

CS472-Foundations of Artificial Intelligence

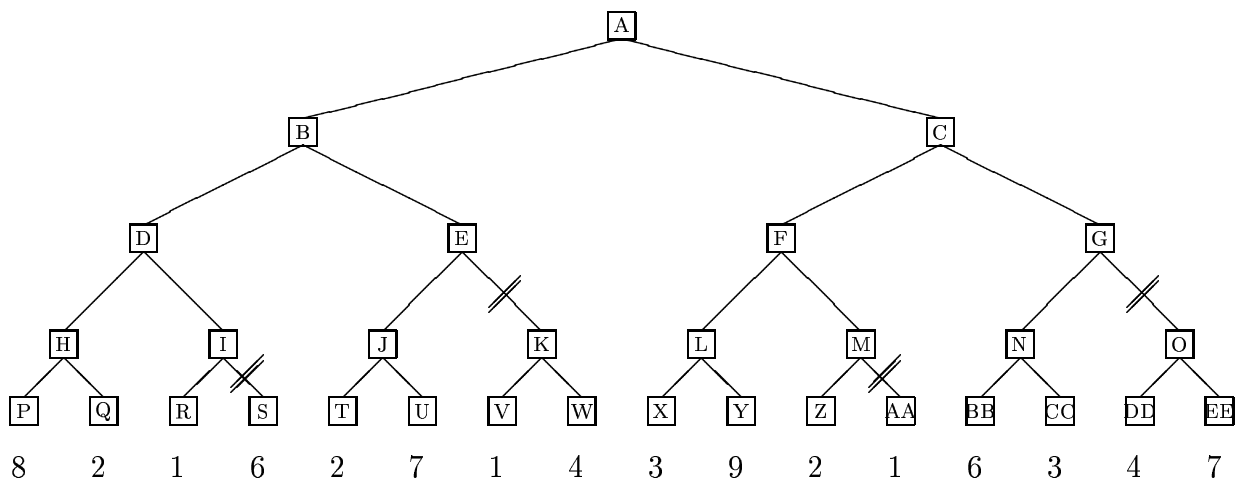
Assignment 2 – Solutions

October 25, 1999

Problem 1 (30pt)

(a). C. 3.

(b).

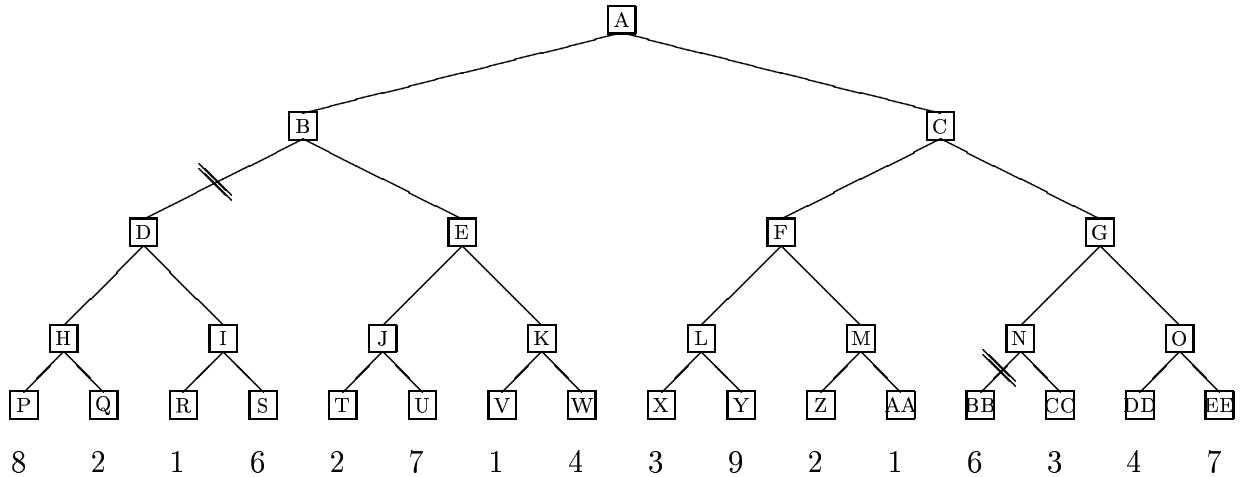


10 leaves are evaluated.

The winning path is **A, C, F, L, X.**

Static evaluation values that do not need to be computed are: **S**,
V, **W**, **AA**, **DD**, **EE**.

(c).



11 leaves are evaluated.

The winning path is **A**, **C**, **F**, **L**, **X**.

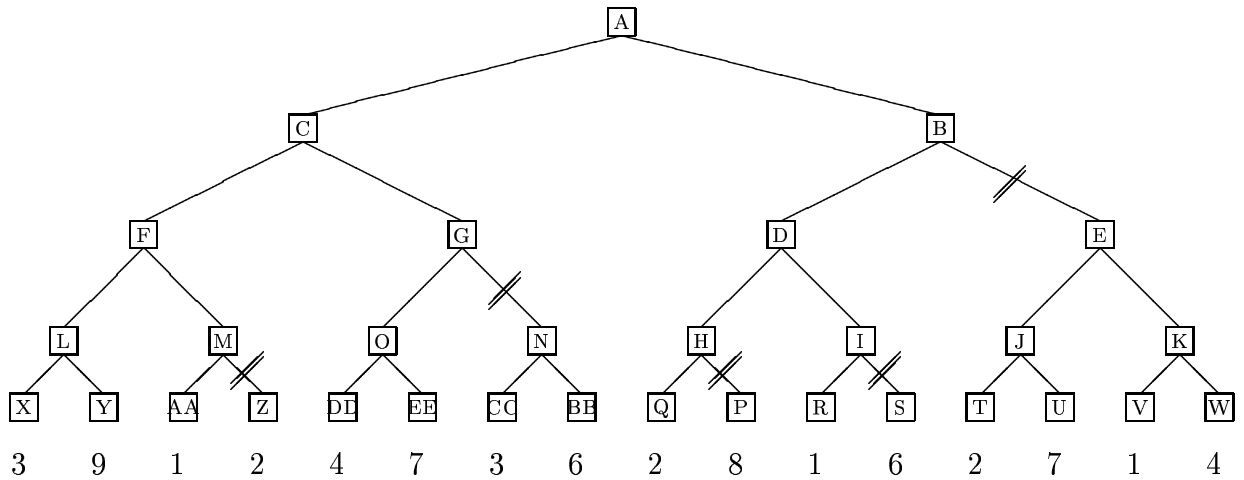
Static evaluation values that do not need to be computed are: **P**, **Q**,
R, **S**, **BB**.

(d). The number of nodes that skipped could be different, if different evaluation order is used.

For example, if we use perfect ordering (examine first the best successors), only 7 leaf nodes are evaluated (we reorder the nodes of the tree so that the “perfect” evaluation order is left-to-right).

If we assume the depth of the tree is d , and there are b legal moves at each point, then the alpha-beta with “perfect” ordering only needs to examine $O(b^{d/2})$ nodes to pick the best move, instead of $O(b^d)$ with minimax.

In practice, e.g. for chess, we can order the moves by the significance of the purpose that a move might accomplish, such as captures, threads,



king safety, center control, etc. A domain-independent technique is to do iterative deepening search, use the backed-up values of one iteration is used to determine the evaluation order in the next iteration.

- (e). First off, there may be many possible goal states, so backward-search would have to either choose (with the possibility of choosing wrong) or try them all. Many of these goals may be unattainable, so searching from them is a waste of time. Finally, in many cases the distance from goal node is so far that there would not be enough resources to search back to the current state.
- (f). Each player has his own utility function. Apply the utility functions to each terminal state to get a tuple of utility values. Instead of a single utility value, the tuple of utility values is backed up from each node. Each player chooses the move that maximize his utility. (Note: a utility function could reflect the goal of advancing the player's own standing as well as the goal of damaging other players). The utility functions need not be symmetrical in that they may not be the same function even if we permute the roles of players. (For example, even the player of top node does not play against one specific player, he may assume all other players would play against him. However, since there is no explicit alliance, we should not assume that the utility functions of all

other players are identical.)

Problem 2(15pt)

1. (a) (3 pts) I gave one point for each of the following observations
 - i. $9!$ gives an upper bound on the number of possible games
 - ii. Games may end before 9 moves. A game may end in as little as five moves, therefore a good lower bound is $9!/4!$ (**not** $5!$)
 - iii. To compute the actually number of possible games we must also consider symmetrically equivalent games
- (b) (3 pts) See Figure
- (c) (3 pts) See Figure
- (d) (3 pts) See Figure. Best starting move is for X to place a mark in the center square.
- (e) (3 pts) See Figure

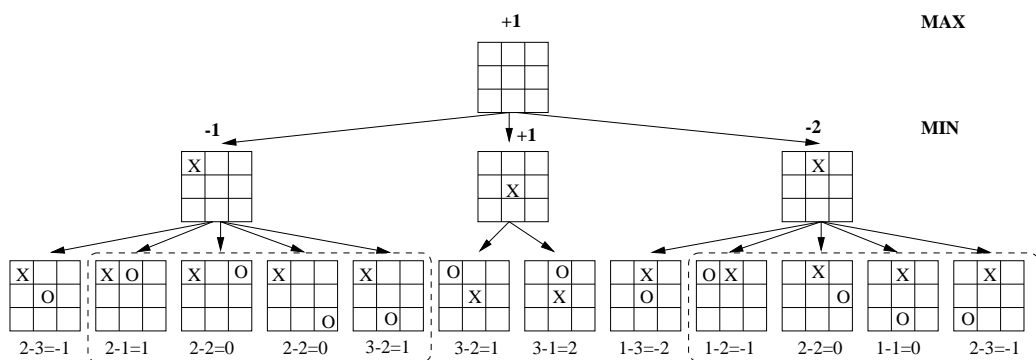


Figure 1: figure for problem 2

Problem 3 (10pt)

1. (3 pts.) Instances will be relatively easy compared to those at the threshold ratio of 4.3.
2. (3 pts.) The key point to note for part (d) and (e) is that our discussion of random 3SAT was concerned with average-case complexity. In average-case complexity you have a problem (here 3SAT) **and** a problem distribution.

The VLSI problem is mapped into SAT but it's unlikely that this would be similar to sampling from pure random instances at the ratio of 10. So, strictly speaking, we cannot say anything about the hardness of this instance.

3. (4 pts.) Similar as for (d). (We can study the issue here more carefully. The distribution of random graphs leads to some distribution on random 3SAT. One may be able to say something about the hardness of that 3SAT distribution, but it's most likely different from the random 3SAT distribution discussed in class.)

Problem 4 (30pt)

1. (10 pts.) (sketch only) The main point to exploit is that each variable occurs in at most c clauses. This means that when you flip a variable x , you only have to update the entries in array A of variables that share a clause with variable x . That means, at most $2 \times c$ entries. With each variable x , you have a list structure that gives you the list of clauses in which x occurs. For each clause you update the entries of A appropriately for the variables occurring in the clause. To do this, you have to consider several possible cases. For example, if the clause was satisfied *and* after flipping x it became unsatisfied, then a flip of any of the variables in the clause would make the clause satisfied again. This means you can add one to the entries in A of the two other variables in the clause. (The update for x is quite subtle. Try an example!) Etc. Visit each clause containing x to make the necessary updates.

Since you're only doing at most a constant amount of work (visit c clauses, update $2 \times c$ entries), the time complexity is $O(1)$.

I took off one point if you didn't mention the fact that the value of the array for variables that are in the same clauses with the flipped variable might change as well. I took one point off for slightly inefficient algorithms ($O(m)$ or $O(n)$), 2 points for algorithms of complexity $O(mn)$ and 3 points if the space required for your algorithm was mn or if complexity was worse than $O(mn)$. I took off additional points for incorrect algorithms.

2. GSAT generally goes toward a solution for the SAT problem. There are some situation in which GSAT is *running away* from the correct solution, namely the situation in which setting the value for a certain variable as the one that makes more clauses true is a wrong decision, the correct solution having the opposite value for that variable. For example in: $(\neg A) \wedge (A \vee B) \wedge (A \vee C) \wedge (A \vee D) \wedge \dots \wedge (A \vee Z)$ for almost all the initial assignments (except if we are one step to the solution) if A is initially true it will chose A to be flipped (is the only one variable in a unsatisfied clause), but if A is false choosing A to be flipped is the best solution (from GSAT point of view) because A will make much more clauses to be true than any other variable. In this moment we are back from where we started, thus GSAT will not find a solution. If initially A is false we are in the same situation, but we begin with the second step, continuing with the alternation we mentioned, never finding the solution. In such a situation we need an exponential number of restarts for GSAT in order to find a solution. For finding such a situation and arguing that it is hard 10 points were given.

A general situation in which the ratio clauses/variables is 4.3 is far less hard for GSAT, altho in general is harder than problems with other ratios. Taking only 100 clauses (and 100 variables) in the example above will require approximative. 2^{100} restarts in order to find a solution. Solving a general problem with the 4.3 ratio, having 100 variables is much less harder. Writing a really hard problem of this kind would require probably 10000 variables. For these reason this solution was not considered correct (except if somebody wrote a huge problem, which was not the case). Usually 5-6 points were given depending on how elaborate the solution was.

Incomplete solutions in the right direction got 5-7 points if they expressed the correct idea even if the example was not build. 1-3 points

got the *tentative* solutions and 0 point the absence of any solution.

3. If a formula is satisfiable, starting from a random initial assignment, the satisfying assignment can always be reached (sometimes in a very long time, so the cutoff must be big enough) because eventually WalkSAT will visit all possible assignments for the variables initially in the unsatisfied clauses and the variables that are not in this category are already assigned proper values (formula is satisfiable starting from that position and the value of this variable will not be touched by WalkSAT, which means that initial assignment is the good one). For this part complete and pertinent explanations received 5 points. For incomplete but in the right direction explanations 3-4 points, depending on the degree of closeness to the right solution, and for wrong explanations 1-2 points. If this subject was not touched at all 0 points were given.

The problem of reaching all satisfying assignments is a little bit delicate. Let's observe that this question is about the theoretic ability to find all solutions not about how efficient this process is. Two points of view have been considered correct because the problem was not explicit about which one to adopt.

- (a) If we are allowed to restart WalkSAT as much as we want (an infinity number of times for example), all the solutions can be founded. Even if WalkSAT does nothing it might *fall* on the right solution with a probability bigger than 0 so trying long enough we find all the solutions.
- (b) If we interpret the question like: *can the WalkSAT alg. form a random initial assignment find all satisfying solutions* than the answer is no. This is because there are situations in which for example we have many solutions and in some of them a predicate P is true and in some of them P is false and P is not present in any unsatisfied clause (for some initial assignment). In such a situation is impossible to find all solutions because WalkSAT will not touch P and will not find the solutions in which P has the opposite assignment.

Correct answer was considered sustaining and correctly arguing one of the two points of view, 5 points were given. For incomplete but in the right direction explanations 3-4 points, depending on the degree of

closeness to the right solution, and for wrong explanations 1-2 points.
If this subject was not touched at all 0 points were given.

Problem 5 (15pt)

1. Each state in the gaming tree would represent the following:
 - (a) Number of coins you have,
 - (b) Number of coins your opponent has, and
 - (c) Number of coins left on the table.

Also, the operators are:

- (a) Number of coins picked up by you or your opponent, and
 - (b) Paying 2 to the opponent.
2. The cost functions used to evaluate the terminal nodes are:

If your opponent picks up the last coin

$$\begin{cases} MAX(n) = MAX(n) + 2 \\ MIN(n) = MIN(n) - 2 \end{cases}$$

If you pick up the last coin

$$\begin{cases} MIN(n) = MIN(n) + 2 \\ MAX(n) = MAX(n) - 2 \end{cases}$$

]

3. The Minimax tree is as follows:

Each state is shown as a rectangular box consisting of all the elements. Operators are shown acting on the states. With each terminal node is associated a rounded rectangular box consisting of the static evaluation function value for MAX at that terminal node. This value eventually float up through the tree traversing other nodes depending on whose turn it is.

MINIMAX TREE

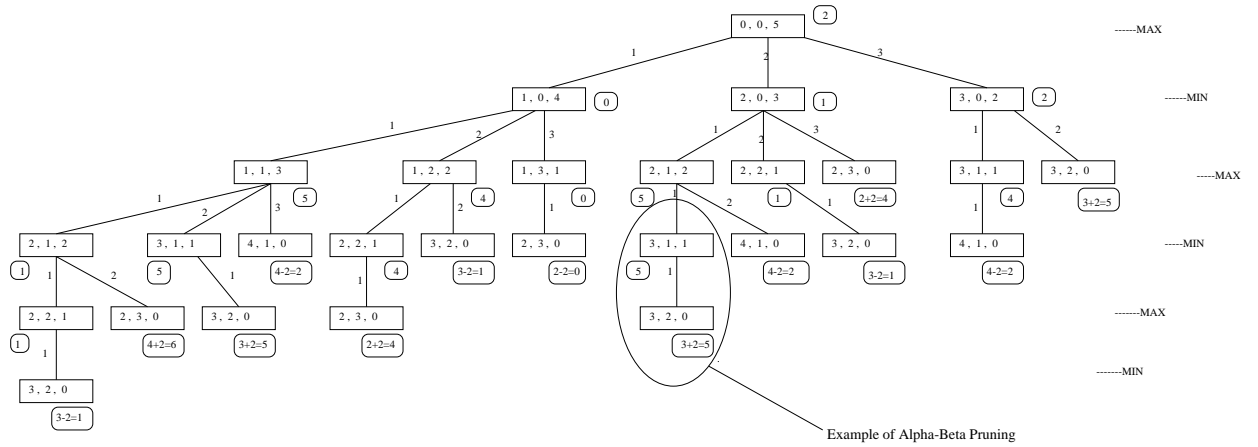


Figure 2: Minimax tree

4. Your opponent courteously offers to let you go first. In this case, you're definitely going to lose but the best move which would save you from losing more money is to select 3 coins in the first chance. [1 pt]

The opponent is assumed to play in such a way so as to minimize your money will go for 1 coin which eventually leaves you with picking up the last coin and paying 2 to your opponent. So you'll finally earn 2 from this move. [1 pt]

5. There are quite a few examples of alpha-beta pruning in this problem if you search from right to left. One of them is illustrated in the minimax tree diagram.

In the diagram, after traversing through node 2,2,1 which has the static value 1 for the MAX player, MIN at node 2,0,3 has a value of at most 1. The question is whether MIN could still find an even lower value possibly via node 2,1,2.

However, after going to node 2,1,2 and 4,1,0 you find that 2,1,2 (MAX level) has a value of at least 2 (from node 4,1,0), so there is no need to explore further under node 2,1,2, because we're looking for something less than 1 at node 2,0,3.

[Remark: Consider following the tree from 1, 1, 2, 1, reaching state [3, 2, 0], concluding that 5 is the maximum value and using this to prune other nodes in the same subtree. This a form of pruning is not valid alphas-beta pruning. (You're using information about the maximum of the evaluation function. The alpha-beta procedure does not.)]