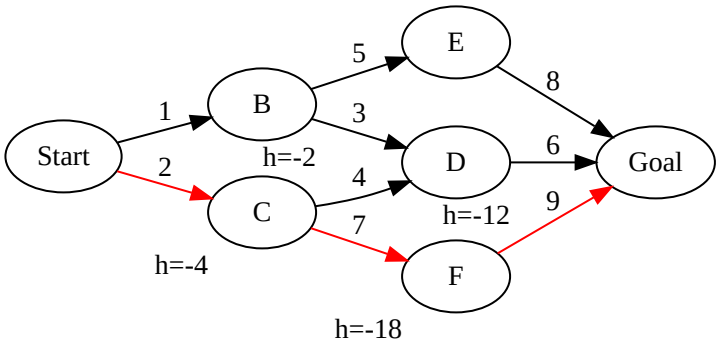
(c) Since we're trying to maximize $Pleasure(i, j)$, in order to be admissible, our heuristic has to be $\geq \max\{Pleasure(i, j)\}$–an attempt at this might be $h(n) = N \cdot Pleasure(i_{\max}, j_{\max})$ where i_{max} and j_{max} are the guests in state S who derive the most pleasure from the arrangement. This isn't admissible since in the case when there are two guests, we can't guarantee that $h(n)$ is an overestimate.

Question 2: (a) No, greedy best-first search can at best find a solution equal in cost to what A^* will find since A^* is guaranteed to find an optimal solution if one exists. For example, using the example graph from class, the green path shows the path found (eventually) by A^* search, and the red path shows where greedy best-first deviated from the optimal path



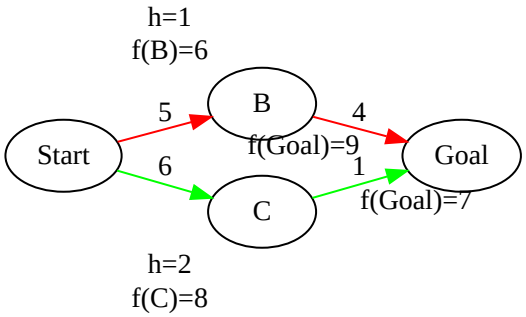
(b) Yes, in general greedy best-first search will be faster A^* search since it stops as soon as it reaches the goal. A^* search, however, will continue until it has found the lowest cost path. A^* can get lucky and find the lowest cost solution as fast a greedy best-first search, but in general it will be slower. It's difficult to show this graphically, but again, taking the example from class, A^* search had to expand every node at least once (E three times) before finding the optimal path. By contrast, greedy best-first search expanded just two nodes.

(c) When $h(n) = -2 \cdot g(n)$, greedy best-first search behaves like a hill climbing algorithm. It evaluates each node by assigning it a negative value equal to twice the path cost of reaching that node. Thus, taking the node that most increases the total cost—much like hill-climbing. The following example show best-first search on with $h(n)$ defined as above. Since $h(n)$ depends on $g(n)$, the value of $h(n)$ is shown only for children of expanded nodes.

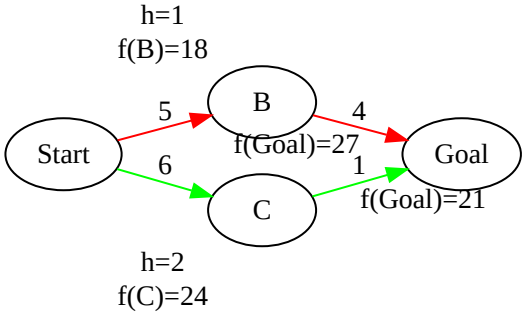


Question 3: (a) Yes. Since we are only multiplying $f(n)$ by a constant, the ordering of the nodes and path costs remains unchanged so not only will A^* still find an optimal path, but its behavior will be exactly the same. See the following example. The first path taken is in red, and the second, optimal path taken is in green. The first graph shows $f(n)$ defined as it usually is, and the second graph shows $f(n)$ defined as $f(n) = 3 \cdot g(n) + 3 \cdot h(n)$.

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



$$f(n) = 3 \cdot g(n) + 3 \cdot h(n)$$



(b) I will assume that $0 \leq w \leq 1$. From part (a), it is easy to see that a value of $w = .5$ will still guarantee an optimal solution, $w = 0$ (uniform cost search) will guarantee an optimal solution, but $w = 1$ (greedy best-first search), will not. From this alone, we can conclude that the answer depends on the value of w

(c) $g(n)$ and $h(n)$ control the shape of search contours of the search—emphasizing $g(n)$ widens the contours, and emphasizing $h(n)$ narrows the contours.

Question 4: (a) $h_3(n) = \max\{h_1(n), h_2(n)\}$ is admissible. Since both h_1 and h_2 are admissible, the maximum of of h_1 and h_2 will also be admissible. I would use h_3 instead of either h_1 or h_2 because h_3 will be at least as accurate as either h_1 or h_2 alone—the more accurate the heuristic, the more narrow the search becomes around the optimal path.

(b) $h_4(n) = \max\{h_1(n), 1.1 \cdot h_2(n)\}$ may not be admissible. Since $1.1 \cdot h_2(n) \geq h_2(n)$, $\max\{h_1(n), 1.1 \cdot h_2(n)\}$ may overestimate the cost to reach the goal.

(c) $h_5(n) = \min\{h_1(n), 2 \cdot h_2(n)\}$ is admissible since $h_5(n) \leq h_1(n)$

(d) $h_6(n) = \frac{h_1(n)+h_2(n)}{2}$ is admissible since $h_6(n) < h_3(n)$

(e) $h_7(n) = \frac{h_1(n)}{3} + \frac{h_2(n)}{2}$ is admissible since $h_7(n) < h_6(n)$

Question 5: (a) Suppose that A^* left some node with $f(n) < C^*$ closed after finding the optimal solution. By the formulation of the algorithm, it would then expand the node, and continue.

(b) Breadth-first search will expand all nodes until the goal is reached—completeness is unrelated to paths cost.

(c) A^* cannot be used for online search. In order to search a state space A^* must be used in a n environment where the resultant state of an action on the current state can be known. By definition, the resultant state of an action in on-line search cannot be known prior to the action being completed. Therefore A^* search cannot be used for on-line search.