

## Problem Set 2

Due via turn-in on Friday, February 10, 2006

**Question 1.** [15 points]

Consider a “world” (or vocabulary) of 4 Boolean variables:  $a, b, c$ , and  $d$ . Let our knowledge base (a set of statements we know to be true) be

$$\begin{aligned}c \vee d \\ b \vee \neg c \\ \neg c \rightarrow \neg a \\ a\end{aligned}$$

**a.** [5 points] List all models (truth assignments) that are consistent with this knowledge base.

**b.** [5 points] Which of the following statements are entailed by the knowledge base?

$$\begin{aligned}a \\ b \\ \neg c \\ a \wedge d \\ a \vee d\end{aligned}$$

**c.** [5 points] Now, add the statement  $b \rightarrow d$  to the knowledge base. The entailment of some of the statements from part b has changed. Which ones are now entailed that were not before? Which ones are no longer entailed that were before?

**Question 2.** [30 points] (adapted from Russell & Norvig, problem 7.11)

Minesweeper, the well-known computer game, is actually a logical puzzle. A minesweeper world is a rectangular grid of  $N$  squares with  $M$  invisible mines scattered among them. Any square may be probed by the agent; instant death follows if a mine is probed. Minesweeper indicates the presence of mines by revealing, in each probed square, the *number* of mines that are directly or diagonally adjacent. The goal is to have probed every unmined square.

**a.** [5 points] Let  $x_{i,j}$  be a boolean variable that is true if and only if the square  $[i,j]$  contains a mine. Write down a logical statement that asserts that there are *exactly* two mines adjacent to the  $[1,1]$  square (note that there are only 3 neighbors to this corner square).

**b.** [10 points] We would like to write a general formula for the fact that  $k$  out of  $n$  neighbors contain mines. We would like it to be in conjunctive normal form (CNF). First, consider how to construct a CNF formula for the statement that *at most*  $k$  out of  $n$  squares are mines. How would you construct such a formula? (You may assume you have a procedure that returns all subsets of length  $l$  from a set of size  $m$ , for instance.)

**c.** [5 points] How would you augment your answer from part b to specify that there must be *exactly*  $k$  neighboring mines. (Hint:  $x = k$  is equivalent to  $(x \geq k) \wedge (x \leq k)$ .)

**d.** [10 points] Give a rough outline of how to use the above information and a SAT-solving algorithm to play the game of minesweeper.

**Question 3.** [35 points]

## Robotic Arm Path Planning

<http://www.cs.ucr.edu/~cshelton/courses/cs170-w2006/> (the course website) has code for this question. In particular, you will find a `README` file that explains how to operate the code. Your job is to replace the `control.cpp` file with a new one that operates the robot without hitting obstacles. You should also add any additional code files you feel are necessary. Following the steps below will probably help.

**part a.** [15 points] **Apply A\* search.** You should use A\* search to solve the problem by searching in the configuration space for a free path. If your A\* search algorithm didn't work from the last assignment, you may use the solutions. This should just involve plugging in another search space definition. If no path exists, your code should report this and quit.

Here are a few tips that might help:

- You should break the space down into grid cells (at least implicitly — don't actually form the whole grid). The big question is “how many?” In order to have a reasonable running time, you should not have too many divisions per dimension. 15 divisions per configuration dimension seems to work fine for the workspaces given. That is, if you have a 2-dimensional configuration space, you would form a 15-by-15 grid.
- You need to be able to tell whether a grid cell is occupied. As I mentioned in class, there is no great way to do this. The simulator will tell you whether a particular configuration is blocked. Use this to check some points within the cell (your choice as to which ones) and then mark the whole cell as occupied if and only if at least one of the checked points is occupied.
- There are two methods for preventing a path through obstacles:
  1. Disallow edges that go into any configuration state that is illegal.
  2. Once at an illegal configuration, disallow any outgoing edges.

That is, the first says that the path can't go through any occupied grid cells. The second says that it can go through such a cell, but it breaks the robot and the robot can't go any farther. The second is probably easier to code, but I leave the choice to you.

Use your A\* search algorithm to solve the environments in files `env0` and `env1`. They should take under 1 minute to solve. You may find it necessary to experiment a bit with your parameters to gain a good result.

**part b.** [20 points] The other environment files are too complex to be solved without a closed list. **Add a closed list to your code.** A few hints for this:

- There are no STL hash functions, so I've included a *hashset* class of my own that you may use if you wish. If you'd prefer to implement the closed list another way, that's up to you.
- If you use a heuristic that is not only admissible but also consistent (and you should be doing this anyway!), then you can avoid adding a new search node, not only if it is on the closed list (those things already expanded), but also if it is on the open list (those things waiting to be expanded). The simplest way in practice to do this is to just make the close list include any node seen thus far (so it will duplicate things on the open list). Then you just have to check against this one list.
- This took me fewer than 20 lines of code. It may be different for you, but it should not take very much programming.

Use your A\* search algorithm to solve the environments in files `env2`, `env3`, `env4`, `env5`, `env6`, `env7`, and `env8`.