CS 530 — Principles of AI
Assignment One
Out: September 25, 2001
Due: October 11, 2001

Programming Exercises

These exercises may all be completed in the programming language of your choice. Problem 3 asks you to run your program on some data. The data is available in formats that can be read in easily into a Lisp or Prolog interpreter. Of course, it is also available in simple text files that can be conveniently read into a C or Java program.

Hand in printout of code for problems 1 and 2 and extra credit A. Handin output for problems 3 and writeup for extra credit B. The amount of extra credit given will be commensurate with the amount of additional effort required to complete it.

Problem 1. Describe concrete data structures for models and policies as described in "Agents in the Real World", and implement the function OPTIMUM to compute the best policy in a model.

Describe is a cover term I use for the different ways you create data structures in different languages. In C or Java, you define the data structures (e.g., in .h or class files); in languages like Prolog where you can introduce any mnemonic structured terms to hold data, you should add a comment saying what terms you're assuming; in a language like Lisp where all complex structures are stored the same way, you describe data structures by defining functions that build and access particular instances of those structures to represent your data. Describing the data structures is not asking for something special, it's just what you would do anyway. And of course your data structures can work any way you see fit to organize your program.

Your implementation of OPTIMUM should work in the general case. However, the dog model from "Agents in the Real World" would provide a good test case for your implementation.

Problem 2. Now we introduce two new representations.

- *Model schemas* are representations that give the structure of a model but not the numerical parameters for probabilities and utilities.
- *Histories* are representations that describe a sample run of an agent, including a sequence of actions and observations and the utility outcome that the agent achieves as a result.

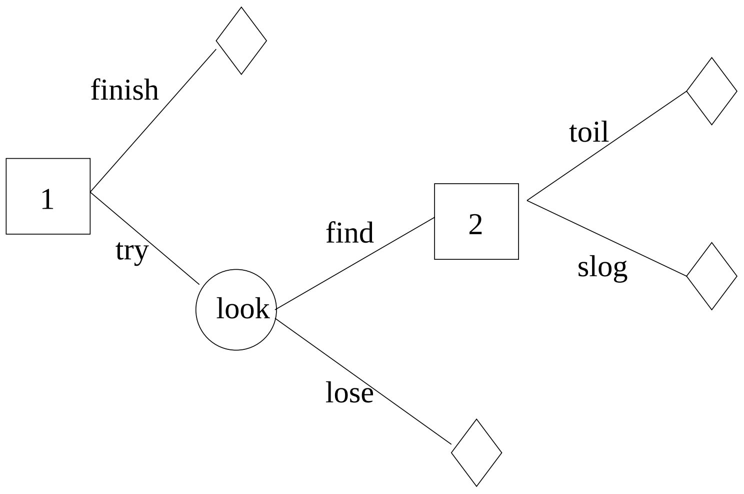
Describe concrete data structures for model schemas and a collection of histories, and implement a function TRAIN that estimates a model from a model schema and a collection of histories. In this case the histories form *training data* for your program. One simple way to write TRAIN is to traverse the model schema recursively, and keep track of the corresponding histories. At each stage, you can estimate probabilities for a node the model by the relative frequency with observed values actually occur in the training data. You can estimate utilities for the model by the average utility that you obtain in the training data. For possible outcomes that never occur in the training data—and you must handle this possibility—you can use a default utility of 0.5.

One way to test this function as you develop it is to use the data for Problem 3. There's also a sample data set of 100 trials generated according to the dog model from "Agents in the Real World", together with the actual parameter sets that you get from this data set.

Problem 3. With this assignment, the class web site contains three sets of test data files. Each data set contains five example files. The first four files in each set involve a small amount of training data (50–100 instances), while the last file involves a larger amount of training data (500-1000 instances).

Each data file comes in three formats. In the vanilla format, each line of the file describes a history. The history is reported as a sequence of observations or actions (character strings separated by spaces) followed by the utility obtained on the history (a real number, as a string of digits). In the lisp format, the file itself defines a variable that holds the training data as a list of lists. In other words, each history is a list containing symbols for the observations and actions encountered and ending with the utility obtained, and the histories appear consecutively in one big list. Finally, in the prolog format, the file defines a unary predicate history; history(L) is true if L is a list describing an element of the training set; the Prolog list takes the same form as the lisp list (modulo Prolog syntax).

All of the histories are instances of this model schema:



So sample elements might look like this

version	statement
vanilla	`try find slog 0.4`
lisp	`(setq t1 ’((try find slog 0.4) ...))`
prolog	`history([try, find, slog, 0.4]).`

All of the files in each set come from the same model with the same parameters. However, different model parameters were used across the sets.

Use your answers to Problems 1 and 2 to automatically train a model from each file of data and compute the optimal policy for each trained model. Note the variability in the answers and at least *think* about why you are finding this variability (but see Extra Credit B).

Extra Credit A. Create a final representation for a *performance model*. The performance model should include the standard deviation of observed utilities as well as the mean of observed utilities. To complete this problem you need to implement two algorithms that work on performance models in a general way.

First, implement a function that simulates a performance model by sampling. Choose actions at random; choose observations with the probability predicted by the model. Generate outcomes with a normal distribution with the specified mean and standard deviations (truncating negative values to 0 and large values to 1). The following frankly mysterious algorithm generates such normally distributed random variables, if the function $r()$ generates a random variable uniformly distributed between 0 and 1, μ is your mean and σ is your standard deviation:

```
repeat
    $x \leftarrow 2 * r() - 1$
    $y \leftarrow 2 * r() - 1$
    $r \leftarrow x^2 + y^2$
until $0 < r < 1$
return $\mu + \sigma * x * \sqrt{-2\ln(r)/r}$
```

Second, implement a function that learns a performance model from a model schema and history data. In this case, compared to problem 2, you need to also estimate the standard deviation of performance. If u_i is the utility on the ith trial, there are n trials, and u is your estimate of the average utility, use the estimate S defined here:

$$S = \sqrt{\frac{(\sum_i u_i^2) - nu^2}{(n-1)}}$$

Again, do not worry if the formula is mysterious. Just note that it can be computed iteratively in a nice way.

Extra Credit B. Present English arguments, including "back-of-the-envelope" calculations and/or tabulations of results obtained using your code from Extra Credit A, to give an analysis of the probability of determining the optimal policy correctly for each of the three models of Problem 3 as the amount of training data gradually increases. The ideal answer in each case would be a one- or two-sentence description of what's going on backed up by some illustrative numbers.