CSC384 - Introduction to Artificial Intelligence
Sample Problems
(Winter 2018)

1 Search

1. It would seem that iterative deepening search should have a higher asymptotic time complexity than breadth-first search because every time the depth-bound is increased it must start its search from scratch. However this is not true. Why?

   If the children of breadth first search are expanded as BFS executes, the order of BFS will be $O(b^{d+1})$. Iterative deepening may execute many repeated searches at each iteration, but none will be more than $O(b^d)$. A fixed number of searches that are each bound by $O(b^d)$ takes $O(b^d)$ time to run, collectively.

2. If $h()$ is admissible and s is the start node, how is $h(s)$ related to the cost of the solution found by A* search?

   $h(s)$ must be less than $h^*(s)$ where $h^*(s)$ is the true cost of the path from s to the goal.

3. What happens if we use a heuristic $h()$ in A* search that does not have the guarantee that $h(n) \leq h^*(n)$ for all states n?

   If it is not the case that $h(n) \leq h^*(n)$ for all nodes, our search will favor some nodes for expansion that are on suboptimal paths to the goal over nodes that are on optimal paths to the goal. This means A-star search will no longer be gauranteed to yield an optimal solution.

4. How do depth-first, breadth-first and depth-first with iterative deepening search compare in terms of their asymptotic time complexity? In terms of their asymptotic space complexity? Please indicate any assumptions you are using (i.e. justifications from the textbook, from other sources, etc).

   (a) *BFS: $O(d^{b+1})$ in both space and time (with some caveats. Assumes we expand nodes in layer d prior to discovering the goal. May however be $O(b^d)$ as per RN ed. 3. Also, this analysis assumes state space may not be able to be explicitly represented).*

   (b) *DFS: $O(b^{dMax})$ time but only $O(b * dMax)$ space.*

   (c) *UCS: $O(b^{C^*/e})$ space and time (with some caveats. Maintenance of ordered frontier adds to space and time complexity ... typically employs a priority queue (which takes*

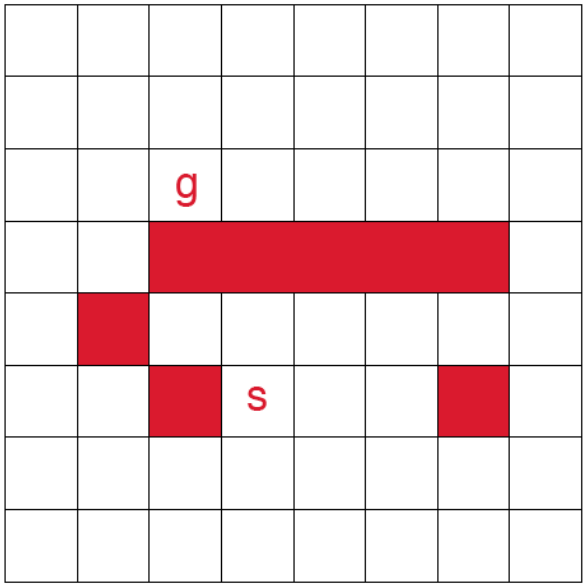

Figure 1: Search Space

logarithmic time to update). Additional caveat is that, if the state space can be explicitly represented, time and and space bounds can be reduced.

(d) Iterative Deepening: $O(d^b)$ time and $O(b * d)$ space.

5. Consider the problem of finding a path in the grid shown below from the position s to the position g. The robot can move on the grid horizontally and vertically, one square at a time (each step has a cost of one). No step may be made into a forbidden shaded area. The search space and the shaded areas are illustrated in Figure 1.

   - On the grid, number the nodes in the order in which they are removed from the frontier in a depth-first search from s to g, given that the order of the operators you will test is: up, left, right, then down. Assume there is a cycle check.
     Refer to Figure 2.
     D6,D5,C5,E5,D5,G5,H5,H4,H3,H2,H1,G1,
     F1,E1,D1,C1,B1,A1,A2,B2,C2,D2,E2,F2,
     G2,G3,F3,E3,D3,C3.
   - Number the nodes in order in which they are taken off the frontier for an A* search for the same graph. Manhattan distance should be used as the heuristic function. That is, $h(n)$ for any node n is the Manhattan distance from n to g. The Manhattan distance between two points is the distance in the x-direction plus the distance in the y-direction. It corresponds to the distance traveled along city streets arranged in a grid. For example, the Manhattan distance between g and s is 4. What is the path that is found by the A* search?

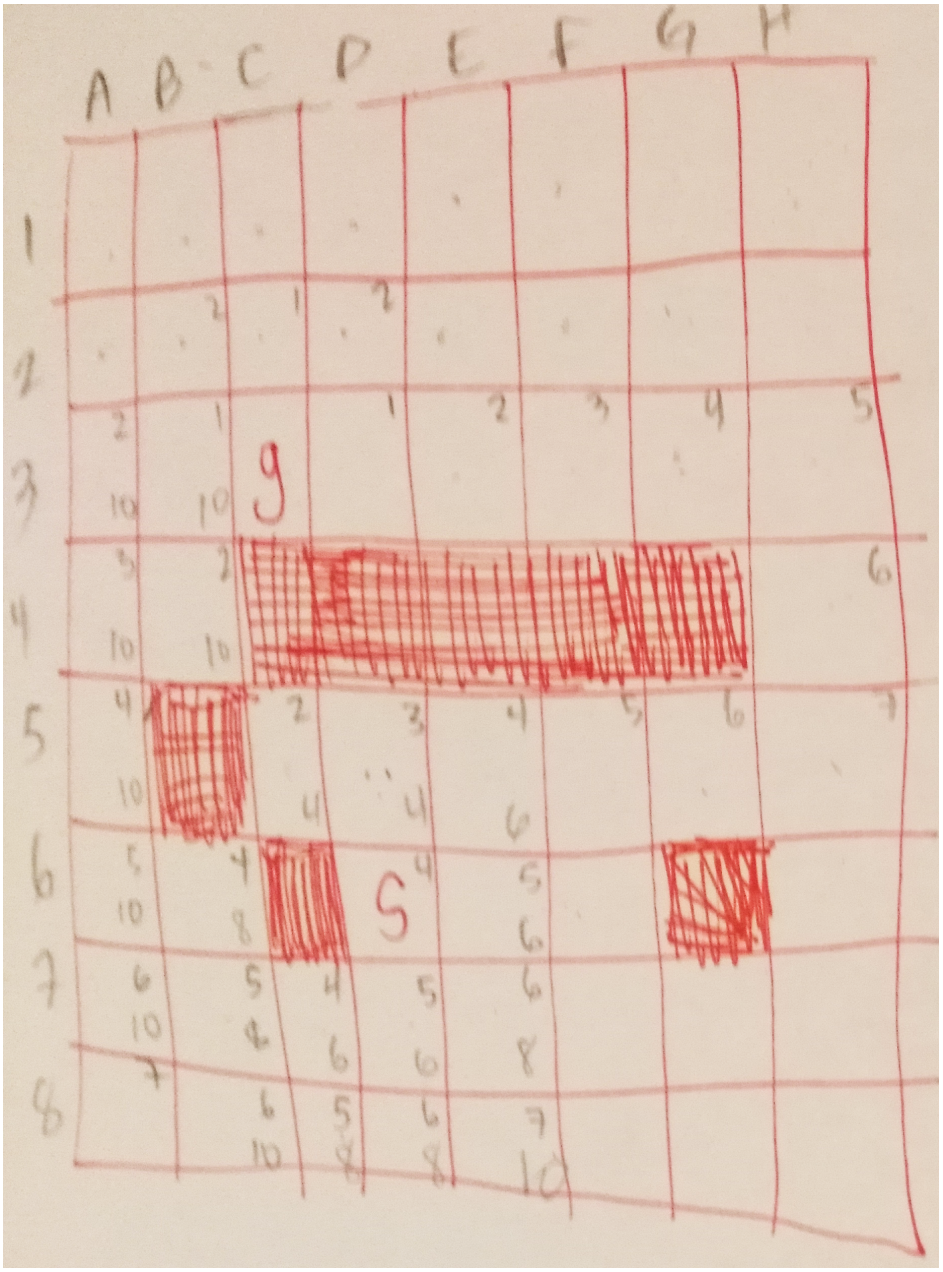

Figure 2: This illustrates the numbering scheme used for the solutions. Note that in the top right of each corner is the Manhattan Distance heuristic value to the goal. The bottom right numbers are "f-values" (i.e. the Manhattan Distance heuristic value + cost from the start.

Refer to Figure 2.
Order of nodes pulled off frontier depends and will vary depending on how people choose to break ties between equal f-values.
first f-contour (value is 4) (nodes are first to be expanded): D6,D5
second f-contour (value is 6) (nodes are second to be expanded): C5,E5,E6,D7,C7
third f-contour (value is 8) (nodes are next to be expanded): E7,D8,C8,B7,B6
final f-contour (value is 10) (nodes are next to be expanded): E8,B8,A7,A6,A5,A4,B4,B3,A3,C3
path discovered: D6,D7,C7,B7,B6,A6,A5,A4,B4,B3,C3

- Suppose that the graph extended infinitely in all directions. That is, there is no boundary, but s, g, and the forbidden area are in the same relative positions to each other. Which search methods would no longer find a path? Would A*, or would depth-first search? Which would be the better method, and why?

  DFS will travel infinitely off the board at the upper right corner. It will not find a path and is therefore not the best method to use. A, by contrast, will be drawn to states that are relatively close to the goal (i.e. that have a low f-value). It will find the optimal path, even if the board is infinite.*

2 Constraint Satisfaction Problems

1. *True or False:* Deleting values from the domains of CSP variables that prevent the constraint graph from being arc consistent will never rule out a possible solution to the CSP.

   True. A solution will include values from each variable's domain that are supported by values in other variables' domains. Values that have support won't be pruned when evaluating arc consistency.

2. State the condition under which an constraint that operates over two variables (X, Y) is generalized arc consistent.

   A constraint is arc consistent over two variables X and Y if, for each value in the domain of X there is a corresponding value in the domain of Y that will satisfy the constraint (and vice versa).

3. Consider a CSP with the following variables and constraints:

   - Variables A, B, C, D, E with all variables having the domain $1, 2, 3, 4$
   - Constraints:
     - $E - A$ is even.
     - $C \neq D$
     - $C > E$
     - $C \neq A$
     - $B > D$
     - $D > E$
     - $B > C$

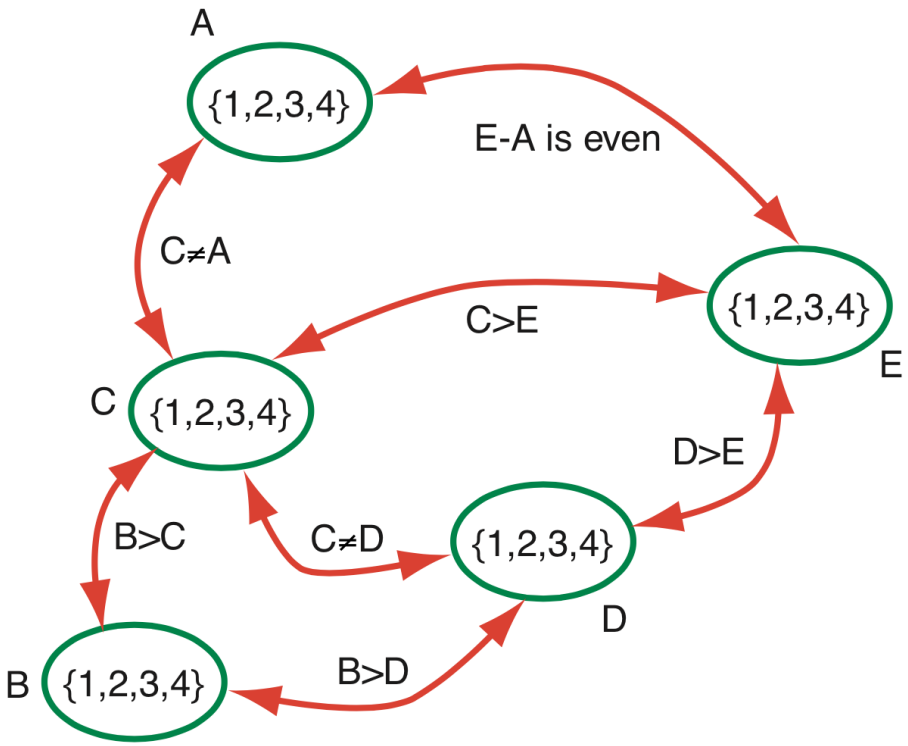

Figure 3: Constraint Network

Draw these variables and constraints as a constraint graph. Then, perform GAC-Enforce to create a graph that is generalized arc consistent. Write the domains for each variable that exist after you've completed GAC-Enforce.

GAC enforce will cut the following values from the domains:

```
Arc: <B,C> removes 1 from the domain of B
Arc: <C,B> removes 4 from the domain of C
Arc: <C,E> removes 1 from the domain of C
Arc: <E,C> removes 3 & 4 from the domain of E
Arc: <D,E> removes 1 from the domain of D
Arc: <B,D> removes 2 from the domain of B
Arc: <D,B> removes 4 from the domain of D
```

3 Game Tree Search

1. The following URL provides a great tool for understanding the details of minimax with alpha beta pruning: http://inst.eecs.berkeley.edu̅cs61b/fa14/ta-materials/apps/ab_tree_practice/

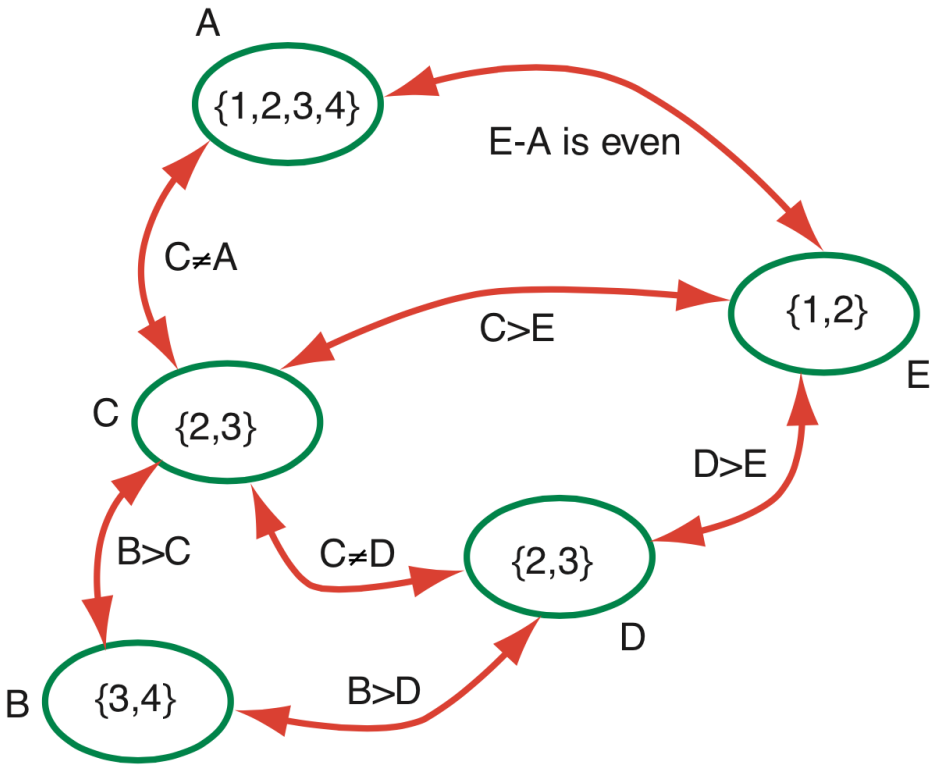

Figure 4: Constraint Network AFTER GAC ENFORCE.

2. Assume you have come up with a minimax policy after traversing to all the terminals of a game tree. Someone then comes along and doubles the value of every terminal over some threshold. Will you come up with the same strategy if you run minimax again? What if someone doubles only values of terminals that are even?

   If you maintain the relative ordering or the terminals in a minimax tree, you'll still end up coming up with the same strategy even after you apply some transform to the terminals. So doubling the terminals won't change the strategy, and doubling terminals over a threshold won't either. If you double only even values, tho, the relative ordering may be changed so there are no guarantees you'll come up with the same strategy.

3. Why is minimax search generally not used to play real games?

   To generate a minimax strategy you must search every game state, all the way to terminal game states, and then back up values to the root. So, if each game state has b successors and terminals are at a depth d, minimax must expand $O(b^d)$ states to generate a strategy. It's not typically possible to explore that many game states during real game play, as real games may have very big branching factors and great depth.

4. *True or False*: When executing the alpha-beta algorithm on a game tree which is traversed from left to right, the leftmost branch will never be pruned.

   True.

5. *True or False*: The alpha-beta algorithm will always result in at least one branch of the game tree being pruned.

   False.