CSE 415–Autumn 2017 — Midterm Examination
SOLUTIONS

by the Staff of CSE 415, Autumn 2017

INSTRUCTIONS: Write your full name at the top of this cover page. Make sure you have all 8 pages. **Also write your name on top of every page or it is possible that you would not receive credit for that page.** (We will be removing staples from the exams, and if a page of yours gets lost due to not having your name on it, you could miss credit for that page.)

Put your answers inside the framed rectangular boxes that are provided. Write legibly and if you use a pencil, make sure that you write darkly enough that a normal scanner will pick up your writing. If you need more space, use the margins. Do all problems. This is a CLOSED-BOOK, CLOSED-NOTES examination. Do not use any books, notes, calculators, or other electronic devices. There are five problems worth 20 points each for a total of 100. Each problem has multiple parts, and the allocations of points among the parts are as shown on each individual problem.

1. Short Answer (20 points)

(a) (5 points) In state-space search, a set of operators specifies how new states can be produced from existing states. Why is the state-transformation function component of an operator in general only a partial function? (Partial means not necessarily defined on the whole domain. Here our domain is the set of possible states.)

A particular operator's state-transformation function might not be applicable to some particular state. For example, in Tic-Tac-Toe, an operator described as "Put an X in the center" is not applicable if it is O's turn or if there is already something in the center square. Thus the function is undefined on some parts of the domain which is the set of possible states of the problem or game. (The operator's precondition is used to determine whether or not the state-transformation function is applicable.)

(b) (5 points) Using some of the following symbols, write a formula that expresses the condition under which a heuristic function $h_i(s)$ is admissible, where $h(s)$ is the true shortest distance from s to the nearest goal. Σ represents the set of possible states.

$$h, s, h_i, \leq, \geq, <, >, 0, \in, \Sigma, \forall, \exists, (,)$$

$$(\forall s \in \Sigma)(h_i(s) \leq h(s))$$

(c) (5 points) In search using basic hill climbing, what is the main hazard?

If the objective function is not convex, then Hill climbing runs the risk of arriving at a local maximum that is not the global maximum, with no way to get down or to reliably go towards the global maximum.

(d) (5 points) Explain how to determine the size of the state space for a Tower of Hanoi puzzle with n disks. (We assume 3 pegs, as usual.)

For each of the n disks, there are three possible pegs for it to be on. Taking the largest disk first, placing it on any peg (with 3 choices), and then taking the next disk and placing it on any peg, etc., we have $\overbrace{3 \cdot 3 \cdot \cdots \cdot 3}^{n} = 3^n$ choices.

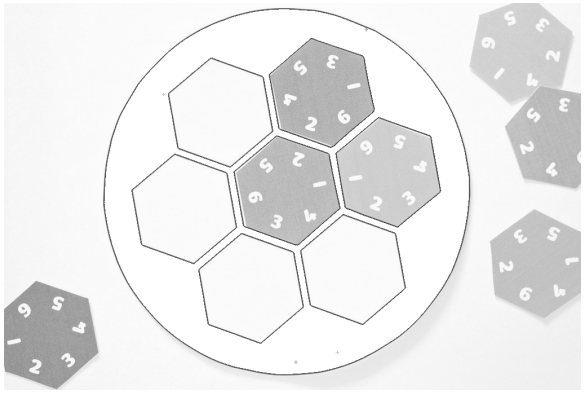
2. (20 points) Python

Complete a class definition for a Poker hand, by completing the methods that have stubs provided. The Card class is given for reference and you do not need to change it.

```
class Card:
  def __init__(self, suit, rank)
    self.suit = suit
    self.rank = rank

class PokerHand:
  def __init__(self):  # For creating an empty hand.
    # Initialize a list that will be used to hold the cards.

    self.cards = []

  def insert(self, card): # Use this to insert a card into the hand.
    # If there are fewer than 5 cards in the list, then
    #    put the card into the list and return True.
    # otherwise return False.

    if len(self.cards) < 5:
      self.cards.append(card)
      return True
    else: return False

  def copy(self): # create and return a copy of the hand.
    # It is not necessary to make copies of the separate cards,
    # but the list of cards must be copied in the new PokerHand object.

    newHand = PokerHand()
    newHand.cards = self.cards[:]
    return newHand

  def isFlush(self):  # If the poker hand is a flush, return True.
    # To be a flush, there must be 5 cards, and all their suits must be
    # equal.
    # Otherwise return False.

    if len(self.cards) < 5: return False
    firstSuit = self.cards[0].suit
    for nextCard in self.cards[1:]:
       if nextCart.suit != firstSuit: return False
    return True
```

3. Determining the Sizes of State Spaces (20 points)

Let us consider the combinatorics of a class of puzzles we will call "Labeled Hexagon" puzzles. In one of these puzzles, seven hexagonal tiles are to be placed into a tray that has a central place for one hexagon and six places around it for the other hexagons. Each hexagon has its sides labeled with numbers from 1 to 6, but the numbers are not necessarily all used or distinct; for example, there could be a hexagon whose sides are all labeled 1 or a hexagon whose sides are alternately labeled 3 and 5. They are like the Painted Squares from the reading in that the available markings for the sides can be arranged in any manner. (Dominoes also have this property.) An example is shown in the figure below.



(a) If all the hexagonal tiles had all their sides labeled with only the number 1, but each of the 7 hexagons used a different color, how many different ways could we fill the tray with the 7 hexagons?

$$7! = 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 5040$$

(b) Now let's assume the hexagons still have different colors, but that they each have a pattern of all 6 numbers, as in the figure. In addition to the choices of what tile to put into each place, as in part (a), let's also have a choice for each tile of 6 different rotational orientations for it. Give an expression for the total number of ways to fill the tray. Here we won't require that numbers that meet on the sides of hexagons match each other. (However, they should match if the state is a goal state.)

For each of the 7! permutations of the tiles, there are 6 choices of rotational orientation for each of the 7 tiles. Thus the total number of possibile arrangements is

$$7! \cdot 6^7 = 5040 \cdot 279936 = 1410877440$$

(c) Finally, consider that an algorithm for finding a suitable arrangement of the tiles, starting from an empty tray will deal with many partially-filled tray arrangements. The initial state corresponds to an empty tray, and successors of the initial state each have

one tile placed somewhere in the tray. Give an expression for the total number of states in the state space for this problem. (Assume operators are available to choose any unplaced piece to go in any vacant space, and to rotate it to any of its 6 orientations.) Briefly explain your formula.

Starting with 0 pieces on the board and going up to 7 pieces on the board, we add up the number of ways to place k pieces on the board. For example, when $k = 3$, we have $\binom{7}{3}$ ways to select the 3 places on the board to be filled and, $7 \cdot 6 \cdot 5 = 7!/(7-3)!$ ways to select a sequence of 3 tiles to fill those positions, and 6^3 rotational variations of the three chosen tiles leading to

$$(7!/(3!4!)) \cdot (7!/4!) \cdot 6^3$$

Summing up the general formula for all 8 values of k, we have

$$\sum_{k=0}^{7} (7!/(7-k)!) \binom{7}{k} (6^k)$$

4. Heuristic Search (20 points)

In a fast-action video game, one of the enemy agents (the "bot") uses the A* algorithm to find a path to its goal. The game takes place in a 2D grid world. The bot has operators that correspond to 1-step moves to go either north, east, west, or south. The cells marked B are blocked. There are obstacles in those cells, and the bot is not allowed to go into them, but must travel around them in order to get to G. It uses the following heuristic function in its search. Here $x(s)$ is the x-coordinate of the state s, and $y(s)$ is its y-coordinate.

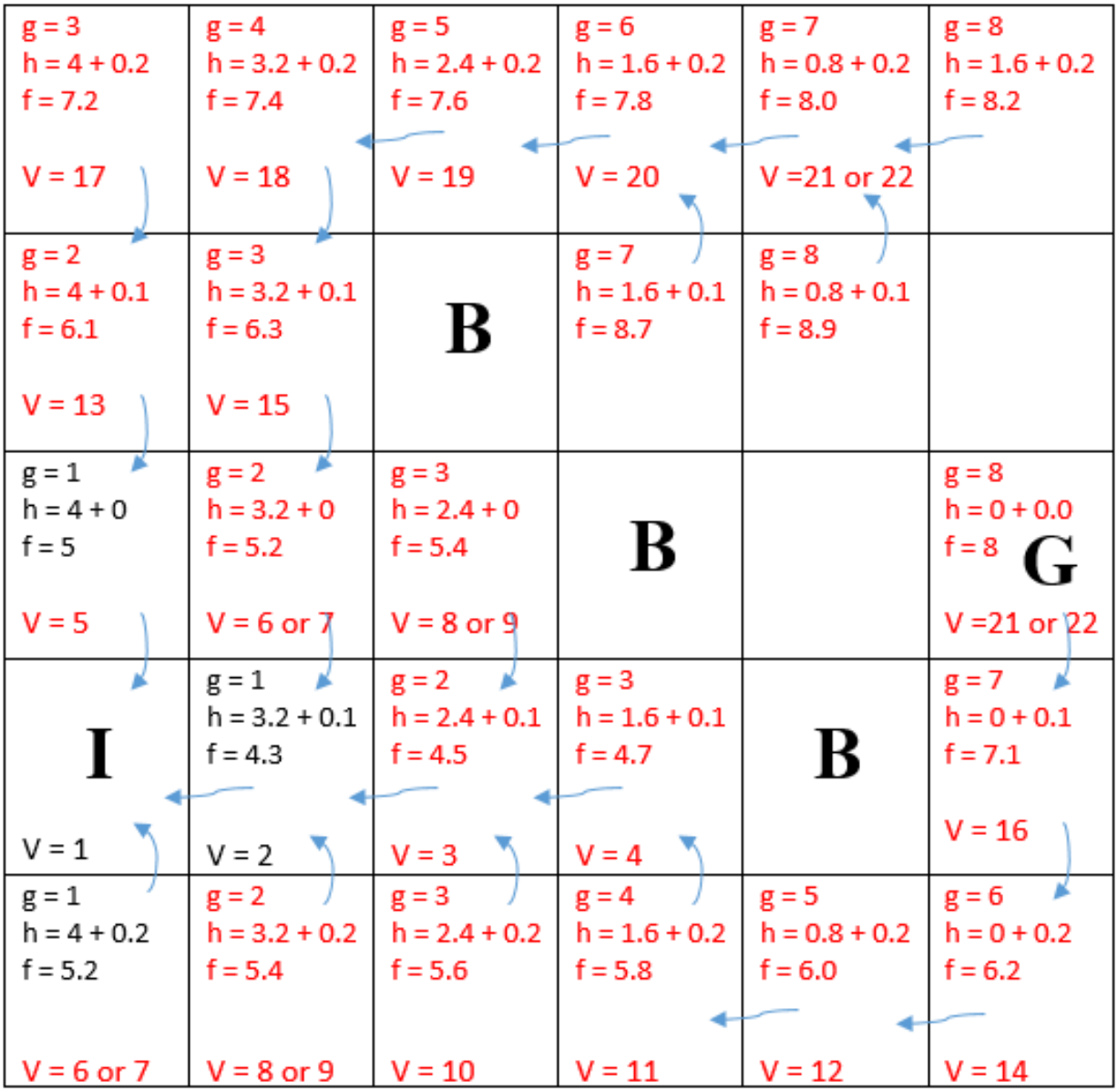
$$h(s) = 0.8 \cdot |x(G) - x(s)| + 0.1 \cdot |y(G) - y(s)|$$

A simulation of the A* algorithm has been started. The first iteration of the algorithm has already been performed for you, and you can see that the initial state is the first state visited (V = 1); and the g values of each successor are 1, because those successors are one step from I. The h value of the north successor of I is 4 + 0, because the difference in x coordinates between it and the goal is 5, and multiplying that by 0.8 gives us 4, and the difference between the y coordinates is 0. Among the three successors of I, the one with lowest f is the one to the east having $f = 4.3$. So that is chosen at the beginning of the second iteration, and it has V=2 indicated.

(a) (5 points) Is h admissible? Why or why not? What is the significance of this?

Yes. It can never overestimate the true distance, which is the minimum number of steps in directions N,E,W,or S needed to reach the goal going around obstacles if needed. $h(s)$ is 0.8 times the horizontal distance plus 0.1 times the vertical distance to G, which will always be less than the true distance except equal when $s = G$. The significance is that A* using h will always return a shortest path from s to G when it arrives at G, and with that guarantee the search can stop as soon as it visits G.

(b) (15 points) Finish simulating the A* algorithm to find a path from I to G. Whenever a successor is generated by the algorithm, write the g, h, and f values for that state as shown. Show the visitation order of the states (the order in which states come off the OPEN priority queue via the DELETEMIN operation.) Also, draw the backpointer links that the algorithm uses to keep track of the parentage of each state generated.



Note: Where ties occur in priorities for the visitation order, they are shown as, e.g., "V = 6 or 7". If the tie at V = 21 or 22 is broken in favor of G, then the other will not be visited, and its two new successor (each having g=8, near the upper right corner) will not be generated.

5. (20 points)

Given $D = \{a, b, c, d, e, f, g\}$ and
$R = \{(a,a), (a,b), (a,d), (b,c), (c,c), (d,b), (d,d), (e,e), (e,f), (e,g), (f,f), (f,g), (g,e), (g,g)\}$

(a) (4 points) Draw a graph of R:

(b) (2 points) We can make the relation reflexive over D by adding one pair. What is it?

(b, b)

(c) (3 points) We can make the relation antisymmetric by *removing* one pair. What is it?

(g, e). We could alternatively remove (e, g) though we would have to put it back to help achieve transitivity in the next part.

(d) (3 points) We can make the relation transitive by removing one pair. What is it?

(b, c)

(e) (8 points) Explain why it makes sense to argue (i) below, but not (ii):

(i) A whale is a mammal. Mammals bear live young. Therefore whales bear live young.

(ii) There are two types of sub-atomic particles. A boson is a sub-atomic particle. Therefore, there are two types of bosons.

Although in both (i) and (ii) we are discussing classes of objects, the forms of inheritance that these arguments attempt to use are different. In the second premise of (i) the class that bears live young is a subclass of mammals consisting of females that generate offspring, i.e., mothers. This is not explicitly stated, but is what the reader is intended to infer. From the first premise we understand that all whales are mammals, including those whales that are mothers. The inference we make is that every whale that is a mother bears live young as all mammalian mothers do. In (ii) The first premise states that there are two types of sub-atomic particles. This is not a property of any individual member of the class of sub-atomic particles. It is a property of the class as a whole, not an instance (member) of the class. It is not inherited either by any subclass (e.g., bosons) or any members of the subclass. Inheritance only works when the property to be inherited is a property of each individual member of the parent class, not the parent class as a whole or the population of all members.

Your UW Student Number: _____

I have neither given nor received assistance during this examination:

(Sign here): _____

I certify that I received all 8 pages of this test.

(Sign here): _____