Homework 2

Due 9/27/2012 2 PM

Each question is worth 5 points.

- **Question 3.3**
  - a. State space: States are all possible city pairs (i, j). The map is not the state space.
    Successor function: The successors of (i, j) are all pairs (x, y) such that $Adjacent(x, i)$ and Adjacent(y, j).
    Goal: Be at (i, i) for some i.
    Step cost function: The cost to go from (i, j) to (x, y) is max(d(i, x), d(j, y)).
  - b. In the best case, the friends head straight for each other in steps of equal size, reducing their separation by twice the time cost on each step. Hence (iii) is admissible.
  - c. Yes: e.g., a map with two nodes connected by one link. The two friends will swap places forever. The same will happen on any chain if they start an odd number of steps apart. (One can see this best on the graph that represents the state space, which has two disjoint sets of nodes.) The same even holds for a grid of any size or shape, because every move changes the Manhattan distance between the two friends by 0 or 2.
  - d. Yes: take any of the unsolvable maps from part (c) and add a self-loop to any one of the nodes. If the friends start an odd number of steps apart, a move in which one of the friends takes the self-loop changes the distance by 1, rendering the problem solvable. If the self-loop is not taken, the argument from (c) applies and no solution is possible.
- **Question 3.20** Clearly, graph search must be usedthis is a classic grid world with many alternate paths to each state. Students will quickly find that computing the optimal solution sequence is prohibitively expensive for moderately large worlds, because the state space for an $n \times n$ world has $n^2 \cdot 2^n$ states. The completion time of the random agent grows less than exponentially in n, so for any reasonable exchange rate between search cost ad path cost the random agent will eventually win.
- **Question 3.21**
  - a. When all step costs are equal,$g(n) \propto depth(n)$, so uniform-cost search reproduces breadth-first search.
  - b. Breadth-first search is best-first search with $f(n) = depth(n)$; depth-first search is best-first search with $f(n) = -depth(n)$; uniform-cost search is best-first search with $f(n) = g(n)$.
  - c. Uniform-cost search is A^* search with $h(n) = 0$.
- **Question 5.7** Consider a MIN node whose children are terminal nodes. If MIN plays suboptimally, then the value of the node is greater than or equal to the value it would have if MIN played optimally. Hence, the value of the MAX node that is the MIN nodes parent can only be increased. This argument can be extended by a simple induction all the way to the root. If the suboptimal play by MIN is predictable, then one can do better than a minimax strategy. For example, if MIN always falls for a certain kind of trap and loses, then setting the trap guarantees a win even if there is actually a devastating response for MIN.
- **Question 5.12** The minimax algorithm for non-zero-sum games works exactly as for multiplayer games, described on *p.*1656; that is, the evaluation function is a vector of values, one for each player, and the backup step selects whichever vector has the highest value for the player whose turn it is to move. The example at the end of Section 5.2.2 (p.165) shows that alphabeta pruning is not possible in general non-zero-sum games, because an unexamined leaf node might be optimal for both players.
- **Question 7.2** The CNF representations are as follows:
  S1: $(\neg A \vee B \vee E) \wedge (\neg B \vee A) \wedge (\neg E \vee A)$.

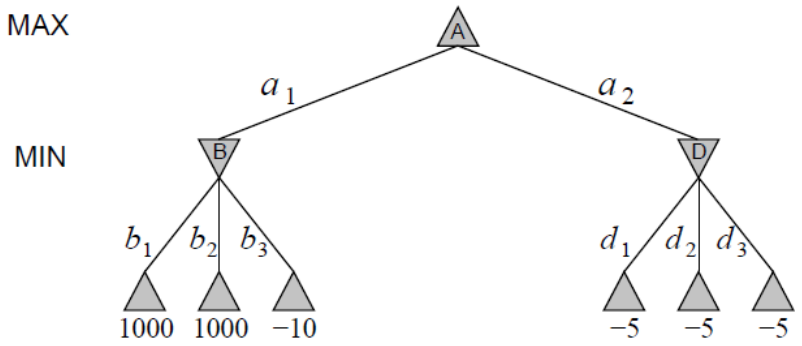

Figure 1: A simple game tree showing that setting a trap for MIN by playing a_i is a win if MIN falls for it, but may also be disastrous. The minimax move is of course a_2, with value -5.

S2: $(\neg E \vee D)$.
S3: $(\neg C \vee \neg F \vee \neg B)$.
S4: $(\neg E \vee B)$.
S5: $(\neg B \vee F)$.
S6: $(\neg B \vee C)$.