1st Midterm Exam Key

Wednesday February 25
75 minutes == 75 points
Open book and notes

1. *15 points*
   You are given the missionaries and cannibals problem, which states that there are 3 missionaries and 3 cannibals on one side of a river, along with a boat that can hold one or two people. The problem is to find a way to get everyone to the other side of the river without ever leaving a group of missionaries in one place outnumbered by the number of cannibals in the same place.

   Multiple solutions will be accepted for this problem. Here is one.

   (a) *Describe how you would represent the state space, including the states, successor function, and goal test.*

   To describe a state, we'd want to know how many missionaries and cannibals are on either side of the river and which side of the river the boat is. It turns out, we can do this by only keeping 3 values per state (m, c, b): the number of missionaries on the other side ($m \in \{0, 1, 2, 3\}$), the number of cannibals of the other side ($c \in \{0, 1, 2, 3\}$), and the side the boat is on ($b \in \{start, other\}$).

   The initial state is $(0, 0, start)$, and the goal state is $(3, 3, other)$.

   The successor function returns feasible states formed adding (or subtracting) 1 to each m and c, 2 to either m or c, or adding 1 to either m or c and toggling b. Each state could have at most 8 possible successors (or children) since it is not possible to add 2 and to subtract 2 from the same number of missionaries or cannibals without going out of range. In all cases, the real number of successors will be smaller since some of these successors will be infeasible: leaving too many cannibals on one side of the river or constrained by the number of available passengers.

   (b) *Is the search space a tree or a graph? What is the branching factor?*

   The search space is most accurately characterized as a graph because some states can be reached on multiple paths. From the number of possible successors, we have our maximum branching factor (8). However, the actual number of children for any parent state will be much less since a number of these successors will be infeasible (either leaving too many cannibals on one side or constrained by the number of available missionaries or cannibals). If you solve the problem, you can see that the maximum branching factor is 3 from the initial state (move 2 cannibals, move 2 missionaries, move 1 cannibal and 1 missionary). Most of the time the branching factor is 1.

   (c) *What search algorithm would you use to solve the problem?*

   A lot of search algorithms would work here. As long as you successfully defended your choice, you got full points. Since the tree is small and all step costs are equivalent, we can use breadth-first search since it produces an optimal solution

2. *10 points*
Answer the following questions briefly but precisely:

   (a) *Can an agent that keeps no history of its percept sequence be rational? Please explain.*

   Yes. From the definition of a rational agent on page 36 of Russell and Norvig, we have:

   > For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

   By this definition, a rational agent only needs to select an action *expected* to maximize its performance measure. So, the quality of rationality depends on the performance measure and the agent choosing the action which maximizes that measure based on whatever help the agent has (history, knowledgebase, etc.) There is no requirement of keeping a history of its percept sequence.

   (b) *Why it is important to know if an agent's environment is fully observable or if it is partially observable?*

   It determines how we program the agent. If the agent's environment is only partially observable, the agent would likely need to keep internal state information. On the other hand, if the environment is fully observable, the agent would not need to keep track of the world internally. The agent would directly observe it.

3. *15 points*
Suppose you are given a CSP problem (for instance, a cryptoarithmetc problem) and you are interested in finding ALL the possible assignments of values to variables that satisfy the constraints.

   (a) *what algorithm would you use? Please explain.*

   We can use any systematic search algorithm as long as we do not terminate the search when a solution is found but only when there are no more nodes in OPEN. Backtracking search will work, as long as we change its exit condition. Backtracking search is a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.

   (b) *what representation (incremental formulation or complete-state formulation) do you think would work best? why?*

   Since we're using systematic search, we'd necessarily use an incremental formulation. Using complete-state would not make sense, because we are populating the assignments as we descend through the tree.

   (c) *would you use the MRV and degree heuristics? why? or why not?*

   There is no need to use any of the heuristics, since we have to explore the entire space, so it does not matter how fast we are in finding the first solution.

4. *25 points*
Answer the following questions explaining your reasoning briefly but precisely.

(a) *Suppose you have several admissible heuristic functions for a problem. Can you use them to produce a better heuristic? how?*

You could evaluate them all for each node and take the maximum as the true h-value for that node. Since all of the functions are admissible, the maximum will be admissible. Since you are taking the maximum, this heuristic will be more accurate to the actual cost from a node to the goal (i.e., By definition, it will dominate all the other heuristic functions).

(b) *If the heuristic function used in A* is monotonic, then the f-cost along any path expanded by A* is nondecreasing. Please explain why, specifically explaining why for this to be true the heuristics has to be monotonic.*

Here, we go back to the triangle inequality. If h is monotonic, then $h(n)$ is less than or equal to the cost of getting to any of n's successors plus the h-value of that successor. Since the g-cost never decreases and the h-value never decreases, we have that the f-cost never decreases. A short proof of this is in the textbook (bottom of page 99). With a non-monotonic heuristics the h values can change up and down along path, making the f-costs to fluctuate.

(c) *Is it true that when $h = 0$ A* has always to search the entire search space before finding the optimal solution?*

No. When $h = 0$, A* would act as uniform-cost search. Uniform-cost search produces an optimal solution, but it does not have to search through the entire space to find the solution.

(d) *What is the advantage of having independent sub-problems when solving a CSP?*

The total work for a CSP is linear in the number of variables when decomposed into independent subproblems. For a CSP without decomposition, the total work is exponential in teh number of variables. Independent subproblems also allow for parallelism in computation.

(e) *Why does IDA* require less memory than A*?*

IDA* uses a cutoff value for the f-cost rather than for depth as with other iterative deepening searches. At each iteration until the goal is found, we increase the cutoff value to the smallest f-cost of any node that exceeded the previous cutoff. With IDA*, we need not keep a relatively large sorted queue of nodes as we do with A*.

5. *10 points*
Write a function in Lisp that takes as an argument a list. The elements of the list are either symbols or lists of one element. The procedure should return a list where the symbols in a list are repeated twice, i.e. each sublist (symbol) is replaced by the list (symbol symbol).

You can use the system defined predicate listp to see if an object is a list, or atom to see if an object is not a cons. You should not use any global variable, but you are free to use as many auxiliary procedures as you like.

It should work like this:

```
(expand '(the (big) boat)) = (the (big big) boat)
```

```
(expand '((oh) wow))  = ((oh oh) wow)
(expand '(this is (very) nice)) = (this is (very very) nice)
```

Here are three solutions, the first uses mapcar, the others plain recursion. The last two solutions differ only in the way the double list is constructed. This is the part on which a number of students had difficulties.

```
(defun expand (lst)
  (mapcar #'(lambda (el)
              (if (atom el)
                  el
                 (list (car el) (car el))))
          lst))

(defun expand (lst)
  (cond ((null lst) nil)
        ((atom (car lst)) (cons (car lst) (expand (cdr lst))))
        (t (cons (list (caar lst) (caar lst))
                 (expand (cdr lst))))))

(defun expand (lst)
  (cond ((null lst) nil)
        ((atom (car lst)) (cons (car lst) (expand (cdr lst))))
        (t (cons (cons (caar lst) (car lst))
                 (expand (cdr lst))))))
```