CS 540-3: Homework 2

Local Search and Game Playing

40 pts total

Release date: February 19, 2016
Due date: February 26, 2016 at 11:59PM

Directions: Please submit *one* PDF file to Moodle by the deadline specified. Failure to convert your solutions to a single PDF file will result in the deduction of points. Please be sure you understand the course policies on plagiarism before consulting with other students. See either the TA or instructor if you have any questions.

1 Genetic Algorithms [10 pts]

Recall that genetic algorithms represent a type of local search. Genetic algorithms have seen application in electronic chip design, a scheme known as evolvable hardware. (Read a tutorial here: http://folk.uio.no/jimtoer/fpl04_Torresen.pdf.) Under this scheme, chip layouts can be represented as a series of bits that correspond to placement of the components (e.g., transistors) on this chip. Suppose we have a population of such chip arrangements, as shown in Table 1. Further suppose our parent selection chooses chips 4 and 6, chips 1 and 3, and then chips 5 and 6, respectively, for three mating events. Assume we know that crossover events happen after bits 2, 5, and 8 for the first, second, and third mating events, respectively.

Chip Number	Bit String Representation
1	0100010011
2	1011000010
3	1101110010
4	0011110010
5	1000001101
6	0100110011

Table 1: **Population of Chips Represented as Bit Strings**

a) Follow the genetic algorithm as it's laid out on the lecture slides. After the three mating events take place, what will the new population be? (Don't worry about the fitness of the children.)

b) In addition to crossover events, mutations are often also used in genetic algorithms. What are mutation events? Why are they used?

c) Why are genetic algorithms less susceptible to local optima than a standard hillclimbing algorithm?

a) (6 points) New population is presented below. Dashes represent the crossover points used to produce the children.

ID	Bit representation
1a	00-00110011
1b	01-11110010
2a	01000-10010
2b	11011-10011
3a	10000011-11
3b	01001100-01

b) (2 points) Mutations are random flips of random bits in the bit representation of new children. Mutations are used to introduce even more randomness into the process of children generation in hopes that we will accidentally discover a very fit child or escape a local optimum.

c) (2 points) GAs are less susceptible to local optima because they let us jump far away from our current state thanks to the cross-over procedure and mutations. We hope that such long random jumps will be able to get us out of local optimas.

2 Simulated Annealing [15 pts]

Consider the function $f(x) = max\{4 - |x|, 2 - |x - 6|, 2 - |x + 6|\}$. It has three peaks with one forming a unique global maximum. Perform 8 rounds of simulated annealing using 4 as your start point, and where the temperature can be calculated from the function $T(i) = 2(0.9)^i$, where i is the iteration number. (E.g., for the first iteration, the temperature will be $T(1) = 2(0.9)^1 = 1.8$.) Show all of your work, including the current point, the round temperature, and the probability of changing position given the successor and the temperature.

Note: Recall that a random successor is generated on each iteration of the simulated annealing. If this successor is better than the current state, it is always accepted. If it is not better, it will still be accepted with some probability. Use Table 2 to see which successor was "randomly" chosen at each step, along with the "randomly" generated number which will help you to decide if you should take a non-optimal successor at that step.

Iteration Number	Random Successor	Random Number Generated
1	3	0.102
2	1	0.223
3	1	0.504
4	4	0.493
5	2	0.312
6	3	0.508
7	4	0.982
8	3	0.887

Table 2: **Successor States and Random Numbers**

1. $x_{curr} = 4, x_{next} = 3, f(x_{curr}) = 0, f(x_{next}) = 1$.
   Since $f(x_{curr}) < f(x_{next})$, we accept x_{next} with probability 100%.

2. $x_{curr} = 3, x_{next} = 1, f(x_{curr}) = 1, f(x_{next}) = 3$.
   Since $f(x_{curr}) < f(x_{next})$, we accept x_{next} with probability 100%.

3. $x_{curr} = 1, x_{next} = 1, f(x_{curr}) = 3, f(x_{next}) = 3$.
   Since $f(x_{curr}) \not< f(x_{next})$, we accept x_{next} with probability $e^0 = 1 = 100\%$.
   So we will accept the new state.

4. $x_{curr} = 1, x_{next} = 4, f(x_{curr}) = 3, f(x_{next}) = 0$.
   Since $f(x_{curr}) \not< f(x_{next})$, we accept x_{next} with probability $p = e^{-3/1.312} = 0.102$. The generated random number is $r = 0.493$.
   Since $r > p$, we do not accept the new state.

5. $x_{curr} = 1, x_{next} = 2, f(x_{curr}) = 3, f(x_{next}) = 2$.
   Since $f(x_{curr}) \not< f(x_{next})$, we accept x_{next} with probability $p = e^{-1/1.181} = 0.429$. The generated random number is $r = 0.312$.
   Since $r < p$, we do accept the new state.

6. $x_{curr} = 2, x_{next} = 3, f(x_{curr}) = 2, f(x_{next}) = 1$.
   Since $f(x_{curr}) \not< f(x_{next})$, we accept x_{next} with probability $p = e^{-1/1.106} = 0.39$. The generated random number is $r = 0.508$.
   Since $r > p$, we do not accept the new state.

7. $x_{curr} = 2, x_{next} = 4, f(x_{curr}) = 2, f(x_{next}) = 0$.
   Since $f(x_{curr}) \not< f(x_{next})$, we accept x_{next} with probability $p = e^{-2/0.95} = 0.12$. The generated random number is $r = 0.982$.
   Since $r > p$, we do not accept the new state.

8. $x_{curr} = 2, x_{next} = 3, f(x_{curr}) = 2, f(x_{next}) = 1$.
   Since $f(x_{curr}) \not< f(x_{next})$, we accept x_{next} with probability $p = e^{-1/0.86} = 0.313$. The generated random number is $r = 0.887$.
   Since $r > p$, we do not accept the new state.

   Final state at the end of 8th iteration: $x_{curr} = 2$.

3 Game Playing [15 pts]

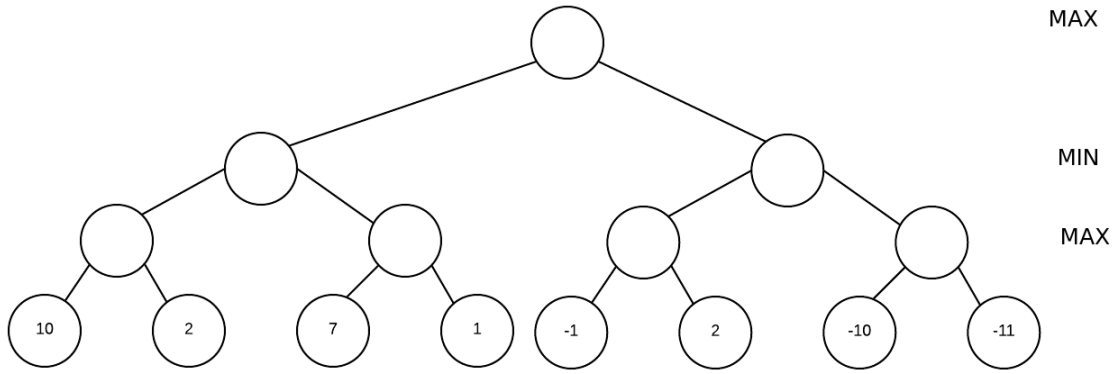

Figure 1: **Question 3 Graph**

a) Refer to Figure 1. Use the Minimax algorithm to compute the minimax value at each node for the tree shown.

b) Refer once more to Figure 1. Use Alpha-Beta Pruning to compute the minimax value at each node for the game tree shown, assuming children are visited left to right. Show the alpha and beta values at each node. Show which branches are pruned, if any.

c) Why do we use Alpha-Beta Pruning?

a) (5 points) Completed solution is shown below in Fig 2.

b) (8 points) Completed solution is shown below in Fig 3.

c) (2 points) Alpha-beta pruning is used to avoid unnecessarily searching redundant branches. This allows for deeper searching and, consequently, a more competitive computer opponent.

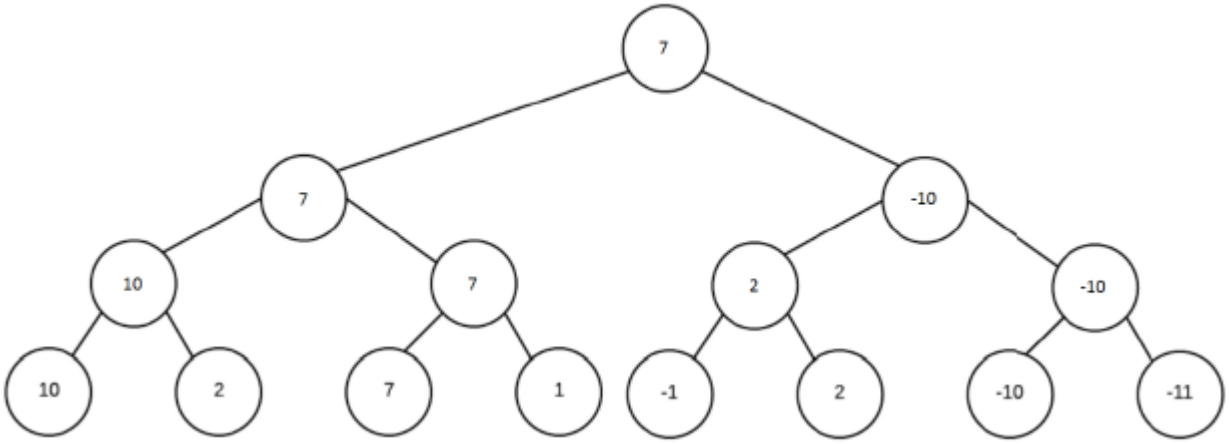

Figure 2: **Question 3a Solution**

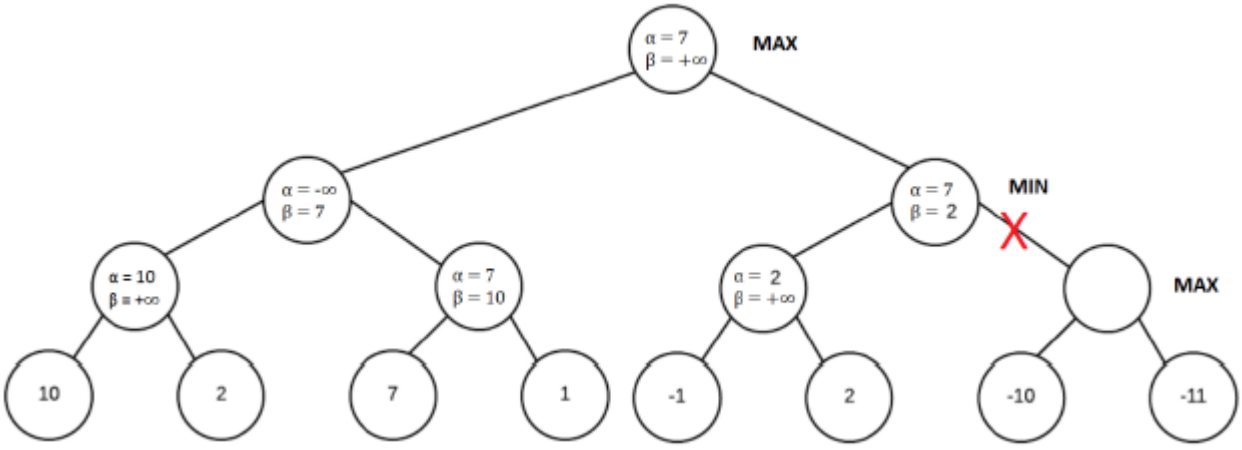

Figure 3: **Question 3b Solution**