Sample Midterm Exam

EECS 649, Prof. Michael S. Branicky

NAME: ______________________________

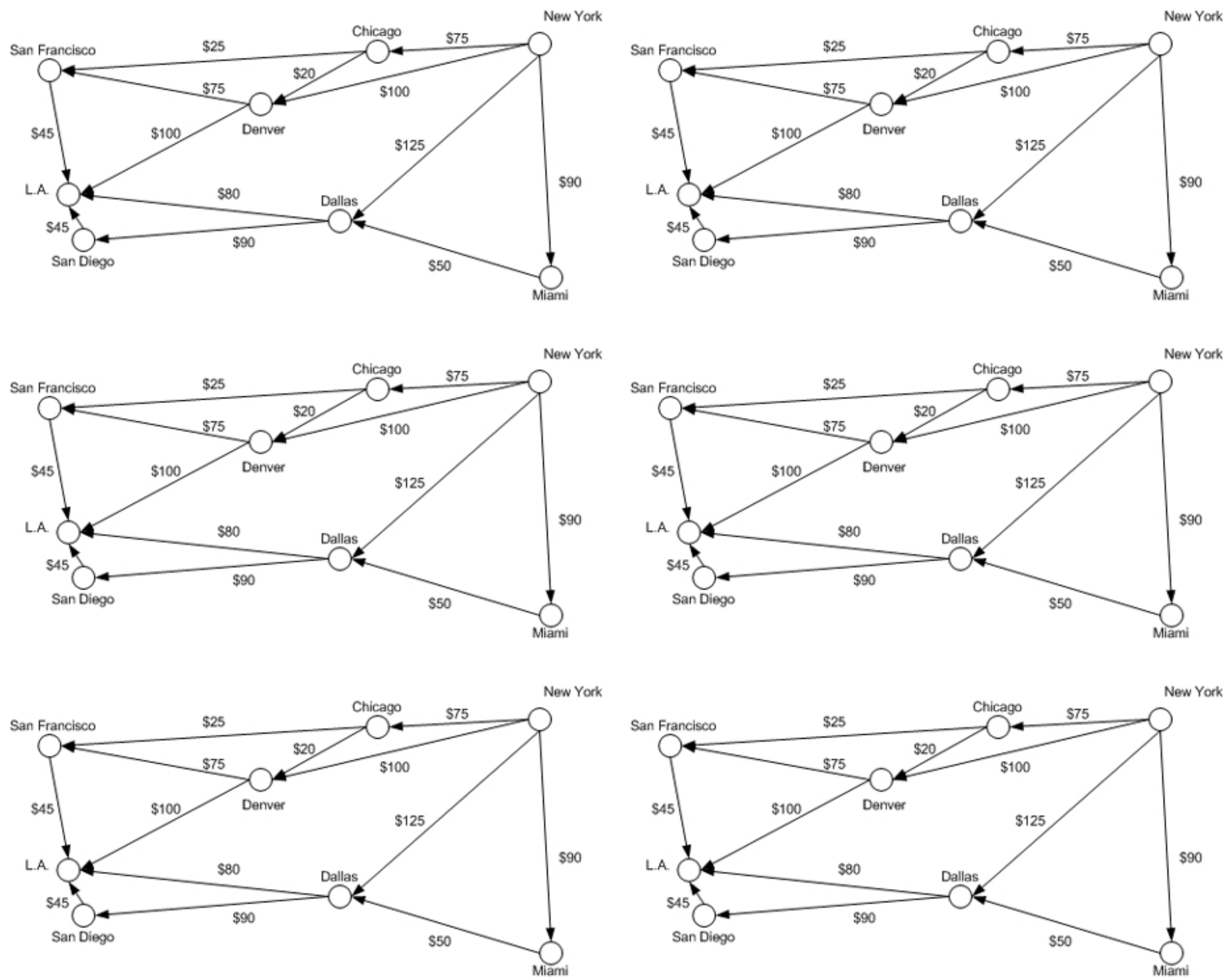
You must do **all five** problems.

General Notes:

- You may use books and notes, including programming books.
- No calculators or phones, indeed, **no electronic devices of any kind.**
- Feel free to perform other calculations on your own scratch paper. (Do not hand in.)
- **Read** each question carefully.
- Work quickly but accurately; come back to those you can't immediately perform.
- When done, **check** your work.

Problem	Score
1	
2	
3	
4	
5	
Total	

Problem M.1. (20 points) *Cheap Airfare Search.* Consider the repeated directed graphs below. Use them to show your partial work. The numbers on the arcs are the incremental airfare costs. The three California cities have a heuristic cost-to-go of \$0, the two East coast cities have a heuristic cost-to-go of \$90, Chicago has one of \$60, and the remaining two cities \$30.



Compute the order in which nodes would be **expanded** by each type of search below from New York to the goal of San Diego using the generic foward search algorithm, and then record the resulting path cost. Do not write down any queue/visited lists, only the sequence of states **expanded**. Assume children are generated in **north-to-south order**. Label any partial work above.

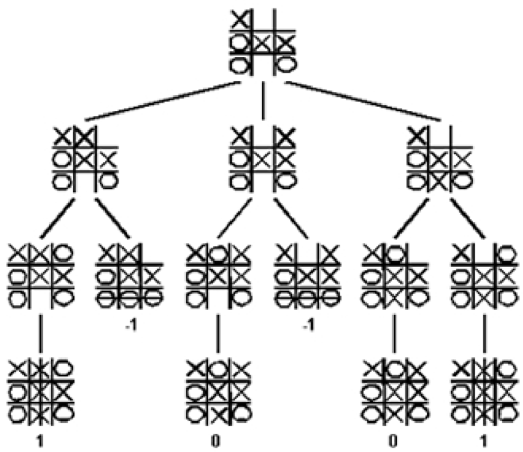
Search Type	List of nodes in order expanded	Path cost
Breadth-first		
Depth-first		
Uniform cost		
Greedy best-first		
A*		

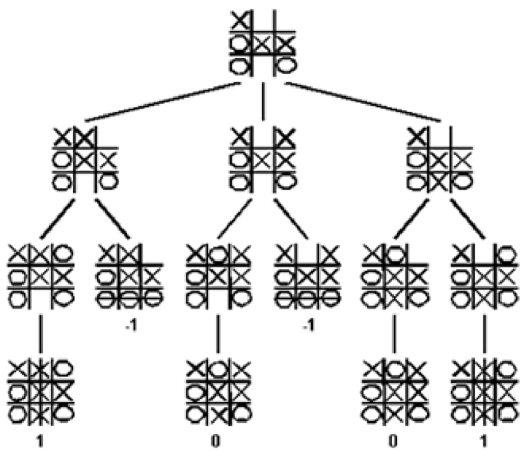
Problem M.2. (20 points) *Games.* Consider the repeated partial tic-tac-toe game tree below. There, it is $\times$'s turn to move, and terminal nodes are labeled with their costs from $\times$'s perspective.

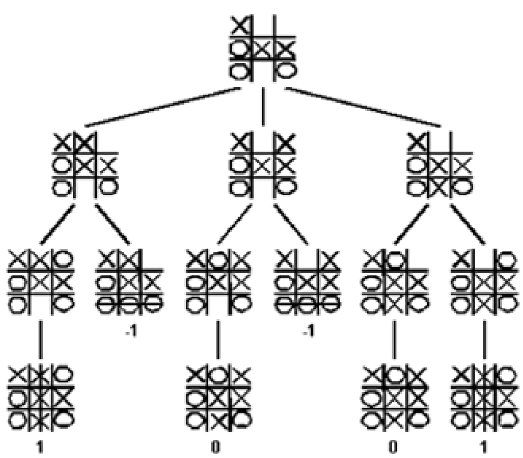
(a) [4 points] In the first tree below, fill in all eight internal minimax scores **and** clearly indicate what move $\times$ should make.

(b) [8 points] Perform a simulation of the alpha-beta algorithm on the second tree below, using $[\alpha, \beta]$ intervals or $\leq$, $\geq$ calculations. **Circle** all nodes that would **not** need to be examined using the alpha-beta algorithm when nodes are examined in **left-to-right** order.

(c) [8 points] Repeat (b) on the third tree below, but examining nodes in **right-to-left** order.



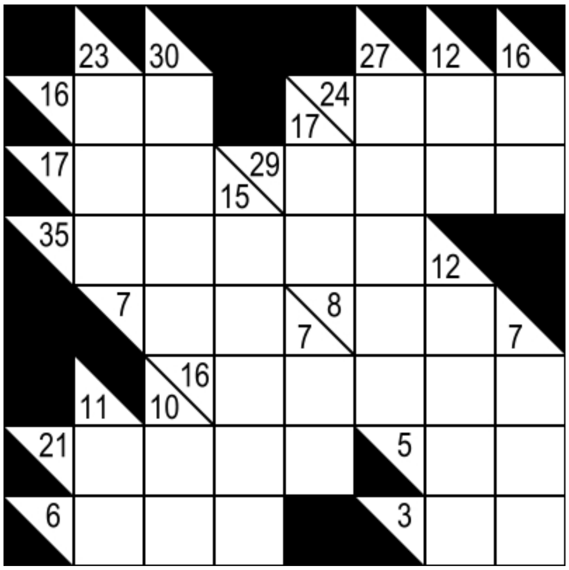


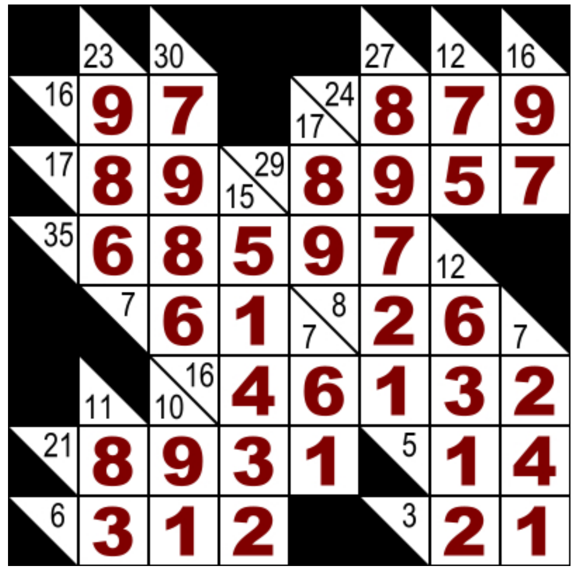


Problem M.3. (20 points) *Kakuro Constraint Satisfaction.* According to Wikipedia:

> Kakuro . . . is a kind of logic puzzle that is often referred to as a mathematical transliteration of the crossword. . . . The object of the puzzle is to insert a digit from 1 to 9 inclusive into each white cell such that *the sum of the numbers in each entry matches the clue associated with it* and that *no digit is duplicated in any entry.* [Italics mine.]

Below are a Kakuro puzzle (left) and its solution (right).





For example, given the clue constraints, forward checking (FC) on the bottom-most cell in the right-most column of the puzzle above would yield {1, 2} as the set of remaining digits.

(a) [8 points] Given the clue constraints, perform FC for the two top-most cells and the bottom-most cell in the left-most column of the puzzle. Enter your answers as *small but legible* numbers within each cell.

(b) [6 points] Given the clue constraints **and** those answers, use FC on the remaining two empty cells in the left-most column. Again, enter your answers within each cell.

(c) [6 points] **Briefly** describe how you'd use CSP methods to solve Kakuro puzzles intelligently.

Problem M.4. (20 points) *Unicorn Logic* [cf. R&N]. Consider the knowledge base (KB) below:

$$Y \Rightarrow \neg R, \qquad \neg Y \Rightarrow R \wedge M, \qquad \neg R \vee M \Rightarrow H$$

(a) [8 points] Convert the statements above into a KB of **CNF clauses** and place the results into the five empty rectangular boxes below. Use this space for intermediate work:

(b) [6 points] $KB \models H$, that is, H is a logical consequence of the KB. You will prove this using resolution on the KB plus $\neg H$ to derive the empty clause. Start with the clauses in the boxes below and proceed by resolving to produce new clauses (which you should place in new boxes, connected by arrows coming from the two source clause boxes—like in Figure 7.13, p. 216 of R&N) until the empty clause is derived. You need **not** produce all allowed clauses!

$\neg H$					

(c) [6 points] Y is not a logical consequence of the KB. You will prove this by finding a model in which Y is false. Specifically, you will use DPLL to satisfy the CNF clauses from part (a) **plus** the clause $\neg Y$. **Briefly justify** your reasoning through DPLL and **record** your ultimate truth assignments for each variable in the spaces below (use T for true, F for false).

$$Y = \qquad , \qquad R = \qquad , \qquad M = \qquad , \qquad H =$$

Problem M.5. (20 points) *Misc. Search.* The following is from an AI book by Alison Cawsey:[1]

In simple planning systems the problem state can be represented as a list of facts that are true, e.g.:

[at(robot, living_room), at(beer, kitchen), at(fred, living_room), door_closed(kitchen, living_room)] — door_closed(living_room, kitchen)

This might represent a state where Fred is in the living room with his robot, but the beer is in the kitchen and the door to the kitchen is shut.

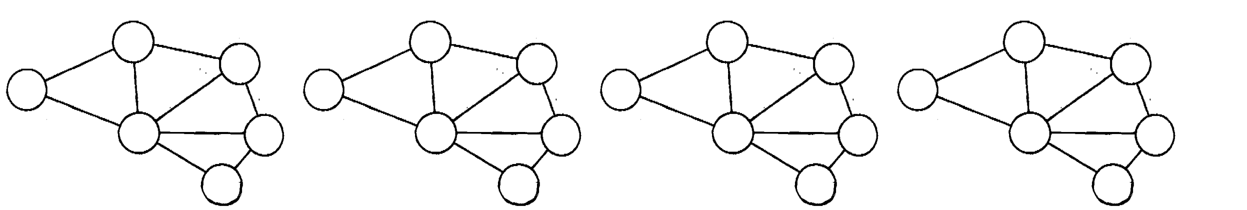
... if we assume that the only actions allowed in our example are "robot opens/closes door", "robot moves from one room to another" and "robot carries object from one room to another" then we can have the following operators to describe the possible actions:

Operator	Preconditions	Add	Delete
open(R1, R2)	at(robot, R1) door_closed(R1, R2) door_closed(R2, R1)	door_open(R1, R2) door_open(R2, R1)	door_closed(R1, R2) door_closed(R2, R1)
close(R1, R2)	at(robot, R1) door_~~closed~~ open(R1, R2) door_open(R2, R1)	door_closed(R1, R2) " (R2, R1)	door_open(R1, R2) " (R2, R1)
move(R1, R2)	at(robot, R1) door_open(R1, R2)	at(robot, R2)	at(robot, R1)
carry(R1, R2, O)	door_open(R1, R2) at(robot, R1) at(O, R1)	at(robot, R2) at(O, R2)	at(robot, R1) at(O, R1)

Suppose our target state involves Fred with his beer in the living room, his robot by his side, the door closed.

(b) [8 points] Find a plan from the initial state (at the top of the excerpt) to the target state.

(c) [6 points] Use the maps below to solve a "coloring problem" over three colors—R, G, and B—where no two adjacent/connected nodes/regions should be the same color. Start with an initial configuration where every node is labeled R. Then, use hill-climbing/min-conflicts to move successively toward a solution. Break ties by preferring **north-and-west** nodes.



[1] I have corrected a few bugs from the book: The second predicate in the Preconditions List for `close` should be `door open`. The Preconditions and the Delete List for `open` and the Add List for `close` should include `door_closed(R2,R1)`. The Preconditions and the Delete List for `close` and the Add List for `open` should include `door_open(R2,R1)`.