CS360 – Homework #8

Genetic Algorithms

1) What are the advantages and disadvantages of carrying over the fittest two individuals to the next generation?

Carrying over the fittest individuals (also known as 'elitism') tilts the genetic algorithm towards local search, because the best individuals stay around to (mostly) create more and more children very near to themselves in the genotype space. This could make the genetic algorithm more efficient, but also can cause it to more easily get caught in local optima.

2) What are the advantages and disadvantages of running genetic algorithms with only mutations and no crossovers? How about only crossovers and no mutations?

Genetic algorithms without mutations cannot explore outside the possible points in space formed by choosing any combination of parameter settings in its original population. As individuals are weeded out, this combination reduces, ultimately to a single individual. For instance, if the individuals are represented as bit strings and the first bit of every individual is 0 in the first generation or becomes 0 (if all the 1's are eliminated), then the algorithm cannot generate solutions whose first bits are 1.

Genetic algorithms without crossovers perform purely random searches whose children are very similar to their parents. Crossovers help rapidly transfer high-performing parameters throughout the population, making genetic algorithms more efficient in problems where the parameters are at least somewhat independent of one another.

So, in general, mutations keep the search (somewhat) global and crossovers make it more efficient.

3) How would you encode a state if you were using a genetic algorithm to solve the Traveling Salesman Problem but only wanted to use a straightforward crossover operation that switches prefixes of both parents' encodings (that is, we randomly pick a cutoff point in the encoding, use the encoding of the first parent up to that cutoff point, and use the encoding of the second parent after that cutoff point)?

We can, for example, assign an integer to each city, encoding its priority. These priorities correspond to a unique tour, where the salesman visits the cities in the order of their priorities (breaking ties lexicographically). For this encoding, crossovers and mutations always create tours and thus valid solutions. Compare this encoding to one where, for each vertex, we keep the next vertex to visit. The crossover operator would have to be defined very carefully in order to generate valid solutions.

SAT-based Planning

4) In class, we have seen how to formulate a planning problem as a SAT (= satisfiablity) problem. The formulation allows the SAT solver to find plans that can execute actions in parallel, as long as all ways of sequentializing these parallel actions result in valid plans, that is, none of the actions deletes a precondition of another one. How would you modify the formulation a) if we want to find plans that cannot execute actions in parallel? b) if we want to find plans that execute actions in parallel as long as there exists at least one way of sequentilalizing theses parallel actions that results in a valid plan?

In both cases, we change the definition of conflicting actions and then ensure that conflicting actions cannot be executed in the same timestep.

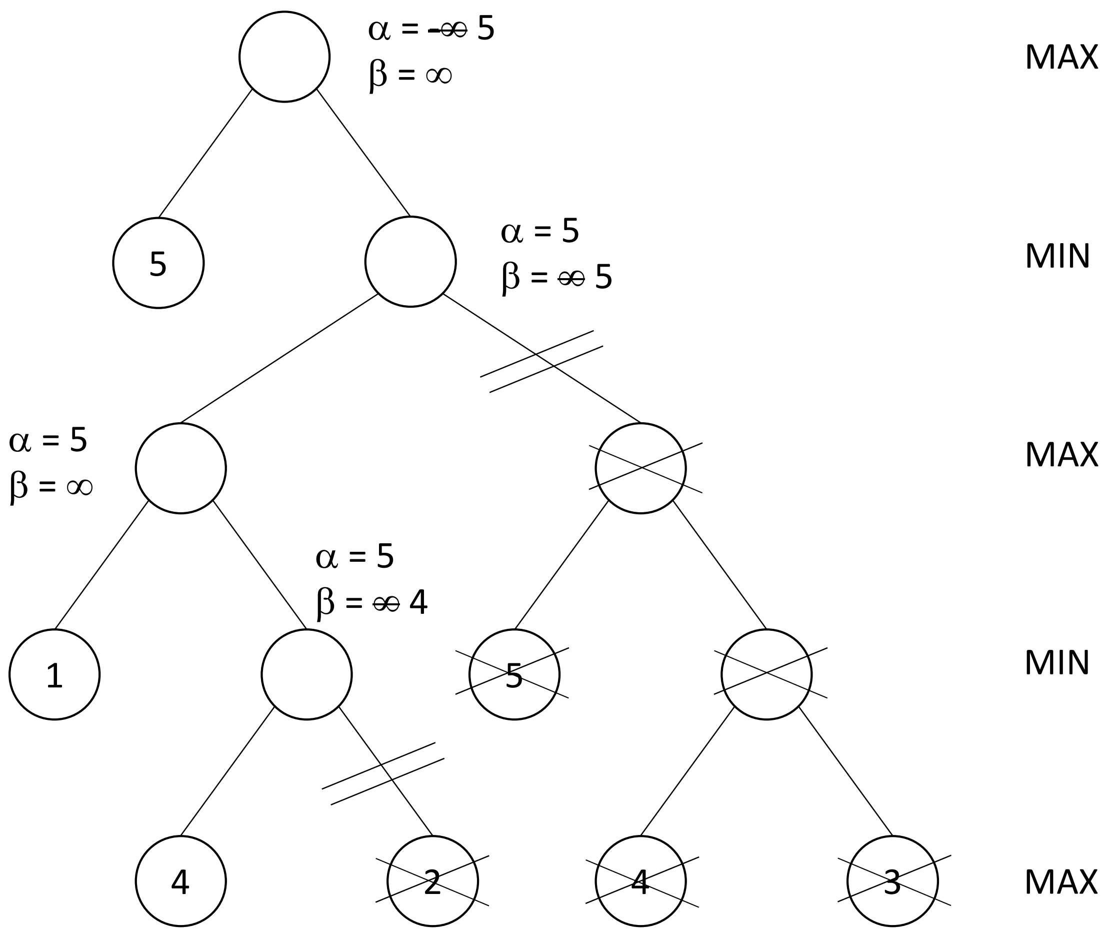
a) Any two actions conflict during each time step. That is, let X_t denote the proposition that action X is executed at timestep t. For each pair of actions X and Y and for each time step t, we add the clause $\neg X_t \vee \neg Y_t$.

b) Several actions conflict during each time step iff (= if and only if) there does not exist an ordering of them so that earlier actions do not delete the preconditions of later ones. Let $X_{i,t}$ denote the proposition that action X_i is executed at time step t. For each set $\{X_1, \ldots, X_n\}$ of conflicting actions and for each timestep t, we add the clause $\neg X_{1,t} \vee \cdots \vee \neg X_{n,t}$.

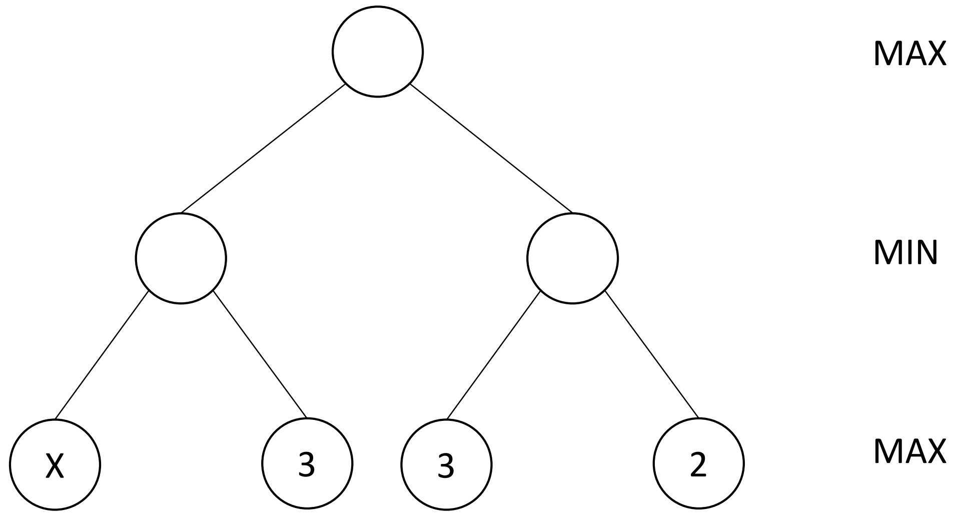
Game Tree Search

5) What is the minimax value of the root node for the game tree below? Cross out the node(s) whose value(s) the alpha-beta method never determines, assuming that it performs a depth-first search that always generates the leftmost child node first and a loss (and win) of MAX (and MIN) corresponds to a value of $-\infty$ (and ∞, respectively). Determine the alpha and beta values of the remaining node(s).

The minimax value is 5.



6) Assume that you are given a version of the alpha-beta method that is able to take advantage of the information that all node values are integers that are at least 1 and at most 6. Determine ALL values for X that require the algorithm to determine the values of ALL nodes of the following game tree, assuming that the alpha-beta method performs a depth-first search that always generates the leftmost child node first.



(Remember to initialize $\alpha = 1$ and $\beta = 6$ for the root node of the minimax tree.) Let a be the sibling of the node with value X, and b be the node with value 2. If we assign X = 1, then only node a can be pruned. If we assign X = 2, then all nodes must be expanded. For higher values of X, node b can be pruned. Therefore, the answer is 2.

7) The minimax algorithm returns the best move for MAX under the assumption that MIN plays optimally. What happens if MIN plays suboptimally? Is it still a good idea to use the minimax algorithm?

The outcome of MAX can only be the same or better if MIN plays suboptimally compared to MIN playing optimally. So, in general, it seems like a good idea to use minimax. However, suppose MAX assumes MIN plays optimally and minimax determines that MIN will win. In such cases, all moves are losing and are "equally good," including those that lose immediately. A better algorithm would make moves for which it is more difficult for MIN to find the winning line.