CSCE 420 Homework One

Cameron Quilici – quilicam@tamu.edu – 630004248

February 23, 2023

---

1. Solve the following problems:

   (a) Given the simple game tree (binary, depth 3) below, label the nodes with up or down arrows, as discussed in the textbook.

   (b) Compute the minimax values at the internal nodes (write the values next each node).

   (c) Should the player 1 take action A or B at the root?

   (d) What is the expected outcome (payoff at the end of the game)?

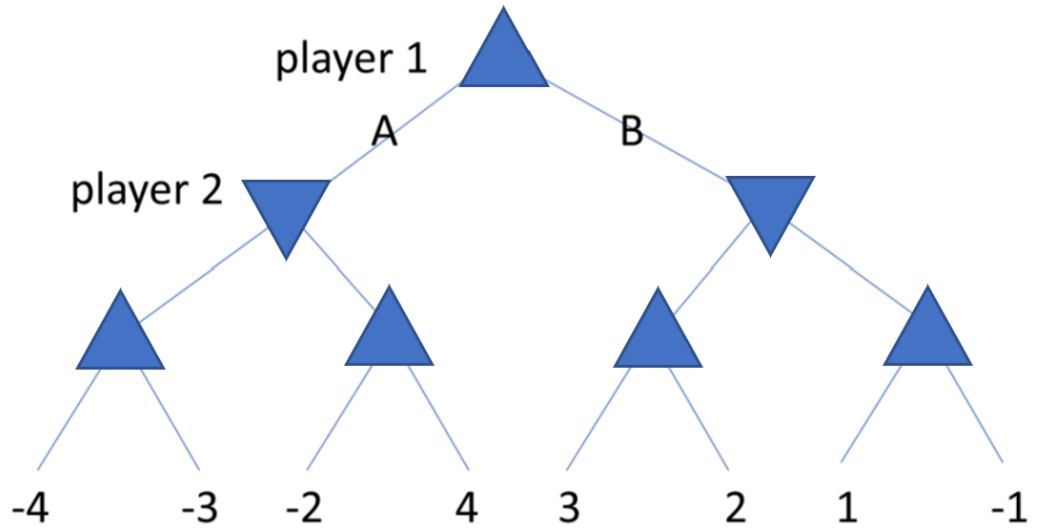
   (e) Which branches would be pruned by alpha-beta pruning? (circle them)

   (f) How could the leaves be relabeled to maximize the number of nodes pruned? (you can move the utilities around arbitrarily to other leaves, but you still have to use -4,-3,-2,-1,+1,+2,+3,+4)

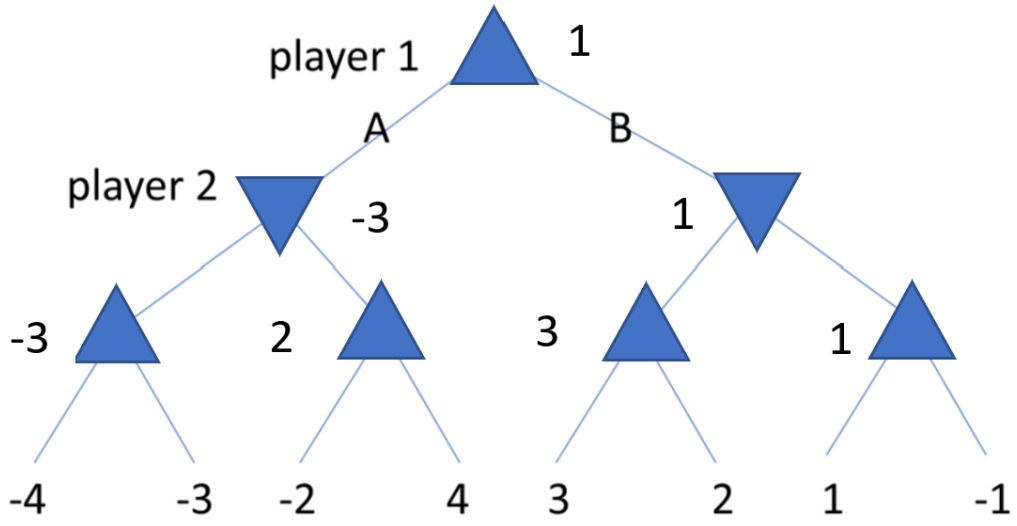
   (g) How could the leaves be relabeled to eliminate pruning?

Solution:

(a) Below is the graph labeled with arrows.



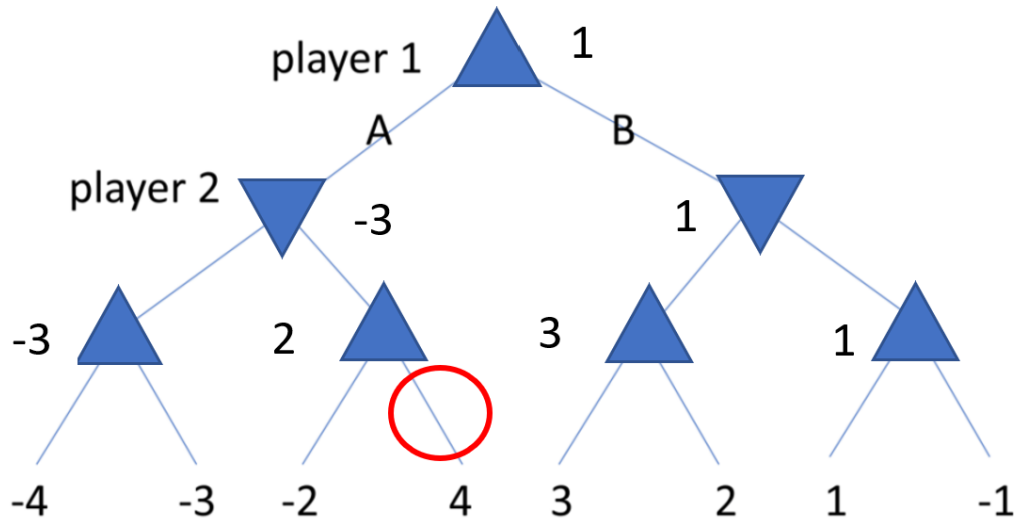
(b) Now, with the labeled minimax values at each node.



(c) Player 1 should take action **B** at the root because it leads to the highest possible score.

(d) The expected outcome at the end of the game is the final minimax value for player 1 at the root node which is 1 in this case.

(e) The branch that is pruned during α/β pruning is circled in red below.



(f) I believe there are multiple variations that yield the same pruning, however the one I found prunes 4 total branches:

$$3 \quad 2 \quad 4 \quad 1 \quad -3 \quad -4 \quad -2 \quad -1.$$

(g) One that works is:

$$1 \quad -1 \quad 2 \quad -2 \quad 3 \quad -3 \quad 4 \quad -4.$$

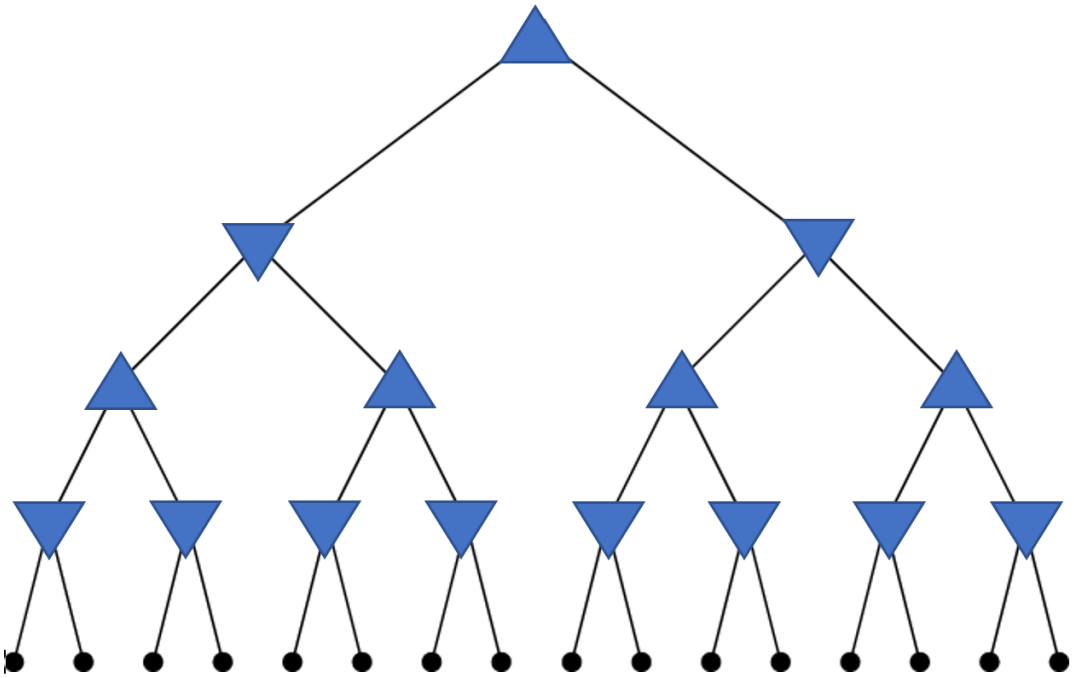
2. In a simple binary game tree of depth 4 (each player gets 2 moves), suppose all the leaves have utility 0 except one winning state (+1000) and one loosing state (-1000).

   (a) Could the player at the root force a win?

   (b) Does it matter where the 2 non-zero states are located in the tree? (e.g. adjacent or far apart)

   (c) If this question was changed to have a different depth, would it change the answers to the two questions above? If yes, how do the answers change? If no, explain why no change would happen.

Solution: We can picture this minimax problem with the following diagram:



(a) From the figure above, it is not hard to see that the player at the root cannot force a win. Say the two non-trivial utilities are on the two leaves of a common child, then the minimizer picks the -1000 and passes it up to the parent. Then, the maximizer gets to pick between -1000 and 0; obviously it will pick 0.

(b) It does not matter where the two non-zero utility values are in the tree (adjacent or far apart). This is due to the same reason stated in part (a). No matter where the two are, the minimizer will always pick $\min(0, -1000) = -1000$ and $\min(0, 1000)$ which discards the 1000 utility value that the maximizer *needs* propagated up to the root in order to win. Further, once this 1000 value is discarded, the maximizer either must choose from either $\max(0, 0)$ or $\max(-1000, 0)$ which will always result in a 0.

(c) The only depth where the player at the root (assuming the player at the root is a maximizer), is in the trivial case where the tree is depth 2. In this case, the maximizer simply picks $\max(-1000, 1000) = 1000$ and they can force a win.

Even if the tree has depth 100, the minimizer will *always* have a chance to play and will *always* pick the lowest value and propogate it up to its parent (a maximizer) in which case the maximizer will have to choose between either $\max(0, 0)$ or $\max(-1000, 0)$ which will always result in a 0.

3. Consider the task of creating a crossword puzzle (choosing words for a predefined layout).

   Suppose you have a finite dictionary of words (see `/etc/dictionaries/` on Linux distributions for about 50k English words; used for 'ispell' command).

   Given a layout with a list of n "across" and m "down" words (see the example in the Figure below, the goal is to choose words to fill in (that satisfy all the intersections between words). (Clues for these words can then be defined later by a puzzle expert.)

   Show how to model this crossword puzzle design problem as a CSP.

   (a) **What are the variables, domains, and constraints?** (Hint: Not all variables have to have the same domain.)

   (b) **Draw the constraint graph.** Label one of the nodes and one of the edges as examples (the nodes are easy; the edges require some thought). (of course, you don't have to write down the labels completely; just explain what they would look like and give a couple examples)

   Describe the first couple of steps of how standard backtracking search would work (selecting variables and values in default order), making reasonable choices as you go. (e.g. you could state: suppose 'ace' is the first 3-letter word starting with 'a'...).

   Describe how using the MRV would change the search; describe the first couple of steps again. (Hint: You might need to make some assumptions about how many words have 3 letters, 4 letters, etc, or how many 5-letter words start with 'a' or 'y'...)

Crossword layout:

1across (len=7)
2across (len=5)
3across (len=6)
1down (len=3)
2down (len=2)
3down (len=6)

Example of a solution:

```
a b a l o n e
    c       a
    e a g e r
            b
      h     u
  a b a s e d
```

Solution:

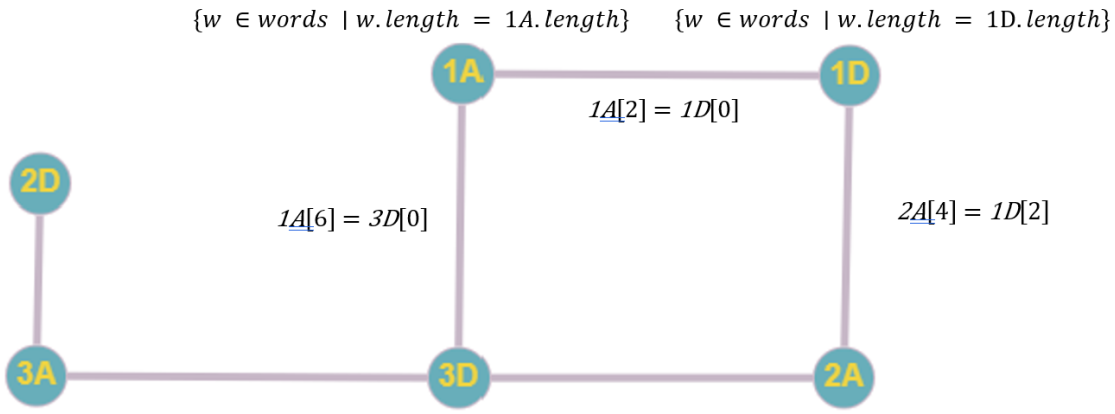
(a) The variables in this proposed CSP are the words which need to be filled in. Consider the list of n "across" and m "down" words to be filled in, then the set of variables for this CSP are all of these lengths and orientations.

For this particular problem, there are multiple domains. Namely, the domain for a certain variable is based on how many letters are in that particular variable. For instance, the domain for `1across` (from the example above) would be all 7-letter words in the dictionary. Suppose we have the set of all English words called *words*. Then we have

$$\text{dom}(var_i) = \{w \in words \mid w.\texttt{length} = var_i.\texttt{length}\}.$$

There are a couple constraints. First and most obvious, we constrain any possible word to be a real word in our set *words*. That is, a solution to a variable cannot be some random string of letters. The second and perhaps more important constraint is that two variables that overlap at some point must share the same letter at that position, *and* the word still must be a real word.

(b) Consider the example in the figure above. Each variable is given a concise code name: `1A` and `1D` for "one across" and "one down" respectively, so on and so forth. Furthermore, suppose we can index each variable's letters as we would an array (0-based). Then, we have a CSP graph that would look something like this.



This is saying that variables `1A` and `3D` have an overlapping letter and moreover that overlapping letter is in the sixth index of `1A` and the zeroth index of `3D` and therefore the letters at those indices must be the same.

The steps for backtracking for a crossword puzzle CSP will be slightly different from the example we did in class with map coloring. With the crossword puzzle, we can any word with constraints and make it out root word. Consider we picked the word "abacist" for `1A`. Then, say the next variable we check is `1D`. We start checking all three letter words in the English dictionary until we find one that starts with the letter "a." We find it immediately so we move on to the next variable, `2A`. Once again, we check every single 5-letter word until we find one that meets the constraint (starts with an "h"). We find one and it is "habit."

Then we move on to the next variable, `3D`. Say, for the sake of example, we search all words and cannot find one that meets the constraints. Then, we backtrack to the `2A` variable and pick a different word, say "hacks." Then, we can continue this process until we find a solution for each variable that meets all the respective constraints (see figure below).

`1A`	abacist	Pass
`1D`	aah	Pass
`2A`	aahed	Failure
	abaft	Failure
	⋮	⋮
	habit	Pass
`3D`	abcess	Failure
	accost	Failure
	⋮	⋮
Assume nothing works		
	Backtrack	
`2A`	hacks	Passes
	⋮	⋮

Of course, this method is not the best because we are checking all the words of the required length even though we know beforehand (based on the selection of the previous variable) what the constraints are for the next word. So we can improve this process using the MRV heuristic. Now, instead of checking *all* words of length n for a given variable, we only check the ones that meet the necessary constraints. This will speed up the process greatly. See the example below that illustrates the first few steps of this process.

`1A`	abacist	Pass
`1D`	aah	Pass
`2A`	habit	Pass
`3D`	abcess	Failure
	accost	Failure
	⋮	⋮
Assume nothing works		
	Backtrack	
`2A`	hacks	Passes
	⋮	⋮

Notice how we don't have start from the first word and instead can start searching from where the constraints are met.