Artificial Intelligence: Representation and Problem Solving
15-381, Fall 2009

Homework 2 SOLUTIONS

DUE: Wednesday, September 30th, 2009, 3:00 P.M.

REMINDER: Due at the beginning of the class on the day it is due.

[**Maximum points: 100**]

Handin instructions

- Put the following in `/afs/andrew/course/15/381-f09/handin/`*your andrew id*`/hw2/`
  - The output of your code in the 4 files `output1.txt` - `output4.txt`
  - The andrew ID's of your team members in `team.txt`
- Bring the typeset or handwritten answers to the rest of the questions to class. Answers *MUST* be legible for credit.
- Late homework is due by 3:00 P.M. on the day they are due. Please bring late homeworks to GHC 7027 (slide it under the door if Heather Carney is not there).

Guidelines

- You can work in a group of two people. This group does not need to be the same group as for the other homeworks. You only need to turn in one writeup per group, but you need to include the andrew ID's of the group members. You only need to turn in one program output per group.
- If you have any questions about this assignment, contact the instructors at 15381-instructors@cs.cmu.edu

Problem 1: More Search (30 points)

Answer these questions in one or two concise paragraphs.

1. In lecture, we presented a *hill climbing* algorithm for a discrete state space. Discuss how you might extend *hill climbing* for a continuous state space.
   Two popular solutions are discretizing the state space and gradiant ascent. Popular ways to discretize the state space are sampling or making a grid. Then you can run the algorithm as you would with the discrete state space hill climbing. Gradiant ascent works if you can calculate the gradiant (slope) at your current position. If you can, all you have to do is just go up the hill.

2. Consider searching for an answer to the N-Queens problem using *hill climbing*. Let us define the states as a configuration of all the queens (possibly conflicting) and let us define the neighbors as the states you can reach by moving one queen. What is the eval function for this problem? In this case what do plateaux and local maxima represent in terms of the eval function?
   The eval function can be the number of pairs of queens in conflict. A state is a platau if all its neighbors have the same eval value. A state is a local maximum if all its neighbors have a lower eval value.

3. *Simulated annealing* will find a global optimum with probability approaching 1 if you lower T slowly enough. Please explain what *simulated annealing* will do in each of the following cases

   (a) What happens if you lower T very quickly?
   You will very likely get stuck in a local maximum

   (b) What happens if you never lower T?
   You will always have a nonzero probability of going downhill. Therefore, even if you reach the global maximum, you won't stay there

4. Explain other ways that *hill climbing* can add randomness to avoid getting stuck. (By other we mean other than *simulated annealing*)
   One popular method is random restarts. When your algorithm reaches a maximum, randomly pick a new state and start hill climbing from there. Do this N times. When you are done, choose the maximum with the best value as your answer.

5. *Genetic algorithms* are difficult to use well. Please explain three reasons why it is difficult to solve a problem with a *genetic algorithm*.
   A possible answer would be:

   (a) It is very hard to encode a problem as a binary string

   (b) It is difficult to define a fitness function to decide who reproduces and who dies

   (c) There are a lot of parameters to tweek, and hard to get them right.

6. Here are three populations from a genetic algorithm:

   (a) 01010000 10000001

   (b) 10010000 01000001

   (c) 01010000 10100001

   Population b was created by population a reproducing (the crossover is after the second digit). Population c was created by population a mutating (the second string was mutated). Let the fitness function be the number of 1's in a string. What is the maximum fitness of each population? Is there a better crossover point than the one used for creating population b? If so, where is it and why is it better?
   The maximum fitness of a string is 2 for population *a*, 2 for population *b*, and 3 for population *c*. Better crossover points would be after the first digit, after the fourth digit, after the fifth digit, after the sixth digit, or after the seventh digit. These are better crossover points as the child generation, *b* would have a maximum fitness of 3, rather than 2.

Problem 2: Multiplayer Minimax and Alpha Beta Pruning (35 points)

Abe, Mary and Sue Ann are playing each other in a game of Calvinball. (Calvinball is a game for which all the rules are made up. You do not need to know anything about Calvinball for this problem.) Today, the rules dictate that at the end of the game each player gets some points, and the winner is the player with the most points. Each TA would rather win than lose. If a TA is going to win, he or she would like to maximize his or her points. If a TA is going to lose, he or she would also like to maximize his or her points.

2.1: Defining an evaluation function (5 Points)

Define a utility function UTILITY(P_A,P_M,P_{SA}) that outputs the vector (U_A,U_M,U_{SA}), where P_A,P_M,P_{SA} are Abe, Mary, and Sue Ann's respective points and U_A,U_M,U_{SA} are Abe, Mary, and Sue Ann's respective utilities.

This utility function would satisfy the requirements:

- **If $\mathbf{P}_A \geq \mathbf{P}_M$, $\mathbf{P}_{SA}$ then UTILITY($\mathbf{P}_A$,$\mathbf{P}_M$,$\mathbf{P}_{SA}$) = ($\mathbf{P}_A$, -$\mathbf{e}^{-P_M}$, -$\mathbf{e}^{-P_{SA}}$)**
- **If $\mathbf{P}_M > \mathbf{P}_A$, $\mathbf{P}_{SA}$ then UTILITY($\mathbf{P}_A$,$\mathbf{P}_M$,$\mathbf{P}_{SA}$) = (-$\mathbf{e}^{-P_A}$, $\mathbf{P}_M$, -$\mathbf{e}^{-P_{SA}}$)**
- **If $\mathbf{P}_{SA} > \mathbf{P}_A$, $\mathbf{P}_M$ then UTILITY($\mathbf{P}_A$,$\mathbf{P}_M$,$\mathbf{P}_{SA}$) = (-$\mathbf{e}^{-P_A}$, -$\mathbf{e}^{-P_M}$, $\mathbf{P}_{SA}$)**

2.2: Run Minimax (15 Points)

In reality, if a TA is going to lose, he or she will try to minimize the winner's score. Let us define a new utility function to represent this.

- If $P_A \geq P_M$, P_{SA} then UTILITY'(P_A,P_M,P_{SA}) = (P_A, -P_A, -P_A)
- If $P_M > P_A$, P_{SA} then UTILITY'(P_A,P_M,P_{SA}) = (-P_M, P_M, -P_M)
- If $P_{SA} > P_A$, P_M then UTILITY'(P_A,P_M,P_{SA}) = (-P_{SA}, -P_{SA}, P_{SA})

Apply the Minimax algorithm using this utility function and generate a vector of length 3 for each node of the game tree.

You can do your work on the provided worksheet and just fill in the blanks. The numbers below each leaf are (P_A,P_M,P_{SA}).

Solution: see worksheet

2.3: Alpha Beta Pruning (15 Points)

Abe and Sue Ann have decided to team up, to get revenge on Mary. Now Abe and Sue Ann have the goal to get one of them to win with the most points possible. Modify the utility function from 2.2 and run Minimax again, but this time with Alpha Beta Pruning. Draw a line cutting an edge to indicate a branch being pruned. If more pruning would occur with another ordering of the leaves, provide that ordering (for example, if going down the right branch initially would allow you to look at fewer nodes).

You can do your work on the provided worksheet.
Solution: see worksheet.
The new utility function is as follows:

- **If $\mathbf{P}_A \geq \mathbf{P}_M, \mathbf{P}_{SA}$ then UTILITY"($\mathbf{P}_A$,$\mathbf{P}_M$,$\mathbf{P}_{SA}$) = $\mathbf{P}_A$**
- **If $\mathbf{P}_M > \mathbf{P}_A, \mathbf{P}_{SA}$ then UTILITY"($\mathbf{P}_A$,$\mathbf{P}_M$,$\mathbf{P}_{SA}$) = -$\mathbf{P}_M$**
- **If $\mathbf{P}_{SA} > \mathbf{P}_A, \mathbf{P}_M$ then UTILITY"($\mathbf{P}_A$,$\mathbf{P}_M$,$\mathbf{P}_{SA}$) = $\mathbf{P}_{SA}$**

This way Abe/Sue Ann are the max player and Mary is the min player.

A better ordering of leaves would be as follows: If the first 4 nodes were <4,4,5>,<0,3,0>,<7,3,5>,<6,2,3>, then you would be able to prune 3 more nodes (one more cut). This is basically just switching the two left subtrees.

Figure 1: Worksheet solution for 2.2

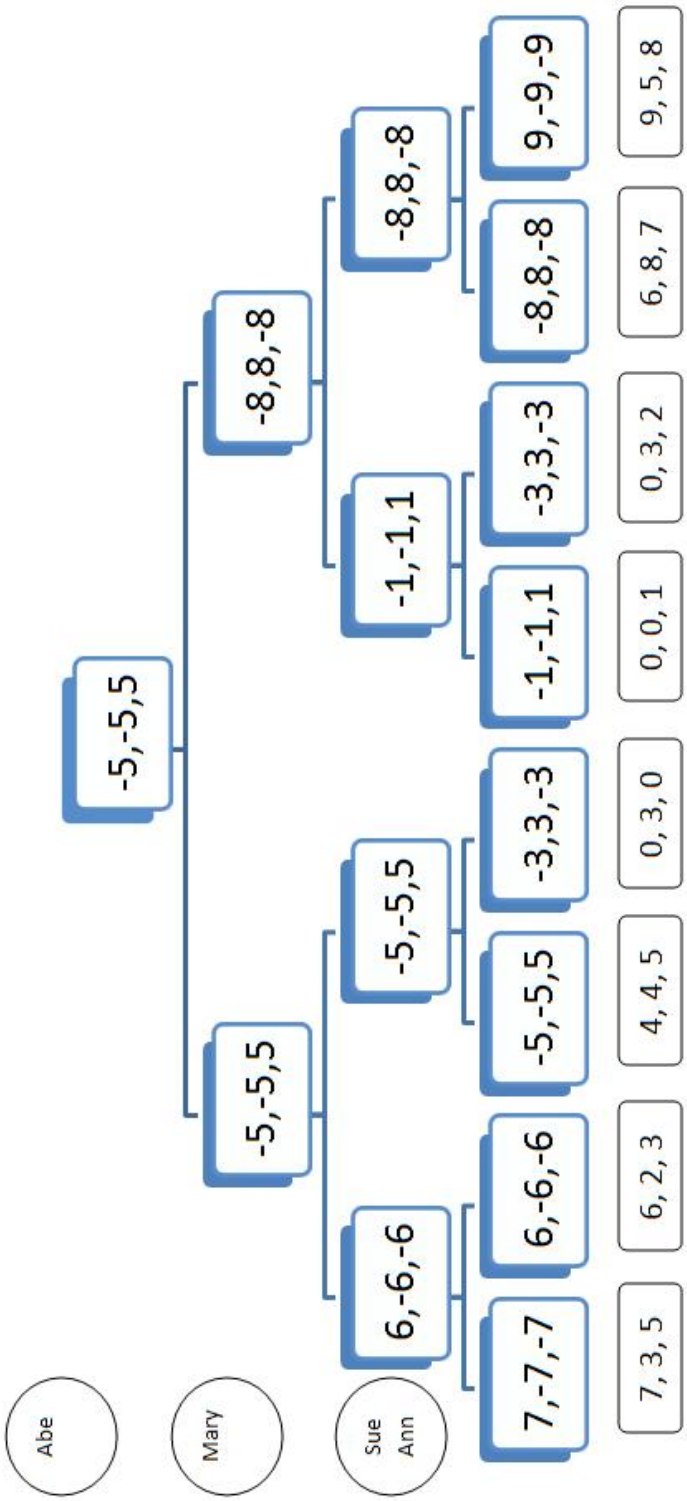

Figure 2: Worksheet solution for 2.3