

## Problem Set 1

### Due via turn-in on Friday, January 27, 2006

**Question 1.** (based on Russell & Norvig, p. 90, question 3.9) [10 points]

The “missionaries and cannibals” problem is a famous toy problem from AI. It’s formulation is as follows.

There are three missionaries and three cannibals on one side of a river. They have a single boat that can only hold two people. They need to get to the other side of the river. The boat requires at least one person to operate. So far, so good. The problem is that if at any time there are more cannibals than missionaries on one side of the river, the cannibals will eat the missionaries. This is a “bad thing.” The goal is to find a way to get all six people from one side to the other without anyone being eaten.

- a. [2 points] Describe the minimal state space necessary for this problem.
- b. [3 points] Draw the complete directed graph of the search space. You may omit edges that lead to “invalid” states. Mark the start state and all end states. You may omit labelling the edges with the corresponding moves.
- c. [5 points] Describe (precisely) a consistent (and utile) heuristic for this problem. Explain why it is consistent.

**Question 2.** [10 points] Consider scheduling programs (jobs) on a server farm.<sup>1</sup>

Assume that the server farm is homogeneous (that is, that a job will take the same amount of time to run regardless of which machine it is run on). Each job will take a given (known) amount of time. We will assign each job to exactly one machine. Each machine, therefore, will have a set of jobs. The time it takes to run all of the jobs on a machine is just the sum of the times it takes to run each job assigned to the machine. The goal is to assign the jobs to minimize the running time of the machine with the longest running time.

- a. [5 points] Describe the state space for this problem. What is it’s branching factor? What is the cost associated with an edge?
- b. [5 points] Describe (precisely) a consistent (and utile) heuristic for this problem. Explain why it is consistent.

---

<sup>1</sup>This problem is similar to the classic problem of “bin packing” in theoretical computer science. However, you don’t need to know anything about bin packing to work on this problem.

**Question 3.** [15 points]

Assume we have a game in which moves are taken alternatively by the two players, and in the end, the player with the most number of tokens left on the board wins (Reversi — or Othello — is one such example, but this problem is more general than just that one game). Let  $E1$  be the evaluation function that returns  $-1$  if MIN has more tokens on the board,  $0$  if the players have an equal number, and  $1$  otherwise. Let  $E2$  be the evaluation function that returns the difference between the number of tokens MAX has on the board and the number MIN has.

**part a.** [5 points] Assume the game tree is small enough to be completely traversed. If we use full (unpruned) search to select our moves, will our ability to play (*ie* win) be better with  $E1$  or  $E2$ ? How about if we use alpha-beta search instead of full search?

**part b.** [5 points] How will the running time of alpha-beta search compare for  $E1$  and  $E2$ ?

**part c.** [5 points] What if the game tree is too big and we implement a depth bound so as to only search to a fixed number of ply, at which point we apply the evaluation function to the unfinished board. Assume we use alpha-beta search. How will our ability to win using alpha-beta search compare for the two evaluation functions?

**Question 4.** [45 points]

$A^*$  search.

**part a.** [25 points] Implement  $A^*$  search in its full generality. Your implementation should mirror the abstractions used to define  $A^*$  search. In particular, the algorithm should not be specific to any particular problem, but rather be able to solve any problem. Use the abstraction layers provided by C++ to do this. You should be able to write this part, and then complete the subsequent parts without needing to modify your search code.<sup>2</sup>

I expect to see good coding practices throughout this course, including this assignment. Give thought to your code's decomposition and structure. The entire amount of code written for this assignment is not large if written well. Let the abstractions of the mathematical definitions guide your code's construction. Try to find the simplest, most elegant way of expressing the algorithms.

**part b.** [10 points] Use your  $A^*$  implementation to solve the problem of the missionaries and cannibals (from problem 1). Your code should print out either "no solution possible" if it cannot find a solution or a sequence of moves which solve the problem. Use the heuristic you designed in part 1(c).

**part c.** [10 points] Use your  $A^*$  implementation to solve the job scheduling task (from problem 2). Your code should take as input the number of machines (an integer) followed by the number of jobs (an integer) followed by the running time of each job (a real value). Each value will be separated by white space. Your code should print out one line for each machine. Each line should list the job numbers to be run on that machine, followed by the total running time for the machine. Use the heuristic you described in problem 2(b).

---

<sup>2</sup>Therefore, you might want to read ahead to have some idea of what capabilities are necessary.