

**INSTRUCTIONS**

- You have 1.5 hours.
- The exam is closed book, closed notes except a one-page crib sheet.
- Please use non-programmable calculators only.
- Mark your answers **ON THE EXAM ITSELF**. If you are not sure of your answer you may wish to provide a *brief* explanation. All short answer sections can be successfully answered in a few sentences at most.
- Questions are not sequenced in order of difficulty. Make sure to look ahead if stuck on a particular question.

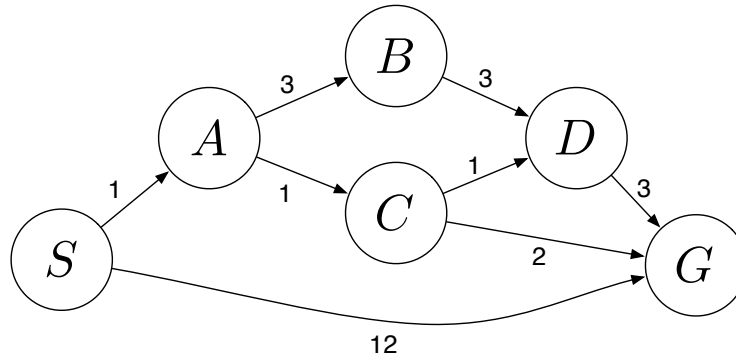
Last Name	
First Name	
SID	
Login	
<i>All the work on this exam is my own.</i> <b>(please sign)</b>	

**For staff use only**

Q. 1	Q. 2	Q. 3	Q. 4	Q. 5	Q. 6	Total
/12	/12	/12	/ 21	/ 24	/19	/100

THIS PAGE INTENTIONALLY LEFT BLANK

## 1. (12 points) Search



Answer the following questions about the search problem shown above. Break any ties alphabetically. For the questions that ask for a path, please give your answers in the form ‘ $S - A - D - G$ .’

(a) (2 pt) What path would breadth-first graph search return for this search problem?

$S - G$

(b) (2 pt) What path would uniform cost graph search return for this search problem?

$S - A - C - G$

(c) (2 pt) What path would depth-first graph search return for this search problem?

$S - A - B - D - G$

(d) (2 pt) What path would A\* graph search, using a consistent heuristic, return for this search problem?

$S - A - C - G$

(e) (4 pt) Consider the heuristics for this problem shown in the table below.

State	$h_1$	$h_2$
$S$	5	4
$A$	3	2
$B$	6	6
$C$	2	1
$D$	3	3
$G$	0	0

i. (1 pt) Is  $h_1$  admissible? **Yes** **No**

ii. (1 pt) Is  $h_1$  consistent? **Yes** **No**

iii. (1 pt) Is  $h_2$  admissible? **Yes** **No**

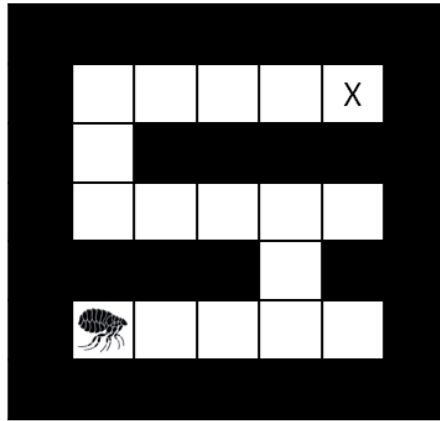
iv. (1 pt) Is  $h_2$  consistent? **Yes** **No**

**2. (12 points) Hive Minds: Redux**

Let's revisit our bug friends from assignment 2. To recap, you control one or more insects in a rectangular maze-like environment with dimensions  $M \times N$ , as shown in the figures below. At each time step, an insect can move North, East, South, or West (but not diagonally) into an adjacent square if that square is currently free, or the insect may stay in its current location. Squares may be blocked by walls (as denoted by the black squares), but the map is known.

For the following questions, you should answer for a general instance of the problem, not simply for the example maps shown.

**(a) (6 pt) The Flea**



You now control a single flea as shown in the maze above, which must reach a designated target location  $X$ . However, in addition to moving along the maze as usual, your flea can jump on top of the walls. When on a wall, the flea can walk along the top of the wall as it would when in the maze. It can also jump off of the wall, back into the maze. Jumping onto the wall has a cost of 2, while all other actions (including jumping back into the maze) have a cost of 1. Note that the flea can only jump onto walls that are in adjacent squares (either north, south, west, or east of the flea).

- i. (2 pt) Give a *minimal* state representation for the above search problem.

The location of the flea as an  $(x, y)$  coordinate.

- ii. (2 pt) Give the size of the state space for this search problem.

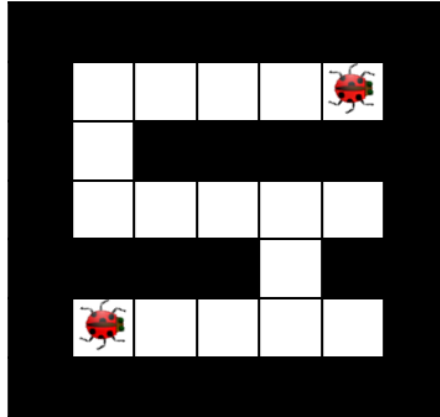
$M * N$

- iii. (2 pt) Is the following heuristic admissible? **Yes** **No**

$h_{\text{flea}}$  = the Manhattan distance from the flea to the goal.

If it is *not* admissible, provide a nontrivial admissible heuristic in the space below.

## (b) (6 pt) Long Lost Bug Friends



You now control a pair of long lost bug friends. You know the maze, but you do not have any information about which square each bug starts in. You want to help the bugs reunite. You must pose a search problem whose solution is an all-purpose sequence of actions such that, after executing those actions, both bugs will be on the same square, regardless of their initial positions. Any square will do, as the bugs have no goal in mind other than to see each other once again. Both bugs execute the actions mindlessly and do not know whether their moves succeed; if they use an action which would move them in a blocked direction, they will stay where they are. Unlike the flea in the previous question, bugs *cannot* jump onto walls. Both bugs can move in each time step. Every time step that passes has a cost of one.

- i. (2 pt) Give a *minimal* state representation for the above search problem.

A list of boolean variables, one for each position in the maze, indicating whether the position could contain a bug. You don't keep track of each bug separately because you don't know where each one starts; therefore, you need the same set of actions for each bug to ensure that they meet.

- ii. (2 pt) Give the size of the state space for this search problem.

$$2^{MN}$$

- iii. (2 pt) Give a nontrivial admissible heuristic for this search problem.

$h_{\text{friends}}$  = the maximum Manhattan distance of all possible pairs of points the bugs can be in.

### 3. (12 points) A\* Graph Search

```

function A* GRAPH SEARCH(problem)
  fringe ← an empty priority queue
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  closed ← an empty set
  ADD INITIAL-STATE[problem] to closed
  loop
    if fringe is empty then
      return failure
    end if
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then
      return node
    end if
    for successor in GETSUCCESSORS(problem, STATE[node]) do
      if successor not in closed then
        ADD successor to closed
        fringe ← INSERT(MAKE-SUCCESSOR-NODE(successor, node), fringe)
      end if
    end for
  end loop
end function

```

The above implementation of A\* graph search may be incorrect! In the list below circle all of the problems that the bugs may cause when executing graph search and justify your answer. Note that the fringe is a priority queue. Nodes are inserted into the fringe using the standard key for A\*, namely  $f = g + h$ .  $h$  is a consistent heuristic.

- (a) The GetSuccessors function could be called multiple times on the same state.
- (b) The algorithm is no longer complete.
- (c) The algorithm could return a suboptimal solution.
- (d) The implementation is incorrect, but none of the above problems will be caused.
- (e) The implementation is correct.

To receive any credit you must briefly justify your answer in space below.

The bug is the insertion of *successor* into *closed* at time of insertion of a node into the fringe, rather than at the time that node gets popped from the fringe. As a consequence of this bug, the first path encountered to a state will put that state in the closed list. This can cause suboptimality as we only have the guarantee that a state has been reached optimally once a node reaching it gets popped off the fringe.

- (a) is False as when a node that reaches a state  $s$  is placed in the fringe, that state  $s$  is also put on the closed list. This means never in the future can a node be placed on the fringe that ends in that same state  $s$ , and hence the same state  $s$  can be the argument to GetSuccessors at most once.
- (b) is False. A\* tree search is complete. The difference is that the above algorithm will cut off parts of the tree search whenever it has placed a node on the fringe in the past that ends in the same state. So compared to tree search we only lose copies of subtrees that we are covering. Hence the above algorithm is complete.
- (c) is True. See explanation at beginning of solution.
- (d) is False.
- (e) is False.

**4. (21 points) Time Management**

Two of our GSIs, Arjun and Woody, are making their schedules for a busy morning. There are five tasks to be carried out:

- (F) Pick up food for the group's research seminar, which, sadly, takes one precious hour.
- (H) Prepare homework questions, which takes 2 consecutive hours.
- (P) Prepare the PR2 robot for a group of preschoolers' visit, which takes one hour.
- (S) Lead the research seminar, which takes one hour.
- (T) Teach the preschoolers about the PR2 robot, which takes 2 consecutive hours.

The schedule consists of one-hour slots: 8am-9am, 9am-10am, 10am-11am, 11am-12pm. The requirements for the schedule are as follows:

- (a) In any given time slot each GSI can do at most one task (F, H, P, S, T).
- (b) The PR2 preparation (P) should happen before teaching the preschoolers (T).
- (c) The food should be picked up (F) before the seminar (S).
- (d) The seminar (S) should be finished by 10am.
- (e) Arjun is going to deal with food pick up (F) since he has a car.
- (f) The GSI not leading the seminar (S) should still attend, and hence cannot perform another task (F, T, P, H) during the seminar.
- (g) The seminar (S) leader does not teach the preschoolers (T).
- (h) The GSI who teaches the preschoolers (T) must also prepare the PR2 robot (P).
- (i) Preparing homework questions (H) takes 2 consecutive hours, and hence should start at or before 10am.
- (j) Teaching the preschoolers (T) takes 2 consecutive hours, and hence should start at or before 10am.

To formalize this problem as a CSP, use the variables F, H, P, S and T. The values they take on indicate the GSI responsible for it, and the starting time slot during which the task is carried out (for a task that spans 2 hours, the variable represents the starting time, but keep in mind that the GSI will be occupied for the next hour also - make sure you enforce constraint (a)!). Hence there are eight possible values for each variable, which we will denote by A8, A9, A10, A11, W8, W9, W10, W11, where the letter corresponds to the GSI and the number corresponds to the time slot. For example, assigning the value of A8 to a variables means that this task is carried about by Arjun from 8am to 9am.

- (a) (2 pt) What is the size of the state space for this CSP?

$8^5$ .

- (b) (2 pt) Which of the statements above include unary constraints?

(d), (e), (i), (j). (i) and (j) are both unary constraints, and binary constraints in a single sentence.

- (c) (4 pt) In the table below, enforce all unary constraints by crossing out values in the table on the left below. If you made a mistake, cross out the whole table and use the right one.

F	A8	A9	A10	A11	<del>W8</del>	<del>W9</del>	<del>W10</del>	<del>W11</del>
H	A8	A9	A10	<del>A11</del>	W8	W9	W10	<del>W11</del>
P	A8	A9	A10	A11	W8	W9	W10	W11
S	A8	A9	<del>A10</del>	<del>A11</del>	W8	W9	<del>W10</del>	<del>W11</del>
T	A8	A9	A10	<del>A11</del>	W8	W9	W10	<del>W11</del>

- (d) (3 pt) Start from the table above, select the variable S and assign the value A9 to it. Perform forward checking by crossing out values in the table below. Again the table on the right is for you to use in case you believe you made a mistake.

F	A8	A9	<del>A10</del>	<del>A11</del>	W8	W9	<del>W10</del>	<del>W11</del>
H	<del>A8</del>	<del>A9</del>	A10	<del>A11</del>	W8	W9	W10	<del>W11</del>
P	A8	A9	A10	A11	W8	W9	W10	W11
S	A8	A9	A10	A11	W8	W9	<del>W10</del>	<del>W11</del>
T	A8	A9	A10	A11	W8	W9	W10	<del>W11</del>

- (e) (3 pt) Based on the result of (d), what variable will we choose to assign next based on the MRV heuristic (breaking ties alphabetically)? Assign the first possible value to this variable, and perform forward checking by crossing out values in the table below. Again the table on the right is for you to use in case you believe you made a mistake.

Variable F is selected and gets assigned value A8.

F	A8	A9	A10	A11	W8	W9	W10	W11
H	A8	A9	A10	A11	W8	W9	W10	W11
P	A8	A9	A10	A11	W8	W9	W10	W11
S	A8	A9	A10	A11	W8	W9	W10	W11
T	A8	A9	A10	A11	W8	W9	W10	W11

Have we arrived at a dead end (i.e., has any of the domains become empty)?

No.

- (f) (4 pt) We return to the result from enforcing just the unary constraints, which we did in (c). Select the variable S and assign the value A9. Enforce arc consistency by crossing out values in the table below.

F	A8	A9	A10	A11	W8	W9	W10	W11
H	A8	A9	A10	A11	W8	W9	W10	W11
P	A8	A9	A10	A11	W8	W9	W10	W11
S	A8	A9	A10	A11	W8	W9	W10	W11
T	A8	A9	A10	A11	W8	W9	W10	W11

- (g) (2 pt) Compare your answers to (d) and to (f). Does arc consistency remove more values or less values than forward checking does? Explain why.



Arc consistency removes more values. It's because AC checks consistency between any pair of variables, while FC only checks the relationship between pairs of assigned and unassigned variables.

- (h) (1 pt) Check your answer to (f). Without backtracking, does any solution exist along this path? Provide the solution(s) or state that there is none.

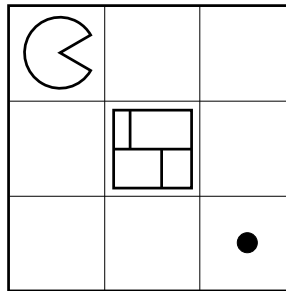
AC along this path gives 1 solution: F: A8 H: A10 P: W8 S: A9 T: W10



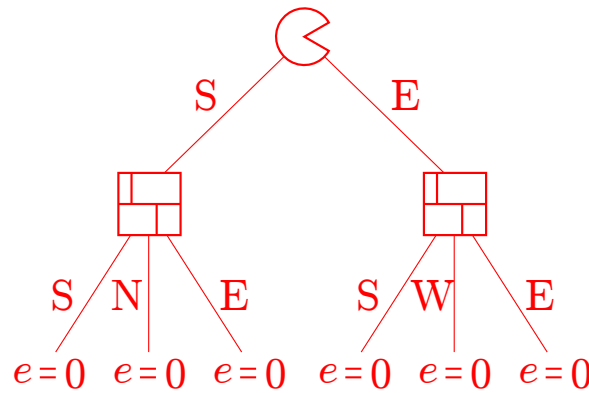
**5. (24 points) Surrealist Pacman**

In the game of Surrealist Pacman, Pacman  plays against a moving wall . On Pacman's turn, Pacman must move in one of the four cardinal directions, and must move into an unoccupied square. On the wall's turn, the wall must move in one of the four cardinal directions, and must move into an unoccupied square. The wall cannot move into a dot-containing square. Staying still is not allowed by either player. Pacman's score is always equal to the number of dots he has eaten.

The first game begins in the configuration shown below. Pacman moves first.



(a) (2 pt) Draw a game tree with one move for each player. Draw only the legal moves.



(b) (1 pt) According to the depth-limited game tree you drew above what is the value of the game? Use Pacman's score as your evaluation function.

0. All leaves have value 0.

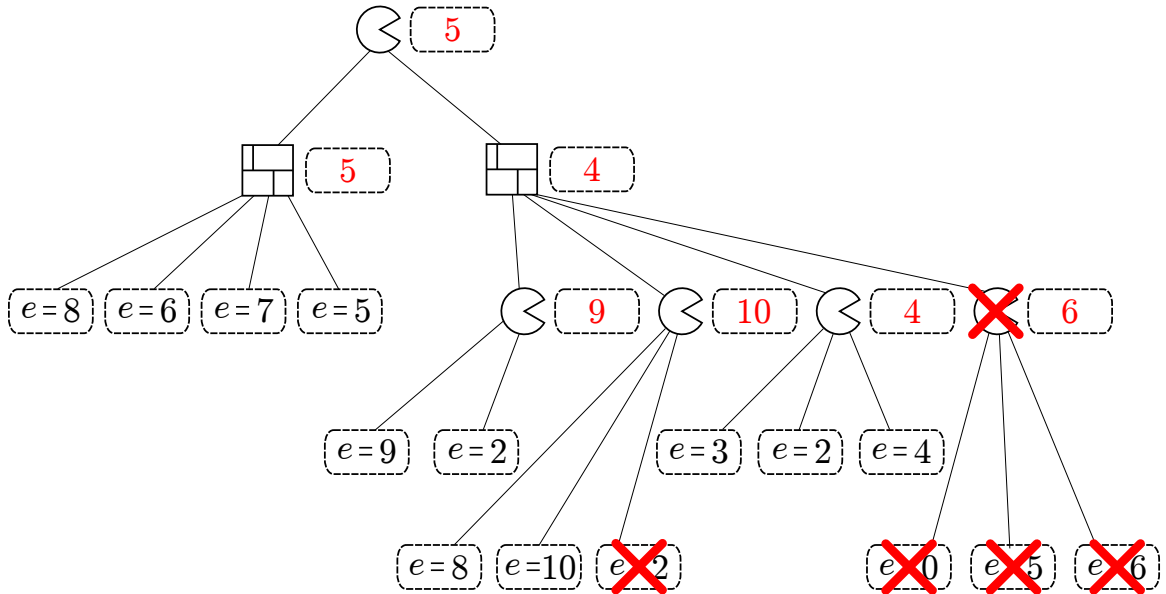
(c) (1 pt) If we were to consider a game tree with ten moves for each player (rather than just one), what would be the value of the game as computed by minimax?

1. Pacman can force a win in ten moves.

A second game is played on a more complicated board. A partial game tree is drawn, and leaf nodes have been scored using an (unknown) evaluation function  $e$ .

(d) (3 pt) In the dashed boxes, fill in the values of all internal nodes using the minimax algorithm.

(e) (3 pt) Cross off any nodes that are not evaluated when using alpha-beta pruning (assuming the standard left-to-right traversal of the tree).



(Filling values returned by alpha-beta or not crossing off children of a crossed off node were not penalized.)

Suppose that this evaluation function has a special property: it is known to give the correct minimax value of any internal node to within 2, and the correct minimax values of the leaf nodes exactly. That is, if  $v$  is the true minimax value of a particular node, and  $e$  is the value of the evaluation function applied to that node,  $e - 2 \leq v \leq e + 2$ , and  $v = e$  if the node is a dashed box in the tree below.

Using this special property, you can modify the alpha-beta pruning algorithm to prune more nodes.

- (f) (10 pt) Standard alpha-beta pseudocode is given below (only the max-value recursion). Fill in the boxes on the right to replace the corresponding boxes on the left so that the pseudocode prunes as many nodes as possible, taking account of this special property of the evaluation function.

```

function MAX-VALUE(node,  $\alpha$ ,  $\beta$ )
   $e \leftarrow$  EVALUATIONFUNCTION(node)
  if node is leaf then
    return  $e$ 
  end if
   (1)
   $v \leftarrow -\infty$ 
  for child  $\leftarrow$  CHILDREN(node) do
     (2)
    if  $v \geq \beta$  then
      return  $v$ 
    end if
     $\alpha \leftarrow$  MAX( $\alpha$ ,  $v$ )
  end for
  return  $v$ 
end function
  
```

Fill in these boxes:

(1)

```

if  $e - 2 \geq \beta$  then
  return  $e - 2$ 
end if
  
```

(2)

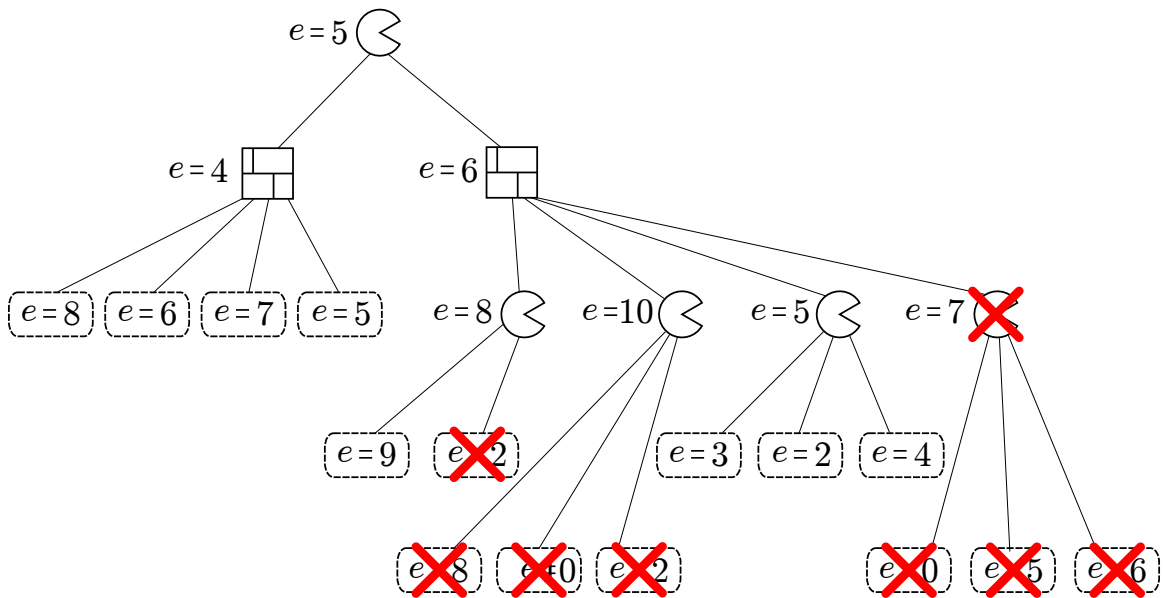
```

 $v \leftarrow$  MAX( $v$ , MIN-VALUE(child,
  MAX( $\alpha$ ,  $e - 2$ ), MIN( $\beta$ ,  $e + 2$ )))
  
```

(Variations are possible.)

The same game tree is shown below, with the evaluation function applied to *internal* as well as leaf nodes.

- (g) (4 pt) In the game tree below cross off any nodes that can be pruned assuming the special property holds true. If not sure you correctly formalized into pseudo-code your intuition on how to exploit the special property for improved pruning, make sure to annotate your pruned nodes with a brief explanation of why each of them was pruned.



## 6. (19 points) Multiple Choice and Short Answer Questions

Each true/false question is worth 1 point. Leaving a question blank is worth 0 points. **Answering incorrectly is worth -1 point.**

For the questions that are not True/False, answer as concisely as possible (and no points are subtracted for a wrong answer to these).

- (a) (7 pt) Consider a graph search problem where for every action, the cost is at least  $\epsilon$ , with  $\epsilon > 0$ . Assume the used heuristic is consistent.
- True False** Depth-first graph search is guaranteed to return an optimal solution.  
**False. Depth first search has no guarantees of optimality.**
  - True False** Breadth-first graph search is guaranteed to return an optimal solution.  
**False. Breadth first search has no guarantees of optimality unless the actions all have the same cost, which is not the case here.**
  - True False** Uniform-cost graph search is guaranteed to return an optimal solution.  
**True.**
  - True False** Greedy graph search is guaranteed to return an optimal solution.  
**False. Greedy search makes no guarantees of optimality.**
  - True False** A\* graph search is guaranteed to return an optimal solution.  
**True, since the heuristic is consistent in this case.**
  - True False** A\* graph search is guaranteed to expand no more nodes than depth-first graph search.  
**False. Depth-first graph search could, for example, go directly to a sub-optimal solution.**
  - True False** A\* graph search is guaranteed to expand no more nodes than uniform-cost graph search.  
**True. The heuristic could help to guide the search and reduce the number of nodes expanded. In the extreme case where the heuristic function returns zero for every state, A\* and UCS will expand the same number of nodes. In any case, A\* with a consistent heuristic will never expand more nodes than UCS.**
- (b) (2 pt) Iterative deepening is sometimes used as an alternative to breadth first search. Give one advantage of iterative deepening over BFS, and give one disadvantage of iterative deepening as compared with BFS. Be concise and specific! **Advantage: iterative deepening requires less memory. Disadvantage: iterative deepening repeats computations and therefore can require additional run time.**
- (c) (2 pt) Consider two different A\* heuristics,  $h_1(s)$  and  $h_2(s)$ , that are each admissible. Now, combine the two heuristics into a single heuristic, using some (not yet specified) function  $g$ . Give the choice for  $g$  that will result in A\* expanding a minimal number of nodes while still guaranteeing admissibility. Your answer should be a heuristic function of the form  $g(h_1(s), h_2(s))$ .  
 **$\max(h_1(s), h_2(s))$**
- (d) (3 pt) Let  $h_1(s)$  be an admissible A\* heuristic. Let  $h_2(s) = 2h_1(s)$ . Then:
- True False** A\* tree search with  $h_2$  is guaranteed to return the shortest path.  
**False.  $h_2$  is not guaranteed to be admissible.**
  - True False** A\* tree search with  $h_2$  is guaranteed to return a path which is at most twice as long as the optimal path.  
**True. A partial plan corresponding to the optimal path could be sitting on the fringe when the (possibly non-optimal) solution is returned. The maximum f-cost of this optimal partial plan is twice the true optimal cost. Therefore the path returned has cost less than or equal to twice the true optimal path.**
  - True False** A\* graph search with  $h_2$  is guaranteed to return the shortest path.  
**False.  $h_2$  is not guaranteed to be admissible.**
- (e) (2 pt) Consider a CSP with three variables:  $A$ ,  $B$ , and  $C$ . Each of the three variables can take on one of two values: either 1 or 2. There are three constraints:  $A \neq B$ ,  $B \neq C$ , and  $A \neq C$ . In the table below, cross off all values that are eliminated by enforcing *arc-consistency*. Also write one sentence interpreting your answer.

A	1	2
B	1	2
C	1	2

No variables should be crossed out. There is no solution to this CSP, but arc-consistency does not detect this.

- (f) (3 pt) Consider an adversarial game tree where the root node is a maximizer, and the minimax value of the game is  $v_M$ . Now, also consider an otherwise identical tree where every minimizer node is replaced with a chance node (with an arbitrary but known probability distribution). The expectimax value of the modified game tree is  $v_E$ .

**True False**  $v_M$  is guaranteed to be less than or equal to  $v_E$ .

True. The maximizer is guaranteed to be able to achieve  $v_M$  if the minimizer acts optimally. He can potentially do better if the minimizer acts suboptimally (e.g. by acting randomly).

**True False** Using the optimal *minimax* policy in the game corresponding to the modified (chance) game tree is guaranteed to result in a payoff of at least  $v_M$ .

True. The minimax policy is always guaranteed to achieve payoff of at least  $v_M$ , no matter what the minimizer does.

**True False** Using the optimal *minimax* policy in the game corresponding to the modified (chance) game tree is guaranteed to result in a payoff of at least  $v_E$ .

False. In order to achieve  $v_E$  in the modified (chance) game, the maximizer may need to change his or her strategy. Moreover, even if the maximizer followed the expectimax strategy, he or she is only guaranteed  $v_E$  in expectation.