# 6.034 Midterm Quiz 2 Solutions, Spring 2012

May 14, 2012

## 1 Morsels (25 points)

To enable more efficient communication among the residents, your living group is developing a new form of communication – tapping on the walls. Your job is to develop an automated decoding algorithm for this communication channel. Let's assume that our code has 3 symbols:

- **0** – a short tap on the wall
- **1** – a more prolonged scrape on the wall.
- # – a delay of a second which indicates an end of word.

So, a communication could be:

```
000000#0101000100#
```

This is "hi all" in our language.

The problem, of course, is that this communication channel is "noisy": the 0 and 1 symbols may be mistaken for each other. We'll assume that there's a 75% chance that you will hear the correct symbol and a 25% chance that you will hear the wrong symbol.

Let's make some simplifying assumptions: We can reliably detect the end of word (#) symbol. We never miss a symbol and we can always tell when one symbol ends and another begins. So, basically, we see a stream of 0, 1 and # symbols and we need to translate them into the most likely words using a dictionary.

Assume that we have a dictionary with $N$ words, $w_0$ through $w_{N-1}$. For each word $w_i$, we have $w_i[k]$ for the $k^{th}$ symbol (one of 0 or 1) of the $i^{th}$ word. Furthermore, $w_i[k]$ for any $k$ beyond the end of the word is #.

## 1.1 HMM

We'll define an HMM that will help us decode each word. Since the # symbol is not noisy, we can just divide the message into words and decode each word separately.

We'll assume that we can access the value of $t$, the current number of symbols we have seen in the word.

The state of the HMM is defined by the variable $W_t$, which has possible values $0, \ldots, N-1$, representing the words in the dictionary. Note that the state does not change over time, because the underlying word is constant regardless of which character we are observing; $W_r$ and $W_s$ have the same value for any two times $r$ and $s$, but we treat them as different variables because our beliefs about $W_r$ are based on different observations than our beliefs about $W_s$.

$E_t$ is the evidence variable at time $t$, which is simply the observed symbol at time $t$.

Assume all the words $w_0, \ldots, w_{N-1}$ are equally likely, and that the probability of a word does not depend on what other words are in the message.

Fill in the values of the probabilities below, based on the description above. Each answer is a single probability, either a constant or an expression involving $N$.

1. The sensor model for the HMM is (for $i = 0, \ldots, N-1$):

   (a) If $w_i[t] = \#$ then $P(E_t = \# \mid W_t = i) = 1$

   (b) Otherwise, $P(E_t = w_i[t] \mid W_t = i) = 0.75$

2. The transition model for the HMM is (for $i = 0, \ldots, N-1$):
   $P(W_{t+1} = i \mid W_t = i) = 1$

3. The initial state distribution for the HMM is (for $i = 0, \ldots, N-1$):
   $P(W_0 = i) = 1/N$

## 1.2 Decoding

Let's consider a situation where we have the following 3 words in the dictionary:

- $w_0 = 01\#\#$
- $w_1 = 10\#\#$
- $w_2 = 110\#$

Suppose that we see the following message: **11#**. Give numerical values for the state distribution at each time step.

We prefer that you use fractions for the probabilities.

| | $i = 0$ | $i = 1$ | $i = 2$ |
|---|---|---|---|
| $P(W_0 = i)$ | 1/3 | 1/3 | 1/3 |
| $P(W_1 = i \mid E_1 = 1)$ | 1/7 | 3/7 | 3/7 |
| $P(W_2 = i \mid E_1 = 1, E_2 = 1)$ | 1/5 | 1/5 | 3/5 |
| $P(W_3 = i \mid E_1 = 1, E_2 = 1, E_3 = \#)$ | 1/2 | 1/2 | 0 |

Give a **short** intuitive explanation for the probabilities that you computed.

We start with equal probability for each word. When we see a 1, words 1 and 2 become 3 times as likely as word 0 (which has a 0 for the first character). Multiplying the appropriate observation probabilities times the state probabilities, we get 1/12, 3/12, 3/12, then we normalize. When we see another 1, we multiply the observation probabilities times the state probabilities, we get 3/28, 3/28, 9/28, then we normalize. At the third time step, we see the end of word symbol, so we multiply by 1 or 0 depending on whether the word ends there or not.

Bottom line: We have independent probabilities of errors on each character of the message, so given an observed message of 11, it is equally likely that the actual message was 10 or 01.

# 2 Planning (25 points)

When our robot from Project 3 finishes its shift at the factory, it goes home and wants to prepare a well deserved meal. But, before it can do that, you must build the planner that enables it to do this.

There are a set of fixed objects (more like locations) that we know about: `TABLE1`, `TABLE2`, `STOVE1`, `STOVE2`, `SHELF1`, `SHELF2`.

There are a set of movable objects that we know about: `PAN1`, `PAN2`, `PLATE1`, `PLATE2`, `FOOD1`, `FOOD2`.

We have the following types of assertions in the world. First some assertions indicating the "types" of the individuals:

- `stove(x)` – x is a stove
- `shelf(x)` – x is a shelf
- `table(x)` – x is a table
- `movable(x)` – x is movable
- `food(x)` – x is food
- `plate(x)` – x is a plate
- `pan(x)` – x is a pan
- `cooked(x)` – x is cooked

Then some assertions indicating relationships

- `on(obj1,obj2)` – object obj1 is on object obj2 (only one object can be on an object and every movable object must be on some object)
- `clear(x)` – there is no object on top of x.

## 2.1 Rules

Write rules (by specifying the preconditions and the add and delete assertions) for the following operators (you can use the variables in the argument list of the operators in your add and delete assertions):

1. `cook(f, p, s)`: where f is food, p is a pan and s is a stove. If the pan is on a stove and the food is on the pan, the food becomes cooked.

```
cook(f, p, s):
   Pre: food(f), pan(p), stove(s), on(f, p), on(p, s)
   Add: cooked(f)
```

2. `puton(o1, o2, o3)`: where o1, o2 and o3 are objects, o1 must be movable. Used to move o1 from on o2 to on o3. You can only pick up objects that are clear and put them on clear objects. The operator should update clear assertions as necessary.

```
puton(o1, o2, o3):
   Pre: movable(o1), clear(o1), on(o1, o2), clear(o3)
   Add: on(o1, o3), clear(o2)
   Del: on(o1, o2), clear(o3)
```

## 2.2 Search

Assume that, as always, we never repeat a state on a path (Pruning Rule 1 in 6.01). Answer these questions for the example above, with 6 movable objects, 6 fixed objects.

1. If we used Depth First Search for planning in this domain, what is the (worst-case) length of a valid plan that DFS could find? Is it on the order of 6, 6*6=36, 2**6=64, 6! = 6*5*4*3*2*1 = 720, or unbounded (pick the closest answer). Describe briefly what such a longest plan would involve.

   `On the order of 6!. The path could end up building up (and tearing down) all possible stacks among all the objects. But, there cannot be any repeated states, so it is not unbounded.`

2. If we used Breadth First Search for planning in this domain, what is the (worst-case) search branching factor, involving only `puton` actions? Count only "legal" instances, in which the preconditions are satisfied. Describe briefly how you got your answer.

   `36 - each of 6 objects could move to one of 6 different locations (6 of those moves are trivial -- so 30 is also a good answer).`

# 3 Minesweeper CSP (25 points)

Let's consider how we might use CSPs to play Minesweeper.

## 3.1 Formulation

The Minesweeper board, as we saw in Project 1, is fundamentally a grid of squares, each of which either has a mine (denote this as 'x') or no mine (denote this as 'o'). We will call an assignment of 'x' or 'o' to each square on the board a "labelling" of the board.

At any point in the game, we have only visited some of the squares, so we see a display that has squares labeled with either '?', indicating unknown, 'x', indicating a known mine, or a number indicating how many of the neighboring squares are mines. Note that this is **not** a labelling of the board, it's information available for the player to reason about the underlying labelling. We will call this a "view" of the board.

You are given the current view of the one-dimensional Minesweeper board. Define a CSP with unary and/or binary constraints on the underlying labelling of the board.

Let's consider only one-dimensional Minesweeper, that is, playing on a 1xN board.

Our goal is to use the CSP to pick what moves to make at each step of the game. Once we make a move, we get a new view of the board, set up a new CSP and pick the next move, and so on.

HINT: The relevant variables for the CSP at any time step are the '?' locations in the current view, that is, the unvisited squares. The constraints will depend on the squares with numbers and mines ('x') in the current view.

1. What is the domain of each variable?

   `'x' or 'o'`

2. Describe the unary constraints for this type of CSP? Or indicate if there are none.

   `When there is a number with one ? neighbor, then that square can only have a label that makes the number correct.`

3. Describe the binary constraints for this type of CSP. Or indicate if there are none.

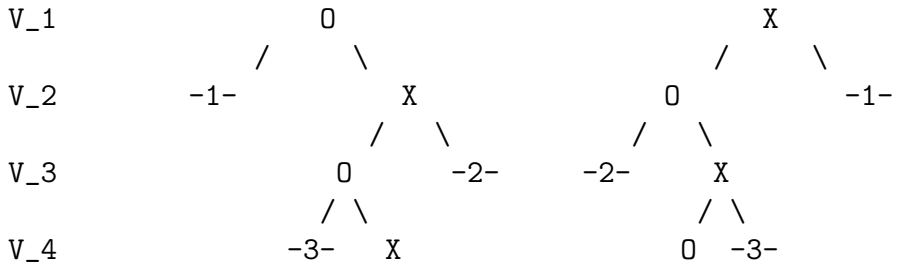
   `When there's a number on a square with two ? neighbors, then there is a constraint between the ? squares so that the number of mines make the number correct.`

## 3.2 Example

Consider the following view of a one-dimensional board, the variables are numbered $V_1, \ldots V_4$ from left to right.

```
|?|1|?|1|?|1|?|
```

1. Draw the constraint graph for this problem. $V_1 - - - V_2 - - - V_3 - - - V_4$
2. Draw the full backtracking with forward checking (BT-FC) search tree; order the variables by number. Find all solutions. Make sure that you cross out any inconsistent assignments as soon as they are discovered.




The `-i-` label above indicates that this value is inconsistent with the value of $V_i$ above.

# 4 MDPs (25 points)

Let's consider a simple 3 state MDP with two actions (L and R) – you might want to draw a picture for yourself. The transition probabilities are:

```
Action L:
            State 1  State 2  State 3  (outcomes)
In state 1:    0       1/4      3/4
In state 2:   3/4       0       1/4
In state 3:   1/4      3/4       0

Action R:
            State 1  State 2  State 3  (outcomes)
In state 1:    0       3/4      1/4
In state 2:   1/4       0       3/4
In state 3:   3/4      1/4       0
```

The reward in state 2 is 1 and 0 elsewhere. The discount factor is 0.5.

The utilities (values) of the states, with discount factor = 0.5, are **approximately** $U_1 = U_3 = 0.5$ and $U_2 = 1.25$.

1. Show the utility (value) estimates for the first two iterations of the value iteration algorithm. To make things simpler, assume that you keep a copy of the utility estimates from the previous iteration and use those in the new iteration.

```
    Initial: U[1]=0      U[2]=0       U[3]=0


Iteration 1: U[1]=0       U[2]=1       U[3]=0


Iteration 2: U[1]=3/8       U[2]=1        U[3]=3/8
```

2. Using the approximately optimal values of the utility: $U_1 = U_3 = 0.5$ and $U_2 = 1.25$. Compute the best action for state 1 from these values numerically. Explain how you did it.

   The best action is R. The expected utility of action L is:
   P(S2|S1, L)*U(S2) + P(S3|S1,L)*U(S3) = 1/4*1.25 + 3/4*0.5
   The expected utility of action R is:
   P(S2|S1, R)*U(S2) + P(S3|S1,R)*U(S3) = 3/4*1.25 + 1/4*0.5

3. Now, imagine that we don't know the MDP and we need to do reinforcement learning. Why is a random exploration scheme (pick a random action some fraction of the time) generally inferior to using an exploration function involving an optimistic estimate of reward? Explain briefly.

   The exploration function encourages exploration of previously unexplored states, by giving them an optimistic reward. The random exploration strategy will end up revisiting states that have been explored already as frequently as those that have not.