Northern Michigan University (Marquette Co, MI)

CS470-01-26W: Artificial Intelligence (Andrew A. Poe) Name: _____
Practice Midterm Examination (Exam 1) Wednesday 11 March 2026 9:00 A.M. EDT

Time: 50 minutes

1. I'm playing a game of checkers, and I'm cheating. I have an iPhone in my lap, and it's using the minimax algorithm to calculate my guaranteed next move.

I'm joking, of course. An iPhone has nowhere near the processing power or speed to use minimax on checkers. But suppose it did. Why is minimax STILL a bad idea to use to compute the best move in checkers?

Depending on how rigidly you define the game of checkers, a game of checkers isn't guaranteed to terminate. Checkers terminates when a player has no legal move on their turn (which causes that player to lose). But you can have games that could go on forever when both sides have kings. Minimax isn't equipped to handle that. (Of course, you could define checkers to end in a draw the same pattern shows up multiple times. It's still tricky to test for.)

2. The jumping peg puzzle consists of *n* black pegs and *n* white pegs on a linear board with 2*n*+1 holes. The starting position is BBBBBB<empty>WWWWWW (for however many pegs you have). The ending position is WWWWWW<empty>BBBBBB. The goal is to find the set of moves that will lead from the start state to the end state in the smallest possible number of moves. Black always moves right. White always moves left. A peg can either move one space to a hole OR it jump over one peg of the opposite color into a hole. A* can be used to find the correct solution to the game. But how would you interpret the game in the light of A*? What are the nodes? What is the distance between the nodes? What is the heuristic to estimate the distance to the final position?

You can imagine a node to be any permutation of n B's, n W's and one <empty>. Adjacent nodes are nodes for which there is a valid move from one to the other. The distance between adjacent nodes is 1.

Remembering that A*'s heuristic is not allowed to be an overestimate, one way to do this is count far each peg is from its final position, add them up, and divide by 2 (since a peg can move two positions on one move).

3. In the quiz you just took, you used a genetic algorithm to give a good solution to the Traveling Salesman problem. How would you use simulated annealing to approach that problem? How would you evaluate each position? How would you move to the next position? How would you know whether to keep or retract a move?

A position would be a specific ordering of the cities.

Evaluating that position would just be to compute the length of the traveling salesman tour from that specific position.

Moving to the next position would involve swapping two random cities in the tour. If that gives a better path, GREAT, take it. Otherwise, simulated annealing use a probability (which gets closer to zero as time goes on) to accept a weaker path, anyway.

4. In a genetic algorithm, if your initial population consists of identical strings, and the child of two identical strings is another copy of that string (as is often the case), how can the genetic algorithm still do reasonably well, even in that starting situation? (In class, we started with identical strings in the load balancing problem, and it still did OK.)

Even if the child is the clone of its identical parents, there is still a chance of mutation. The mutated child will then make new and different children with the population. Eventually, you'll have a decent population.