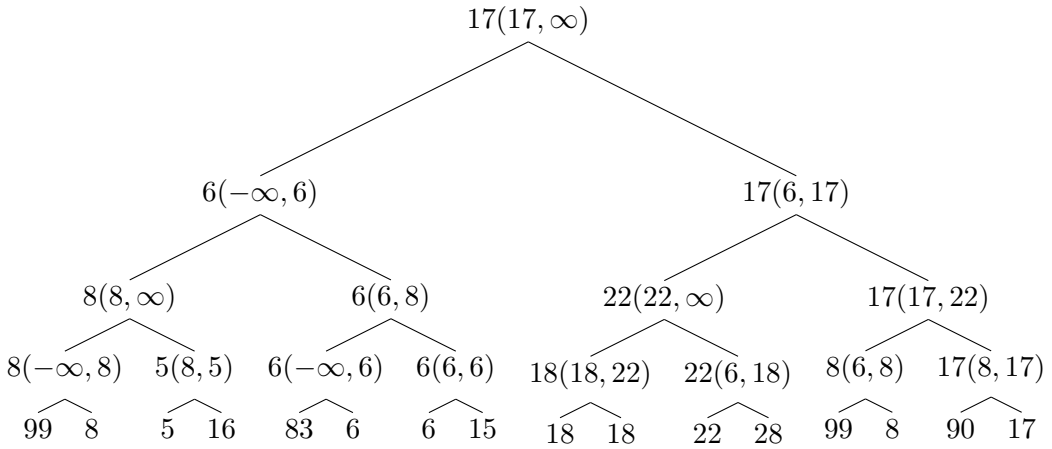
Project 2

Joshua Westbrook & Arifur Rahman

1 problem 1

2 problem 2

2.1 a and b



2.2 b

The alpha and beta values are on the tree above.

The 4th(16) and 8th(15) leaf nodes(from the left) are the only ones pruned from the tree.

2.3 c

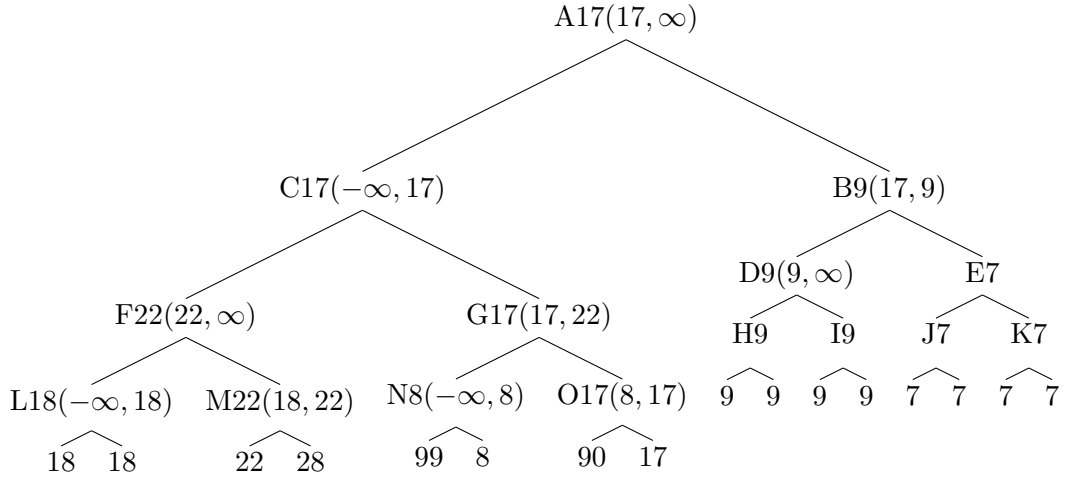
The MAX player will choose to take the action which puts it on the right node of the tree both with exhaustive Minimax and when alpha-beta pruning is employed. In general the best move computed is the same. Alpha-beta pruning purely reduces the amount of time spent looking at nodes which will not be reached in any situations anyways.

2.4 d

The re-ordering will move put the node with the highest heuristic on the left side of the tree and continue to order them from highest heuristic value to lowest since this is for

the MAX player looking at it and the MAX player prefers as high a result as possible.

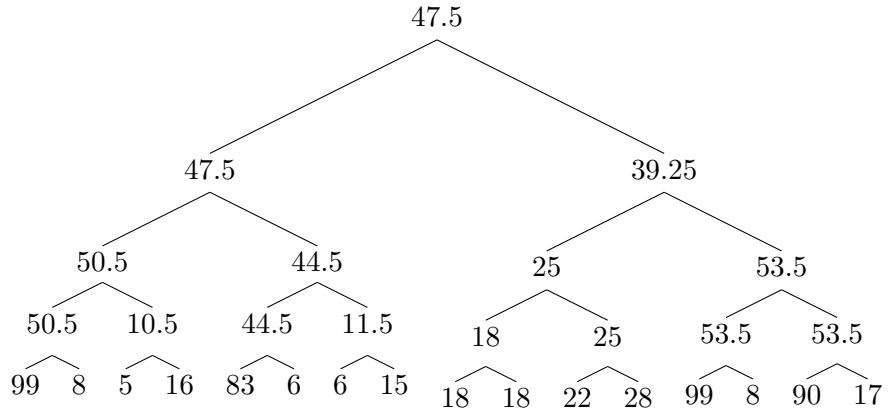
The re-ordered tree with the format [letter on chart][value of node]([alpha, beta]) is below.



All nodes under node E at depth 2 and all nodes under node D at depth 2 were pruned by the heuristic. Node E was also pruned. This happens because the heuristic of 9 at node D tells Alpha-Beta pruning that there is no point in looking under it since we already got a worst case scenario of 17 for the maximum player from the left side of the tree. Since we already have it ordered in a maximum to minimum ordering the algorithm is able to know that there is no point on going to any node to the right of a suboptimal node compared to the previously looked at ones and therefore prunes node E also. This means 13 nodes were pruned compared to the 2 pruned previously.

2.5 e

Since the minimum player is simply choosing between the two with a 50% chance the max player assigns a value of the two choices times .5 added together for the value of a node due to the min player's choice.



Alpha-beta pruning cannot be applied because it is impossible to know which node the opponent will choose and that randomness means that the opponent is no longer

playing optimally which also breaks one of the assumptions of minimax search. Alpha is unclear when you cannot anticipate the moves the opponent will make with certainty.

The filled out tree also shows the problem with the opponent not playing optimally. We end up in a situation where there is a 50% chance of the score being 6 or 8 depending on which the min player chooses during their first choice based on what our tree shows MAX should take. There is however a 50% chance of a 83 or a 99 also. meanwhile we had 17 as the final value when the minimum player was playing optimally also. This gives the MAX player a better score 50% of the time but a worse score 50% of the time. The true optimal way to play in this situation would be to choose C and then M or N depending on whether MIN chooses F or G. For the tree to show that value as the best we would need to choose weights based on the difference between the two nodes MIN is choosing randomly from.

3 problem 3

3.1 a

The set of variables is as follows

1. $n_{i,j}$: possible domain is the integers 1 through 9
2. i: Row Variable, possible domain is integers 1 through 9
3. j: column variables, possible domain is integers 1 through 9
4. M: number of squares already filled. Possible domain is integer 12 through 80.

3.2 b

The MRV heuristic for backtracking search is better than degree heuristic because the empty spaces are most effectively filled when we assign the ones with the least possible choices first.

The branching factor of the search space is the number of nodes generated divided by M

The solution depth is hte length of the shortest path taken from the initial node to the goal node

The maximium depth of the search space is the length from teh root to the lowest leaf when using depth first search.

The size of the state space is 9^{81-m} due to there being 81-m open positions each of which have 9 possible values on any given map.

3.3 c

The biggest difference between easy and hard sudoku problems is the number of squares already filled(which we represent with M). A larger M makes the problem easier to solve due to reducing the number of possible states which reduces the number of times an invalid assignment can be given. This reduces the amount of backtracking needed due to reducing the states which need to be looked at. More squares being filled also makes

the Sudoku easier because it reduces the number of integers which can legally be assigned to each unfilled square due to more squares being filled increasing the amount of the numbers already present in each row, column, and square.

3.4 d

```
Sudokuassign
        filledCells = List of all squares which were given to us fillled
        mapX = Fill all empty cells randomly
        for i = 0 to 8
            for j = 0 to 8
                if [i][j] is not in filled cells and [i][j] has invalid assignment based on rest
of squares
                    for k = 1 to 9
                    oldscore = evaluate(mapX)
                    old = mapX[i][j]
                    mapX[i][j] = k
                    newscore = evaluate(mapX)
                    if newscore is 81
                        return mapX
                    if oldScore > newScore
                        mapX[i][j] = old
                    else
                      flag = true
            end j for
        end i for
        rerun Sudokuassign
```

Goes through each square that was randomly assigned and currently has an issue with its assignment and tries to see if the issue can be gotten rid of by changing it to any other possible value. Returns map if map is completely valid. If new value has less conflicts it keeps newValue otherwise it stays with the old value. If it was unable to fix the issue after reassigning every randomly filled square to each number it goes back to the beginning of it and tries with a new random generation.

This algorithm will definitely perform worse on easy problems because it randomly assigns the ones which only have one possible value instead of giving them that specific value if there is only one possible choice. As the problems become harder however the gap in performance becomes less due to backtracking having more squares it will likely have to backtrack to since less are certain. Still we believe that the local search algorithm we have created will be less effective on average for any problems due to the nature of having so many values and the issues caused by one cell's incorrect assignment meaning another cell will think its proper assignment is incorrect often.

4 problem 4

4.1 a

Beat Superman $\iff$ (Superman Alone $\wedge$ Have Kryptonite)
Have Kryptonite $\iff$ Ally with Luthor
Ally With Luthor $\rightarrow \neg$ Superman Alone

4.2 b

($\neg$ Beat Superman $\vee$ Superman Alone) $\wedge$ ($\neg$ Beat Superman $\vee$ Have Kryptonite) $\wedge$ (Beat Superman $\vee\neg$ Superman Alone $\vee\neg$ Have Kryptonite)
(Ally With Luthor $\vee\neg$ Have Kryptonite) $\wedge$ ($\neg$ Ally with Luthor $\vee$ Have Kryptonite)
$\neg$ Ally With Luthor $\vee\neg$ Superman Alone
Is 3-CNF since each clause has a maximum of 3 literals.

4.3 c

If we suppose Ally With Luthor is false Have Kryptonite is force to be false and therefore Beat Superman must also be false regardless of if Superman Alone is true or false.

In order to make Have Kryptonite true we must set Ally With Luthor to true. Setting Ally With Luthor to true also sets Superman Alone to false and Superman Alone being false forces Beat Superman to be false.

Therefore it is impossible for both Superman Alone and Have Kryptonite to be true therefore Beat Superman can never be true.

5 problem 5

6 problem 6

In general definition from the text book we say,

Consistent: if the estimated cost from node n to the goal is no greater than the step cost to its successor n' plus the estimated cost from the successor to the goal.

Admissible: if h(n) never overestimates the true cost to the goal state.

Let we have two heurisitics functions for a specific problem h_1 and h_2 and define $h^e(n)$ to be the true cost of the path from that node to the goal,

6.1 a

If we use the minimum value of $h_1(n)$ and $h_2(n)$ at every state and assume at least one of $h_1(n)$ and $h_2(n)$ is admissible and consistent,

$h_3(n)$ = minimum($h_1(n)$, $h_2(n)$) is admissible and consistent, since given that $h_1(n) \leq h^e(n)$ and/or $h_2(n) \leq h^e(n)$ it is always true that min($h_1(n)$, ∞) $\leq h^e(n)$ and/or min(∞ , $h_2(n)$) $\leq h^e(n)$.

if at least one of h_1 and h_2 is admissible but neither are consistent then the minimum will be admissible but not consistent

6.2 b

If we use the maximum value of $h_1(n)$ and $h_2(n)$ at every state and assume both are admissible and consistent,

$h_4(n)$ = maximum($h_1(n)$, $h_2(n)$) is admissible and consistent, since given that $h_1(n) \leq h^e(n)$ and $h_2(n) \leq h^e(n)$ we deduce that max($h_1(n)$, $h_2(n)$) $\leq h^e(n)$.

If either of $h_1(n)$ or $h_2(n)$ is not admissible and consistent then max($h_1(n)$, $h_2(n)$) $\leq h^e(n)$ will no longer be true and h_4 becomes inadmissible.

If either of $h_1(n)$ or $h_2(n)$ is not consistent but both are admissible then h_4 is admissible but not consistent.

6.3 c

for the heuristic function $h_3(n)$ = w * $h_1(n)$ + (1-w) * $h_2(n)$, where $0 \leq$ w ≤ 1,

With w = 0, $h_3(n)$ = 1 * $h_2(n)$ due to removing 0 * $h_1(n)$ from the equation and therefore $h_3(n)$ has the same properties of admissibility and consistentency as $h_2(n)$*does*.

With w = 1, $h_3(n)$ = 1 * $h_1(n)$ due to removing 0 * $h_2(n)$ from the equation and therefore $h_3(n)$ has the same properties of admissibility and consistentency as $h_1(n)$*does*.

With $0 <$ w < 1 we have $h_3(n)$ = w * $h_1(n)$ + (1-w) * $h_2(n)$ instead of being able to remove a factor. This means $h_3(n)$ is admissible and consistent only if both h_1 and h_2 are admissible and consistent. If either of $h_1(n)$ or $h_2(n)$ is only admissible but not consistent then $h_3(n)$ is only admissible. If either of $h_1(n)$ or $h_2(n)$ is inadmissible then $h_3(n)$ is inadmissible. This is proven by the formula below.

Assuming $0 <$ w < 1 then $h^e(n)$ = w * $h^e(n)$ + (1-w) * $h^e(n)$. Therefore if we use the same w and $h_1(n) \leq h^e(n)$ is true(which happens when $h_1(n)$ is admissible) we can times both sides by weight w to get w * $h_1(n) \leq$ w * $h^e(n)$. If $h_2(n) \leq h^e(n)$ is true(which happens when $h_2(n)$ is admissible) we can times both sides by 1 minus weight w to get (1-w) * $h_2(n) \leq$ (1 -w) * $h^e(n)$. Adding these together gives us w * $h_1(n)$ + (1-w) * $h_2(n) \leq$ w * $h^e(n)$ + (1-w) * $h^e(n)$. We already know that $h^e(n)$ = w * $h^e(n)$ + (1-w) * $h^e(n)$ so we can see that * $h_1(n)$ + (1-w) * $h_2(n) \leq h^e(n)$ is also true. However as soon as one of either $h_1(n) \leq h^e(n)$ or $h_2(n) \leq h^e(n)$ becomes false(meaning one of the two heuristics is inadmissible) we are no longer able to do that showing that $h_3(n)$ is no longer admissible.

6.4 d

for the objective function, f(n) = (2-w) g(n) + w h(n), The algorithm is to be guaranteed for $0 \leq$ w ≤ 1 , since multiplied by a constant in g(n) implies no effect on the relative ordering of chosen paths, but if w > 1 then possibly it may overestimates the distance to a goal, and heuristics made to be inadmissible. If w ≤ 1 the estimate reduces and is guaranteed to underestimate the distance to a goal.

For the above function,

When w =0; it gives f(n) = 2 * g(n) which behaves exactly like Uniform Cost search or Uninformed Best first search since the factor of 2 on every cost makes no difference in the ordering of the nodes.

When w = 1; it gives f(n) = g(n) + h(n) is A* search and guaranteed to find an optimal solution.

When w =2; it gives f(n) = 2 * h(n) which behaves exactly like Greedy Best first search since the factor of 2 on every cost makes no difference in the ordering of the node.