Problem 3: Search (35 points)

Figure 2 is a graph to be searched, starting at A and ending at G. The h values are the heuristic estimates, and the numbers on the edges are the actual costs. Assume that the children of a node are ordered in alphabetical order; also use the alphabetical order to break ties, if necessary.

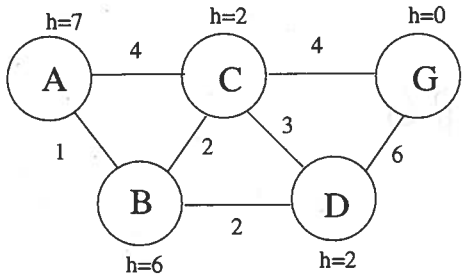


Figure 2: *Search graph.*

Part A (20 points).

A1 (15 points).
Perform an A^* search with an expanded list. Fill in the following table. Which path is returned?

Expanded Node	Partial Path leading to the node	Total estimated cost using partial path
A	A	7
C	AC	6
B	AB	7
D	ABD	5
G	ACG	8

Path returned:

ACG

A2 (5 points).
Did A^* search with expanded list return the optimal path? Explain why in terms of the admissibility and/or consistency of the heuristics.

no. node C's heuristic score B not consistent.

Part B (5 points).

B1 (4 points).
Perform a depth-first search, without using any visited or expanded lists. Show the sequence of expanded nodes. Which path is returned?

ABCDG

B2 (1 points).
Path returned:

ABCDG

Part C (5 points).

C1 (4 points).
Perform a uniform cost search, using an expanded list. Show the sequence of expanded notes. Which path is returned?

ABCDG

C2 (1 points).
Path returned:

ABCG

Part D (5 points).

D1 (4 points).
Of the search algorithms covered in class, which one requires the smallest number of expansions before returning a path? Which path is returned?

Best first search.

D2 (1 points).
Path returned:

A C G.

Problem 4: Game Search (22 points)

Part A (6 points).

You are watching two people play a game called Mini-Four, which is just like Connect Four, but played on a smaller (4x4) board. In this game, the black player and white player alternate dropping a piece into one of the four columns; the piece falls into the lowest row not already occupied. If a player can get four pieces of the same color in a row, he wins.

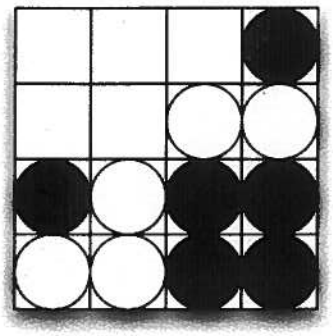


Figure 3: The in-progress game.

A1 (1 points).
The game has already progressed to the point seen in Figure 3. Black went first, and so it is now white's turn to move. You notice that white has a move that guarantees a win. Determine and circle the move white should make to guarantee victory.

Column 1 (Column 2) Column 3 Column 4

A2 (5 points).
Mini-Four is a very small game, and so easy to exhaustively search. As the size of the board gets bigger, even just to the standard 6x7, this becomes tough. For Minimax or α-β to work on these big games, we must stop early and use a static evaluator on non-final board states, as you just did in the last part.

We want you to design a static evaluator for these Mini-Four-like games. That is, you should give a function or algorithm that takes in a board state (the location and color of the pieces) and gives back a number. The number should be large when the board is good for the current player, and small when it's bad. If it makes it easier for you to write, you may assume the current player is white.

Please remember that the static evaluator should only consider the current state. You don't need to look into the future; that's what Minimax does for you!

(this answer depends on what creative, but correct, ideas you have)

Strategy:
here are some features you might incorporate into your function:

- how many un-blocked runs of length {2,3} do you have
- how many un-blocked runs of length {2,3} does your opponent have
- is the state a winning configuration for {you, opponent}
- how many open spaces are adjacent to {your, opponent's} pieces

Part B (16 points).
You write two programs to play Mini-Four, one using Minimax and one using α-β from left to right. We provide some arbitrary game search tree they're both trying to evaluate in Figures 4 and 5. The leaves are labeled with their static evaluation values.

B1 (7 points).
Use Minimax to evaluate the game tree as a maximizer. Fill in the blanks in Figure 4 with each node's Minimax value.

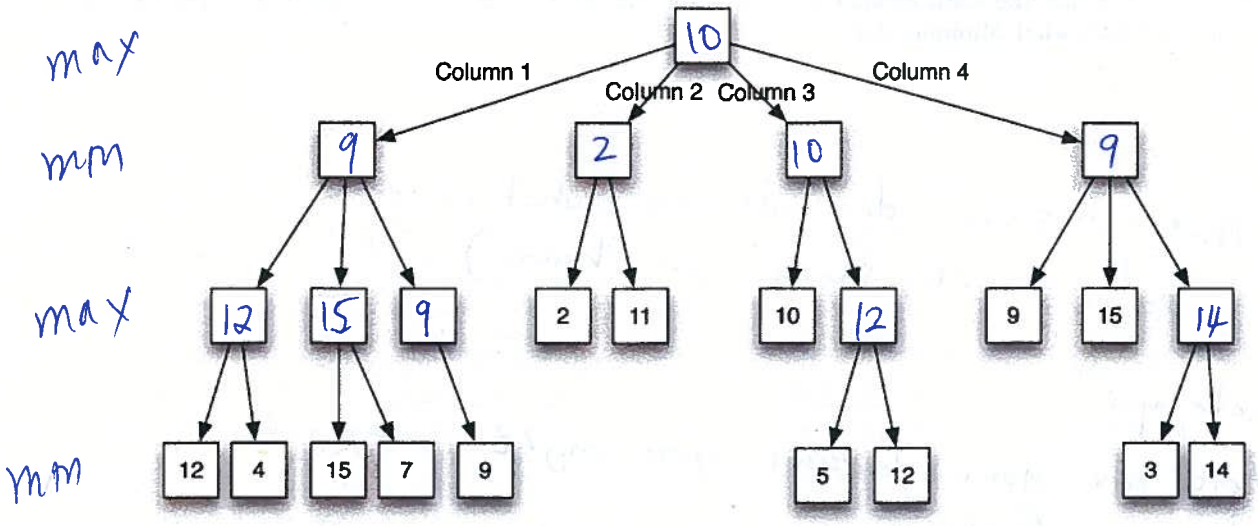


Figure 4: The Minimax search tree.

B2 (1 points).
What move does Minimax say the maximizer should make?

Column 1 Column 2 Column 3 Column 4

B3 (7 points).
Use α-β to evaluate the game tree as a maximizer. Work left to right. Cross out nodes that α-β does not evaluate (both the blank ones and the ones we've given values for). Fill in the blanks in Figure 4 with each node's Minimax value.

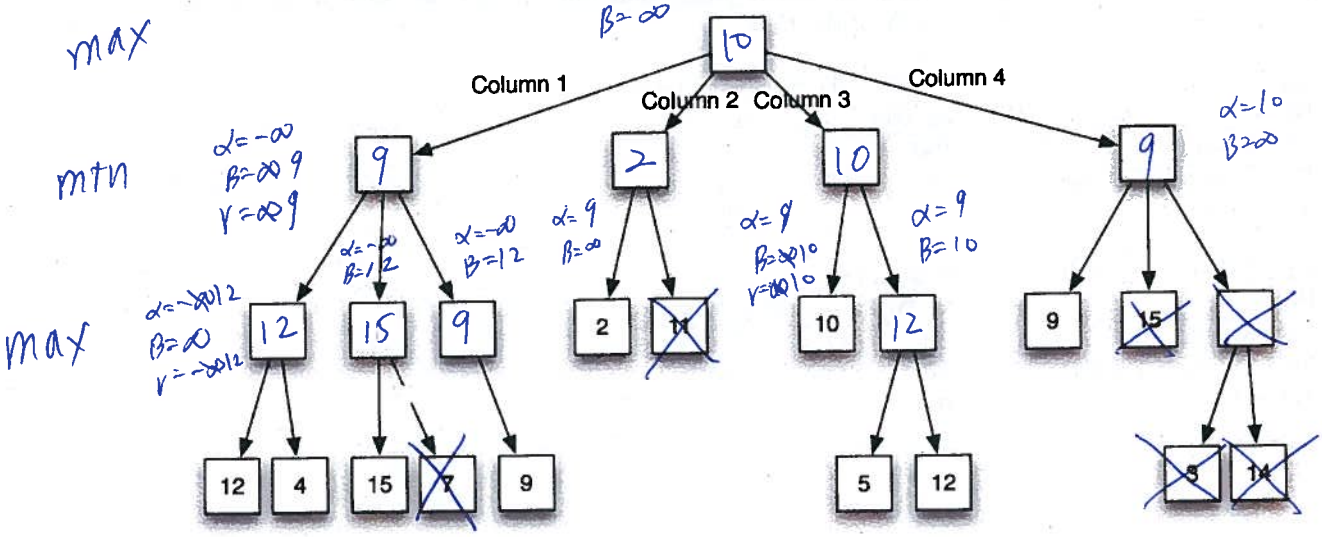


Figure 5: The α-β search tree.

B4 (1 points).
What move does α-β say the maximizer should make?

Column 1 Column 2 Column 3 Column 4

Problem 6: Constraint Satisfaction (18 points)

Misters White, Orange, Pink, and Brown are planning a crime while sitting in a small diner. 4 chairs labeled 1,2,3 and 4 fit around the diner's table.

You will model where each person sits as a constraint satisfaction problem. The variables will be the diners, {(W)hite, (O)range, (P)ink, (B)rown}. The domain for each is $\{1, 2, 3, 4\}$. Chair pairs 1 and 2, 2 and 3, 3 and 4, and 4 and 1 are adjacent to each other. Chairs 1 and 3 are across from each other, as are chairs 2 and 4.

The criminal masterminds are petty and finicky, and this puts constraints on where they can sit.

1. No two people can sit in the same chair.
2. Mr. Pink will only sit in chair 3.
3. Mr. White will not sit across from Mr. Pink.
4. Mr. Orange will to sit adjacent to Mr. Pink.
5. Mr. Brown will only sit across from Mr. White.

Part A (4 points).
Using the constraints, perform full constraint propagation on this seating problem starting with domains of $\{1, 2, 3, 4\}$ for each variable B, P, W, and O. List the domains of each variable after full constraint propagation.

- B = {2,4}
- P = {3}
- W = {2,4}
- O = {2,4}

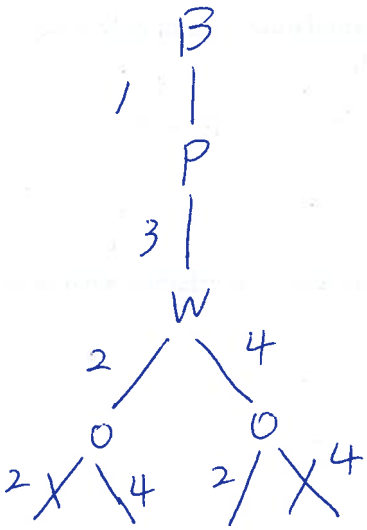
Part B (2 points).
Does full constraint propagation find a solution to this problem? Is there a solution to this problem? Explain.

No, because there are only 2 valid values for 3 variables {B, W, O}
{2,4}

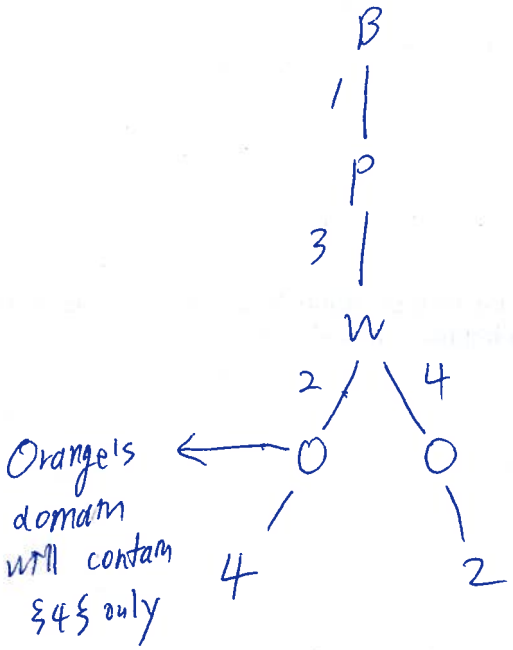
Part C (4 points).
Mr. Brown now will only sit adjacent to Mr. White instead of across from him. After full constraint propagation, now what are the domains for each person?

- B = {1}
- P = {3}
- W = {2,4}
- O = {2,4}

Part D (4 points).
Using the variable order B, P, W, O, and starting with the domains from the above full constraint propation, show the sequence of variable assignments during a pure backtracking search. Assign chair values in numerical order.



Part E (4 points).
Do the same as the previous problem, except this time show the sequence of variable assignments during backtracking *with forward checking.*



Problem 7: Search and CSP (27 points)

Part A (15 points).

Consider the following search problem formulation with finitely many states:
States: there are $d+2$ states: $\{S_s, s_g\} \cup \{s_1, \ldots, s_d\}$
Initial state: s_s
Successor function: `Succ (s)` generates at most b successors
Goal test: s_g is the only goal state
Step cost: each step has a cost of 1

A1 (2 points).
Suppose an optimal solution has cost n. If the goal is reachable, what is the upper bound on n?

$n \leq d+1$

A2 (2 points).
Suppose we must solve this search problem using Breadth-First Search, but with limited memory. Specifically, assume we can only store k states during search. Give a bound on n for which the search will fit in the available memory.

$n \leq \log_b k$

A3 (2 points).
Would any other search procedure allow problems with substantially deeper solutions to be solved? Either argue why not, or give a method along with an improved bound on n.

$n \leq \frac{k}{b-1}$

A4 (5 points).
If we knew the exact value of n, we could formulate a CSP whose complete assignment specifies an optimal solution path $(X_0, X_1, \ldots X_n)$ for this search problem. State binary and unary constraints which guarantee that a satisfying assignment is a valid solution:
Variables: $X_0, X_1, \ldots, X_n$
Domains: $Dom(X_i) = \{s_s, s_g\} \cup \{s_1, \ldots, s_d\}$
Constraints:

① $X_0 = S_s$ ③ Alldiff (not a binary constraint though, but good to kn

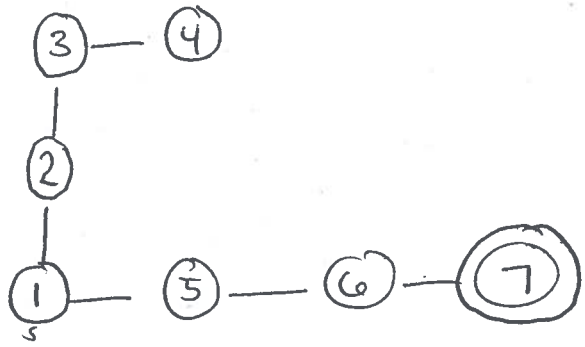
② $X_n = S_g$ ④ $X_i \in$ Successors (X_{i-1})

$\Leftrightarrow$

$X_{i-1} \in$ parent (X_i)

A5 (4 points).
After reducing the domains of any variables with unary constraints, suppose we then make all arcs $X_i \rightarrow X_{i-1}$ consistent, processed in order from $i = 1$ to n. Next, we try to assign variables in reverse order, from X_n to X_0, using backtracking DFS. Why is this a particularly good variable ordering?

Consider:



Start: ①
goal: ⑦
n : 3

unary constraints

Step 1

X_1	X_2	X_3	X_4
{1}	{5,2}	{6,3}	{7}

are consistency

Step 2

Exploring backwards guarantees we don't waste time exploring this path ①–②–③ first... we go directly from ⑦ → ⑥ → ③ → ①

Part B (12 points).
Each problem is worth 2 points. Provide a short explanation that justifies your answer. (Without such explanation, you will not get credit for your answers.)

B1 (2 points).
If a search method is guaranteed to find an optimal solution on trees, then that method is also guaranteed find an optimal solution when applied to general graphs.

No, A* search with admissible/non-consistent heurist scores is optimal on tree but not on graph.

B2 (2 points).
An optimal solution path for a search problem with positive costs will never have repeated states.

Yes, otherwise the solution without repeated states will be the optimal solution

B3 (2 points).
Admissible heuristics are always more effective than inadmissible ones.

yes, ~~##~~ inadmissible heuristics may not generate the optimal solution, so ~~they~~ they're not effective.

B4 (2 points).
If one search heuristic $h_1(s)$ is admissable and another one $h_2(s)$ is inadmissable. then $h_3(s) = min(h_1(s), h_2(s))$ will be admissable.

yes, let actual cost from s to G be $H(s)$. $h_1(s) \leq H(s)$ ~~##~~
$h_2(s) > H(s)$ $\therefore h_1(s) < h_2(s)$ $\therefore h_3(s) = min(h_1(s), h_2(s)) = h_1(s)$
$\therefore h_3(s)$ is admissible

B5 (2 points).
Alpha-beta pruning can alter the computed minimax value of the root of a game tree.

No. Alpha-beta pruning will never change the solution.

B6 (2 points).
When doing alpha-beta pruning on a game tree which is traversed from left to right, the leftmost branch will never be pruned.

Yes, you always need to see what a node has got before you can prune.

THE END!