SAARLAND UNIVERSITY ARTIFICIAL INTELLIGENCE
Prof. Jörg Hoffmann and Prof. Wolfgang Wahlster
Dr. Álvaro Torralba, Daniel Gnad, Marcel Steinmetz
Christian Bohnenberger, Cosmina Croitoru, Akram Elkorashy, Sophian Guidara,
Daniel Heller, Björn Mathis, Lukas Schaal, Julia Wichlacz

Exercise Sheet 2.

Solutions due Tuesday, **May 17**, 16:00 – 16:15, in the lecture hall.[1]

Exercise 4. (2.5 Points)

Consider the following variant of Tic-tac-toe, in which the max player "X" wins (with an utility of 100) if, at the end of the game there is no 3-line of "X" or "O", and otherwise the min player "O" wins (with an utility of -100).

	O	
X	X	O
O	X	

Figure 1: Starting state of the Minimax search. Player "X" is to move next.

(a) Draw the full Minimax tree starting from the state depicted in Figure 1. Annotate every node with its utility. Who wins the game from this position? What move should the starting player "X" perform in this position?

(b) Consider the evaluation function $f(x)$: number of rows, columns, and diagonals that do not contain at least two "X" or two "O". For example, in the starting state $f(x) = 6$. Note that this is equivalent to 8 - number of rows, columns, and diagonals that do contain at least two "X" or two "O". Run Minimax with a depth of 2 and this evaluation function. Draw the tree and annotate every node with its utility.

(Solution)

(a) The search tree is the following. "X" wins the game by playing in the upper right corner.

[1] Solutions in paper form only, and solution submission only at the stated time at the stated place. At most 3 authors per solution. All authors must be in the same tutorial group. All sheets of your solution must be stapled together. At the top of the first sheet, you must write the names of the authors and the name of your tutor. Your solution must be placed into the correct box for your tutorial group. If you don't comply with these rules, 3 points will be subtracted from your score for this sheet.

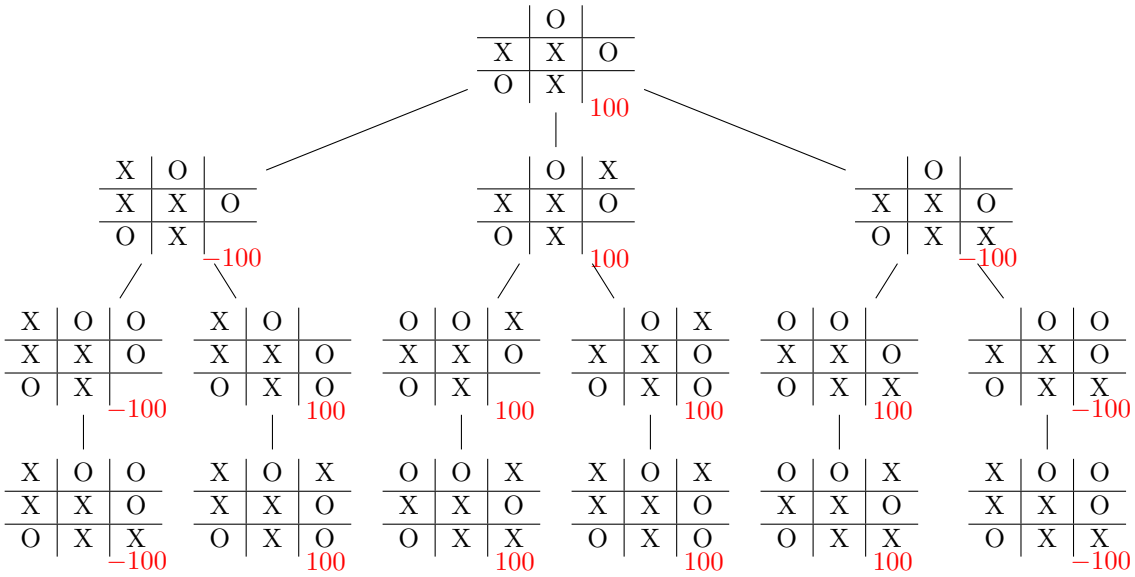


Figure 2: Min-max search tree.

(b) The evaluation function is good since it correctly assigns most points to positions after playing the winning move.

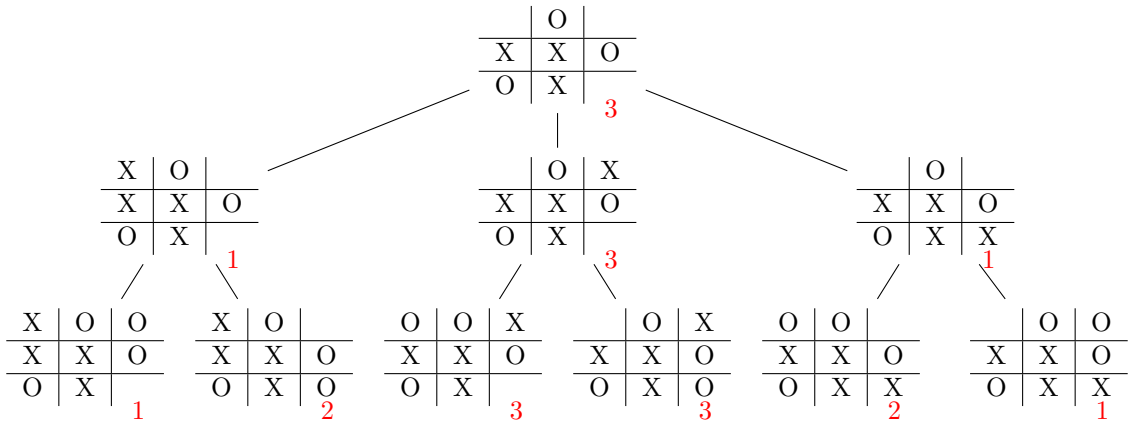
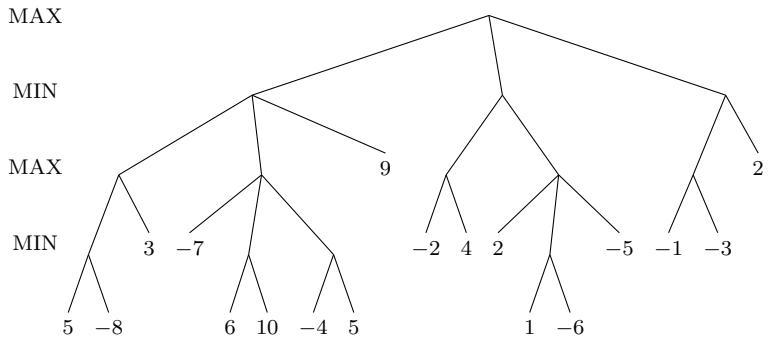


Figure 3: Min-max search tree with the evaluation function and $d = 2$.

(/Solution)

Exercise 5. (3.5 Points)

Consider the following game tree corresponding to a two-player zero-sum game as specified in the lecture. As usual, the Max-player is to start in the initial state (i.e., the root of the tree). For the following algorithms, the expansion order is form left to right, i.e., in each node the left-most branch is expanded first.



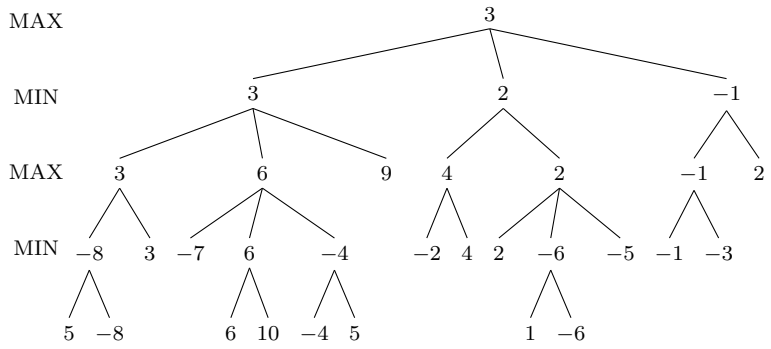
(a) Perform Minimax search, i.e., annotate all internal nodes with the correct Minimax value.

(b) Perform Alpha-Beta search. For this, annotate all internal nodes with the value that will be propagated to the parent node as well as the final $[\alpha, \beta]$ window before propagating the value to the parent (similar to slides 38 and 39 in Chapter 6). Mark which edges will be pruned. How many leaf nodes are pruned?

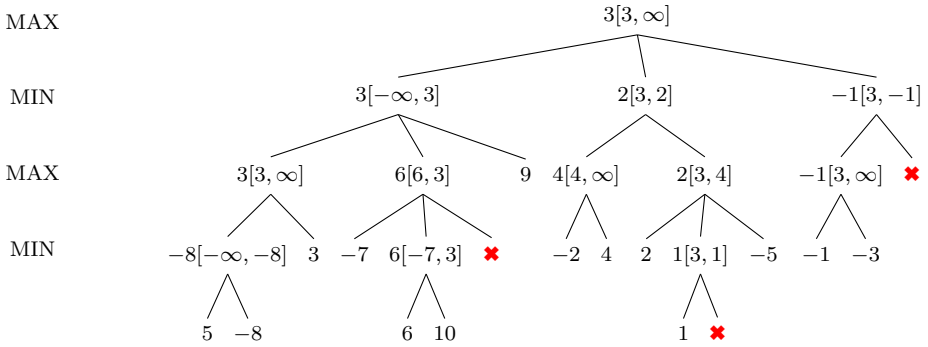
(c) There are better move orderings yielding more effective pruning. Such a changed move ordering corresponds to a changed order of the successors in the tree (compare slide 41 in Chapter 6), and can be applied in any node in any depth of the tree. Find one where 7 leaf nodes are pruned. Give the resulting game tree (along with the propagated values and the $[\alpha, \beta]$ window, same as before).

(Solution)

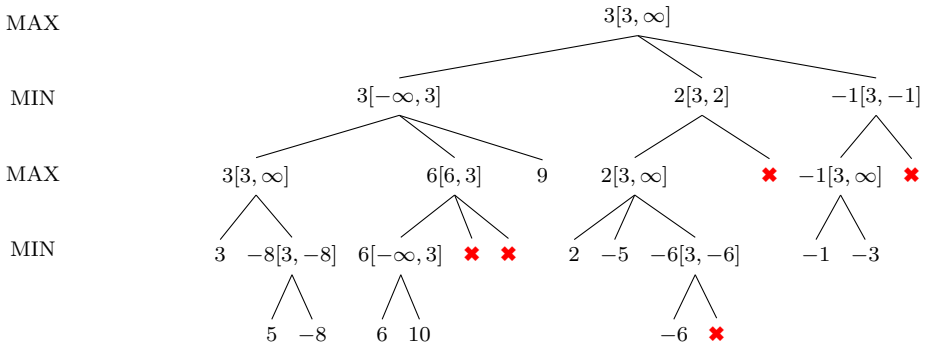
(a) The Minimax tree:



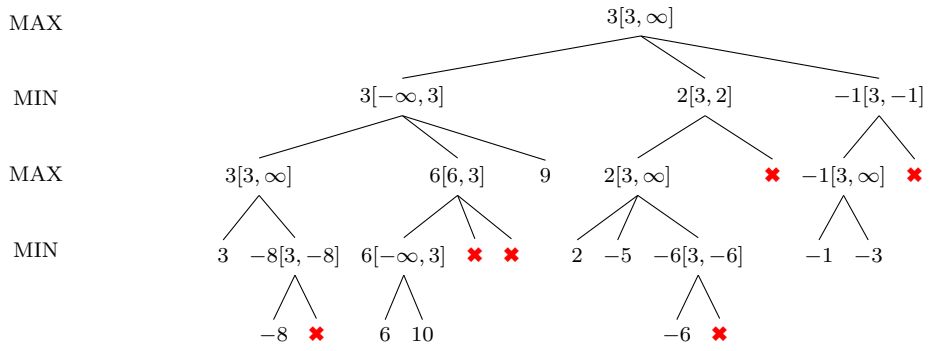
(b) The Alpha-Beta tree (4 leaf nodes are pruned):



(c) The Alpha-Beta tree with 7 pruned leaf nodes:



Maximal possible number of pruned leaf nodes: The Alpha-Beta tree with 8 pruned leaf nodes:



The basic idea of this tree is as follows: First of all, have the optimal goal value on the left-most branch. Then, order the children of the Min-nodes in such a way that their Minimax-values are in increasing order; for the Max-nodes order them so that their Minimax-values are in decreasing order.

(/Solution)

Exercise 6. (2 Points)

Given the following map coloring problem, run the naïve backtracking algorithm to find a coloring such that all adjacent administrative districts are painted in a different color. The corresponding constraint satisfaction problem $\gamma = (V, D, C)$ has a variable $v \in V := \{\text{HOM,MZG,NK,SB,SLS,WND}\}$ for each of the districts, each with domain $D = \{\text{red,green,blue}\}$. The constraints are such that no pair of adjacent districts can have the same color, i.e., $C_{\{u,v\}} \in C$, for all adjacent u, v, meaning that "$u \neq v$". For example, $C_{\{\text{MZG,WND}\}} \in C$, because MZG and WND are adjacent and consequently need to have a different color, but $C_{\{\text{MZG,HOM}\}} \notin C$.

(a) Run the naïve backtracking algorithm using the combination strategy described on the bottom of slide 42 of chapter 7, i.e., among the most constrained variables, pick the most constraining first. If this does still not result in a unique variable choice, use the alphabetical order on the district labels. The value order should first assign red, then green, and eventually blue. Draw the search graph and mark inconsistent assignments.

(b) When using the same value ordering, (red → green → blue), define a variable ordering such that the search graph contains more nodes than the one in the previous part. Explain why more nodes are generated.

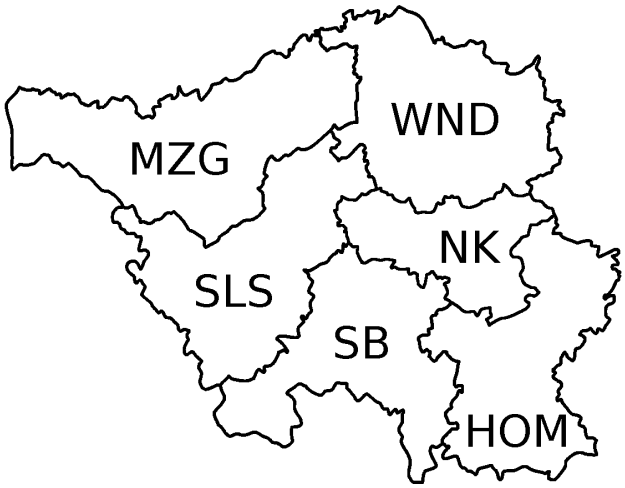


Figure 4: A map of Saarland

(Solution)

1. See figure 5
2. There are many orders that will result in a larger search graph. One example is the alphabetical ordering on the district labels, which in the first two steps assigns red to both HOM and MZG. Since no solution exists in which these two have the same color, but the algorithm does not see this immediately, it creates many partial assignments that cannot be extended to a solution. In particular, at depth level 2 in the search graph, naïve backtracking will have to generate the complete sub-trees of two partial assignments, where with the ordering from the exercise, only one sub-tree needs to be generated.

(/Solution)

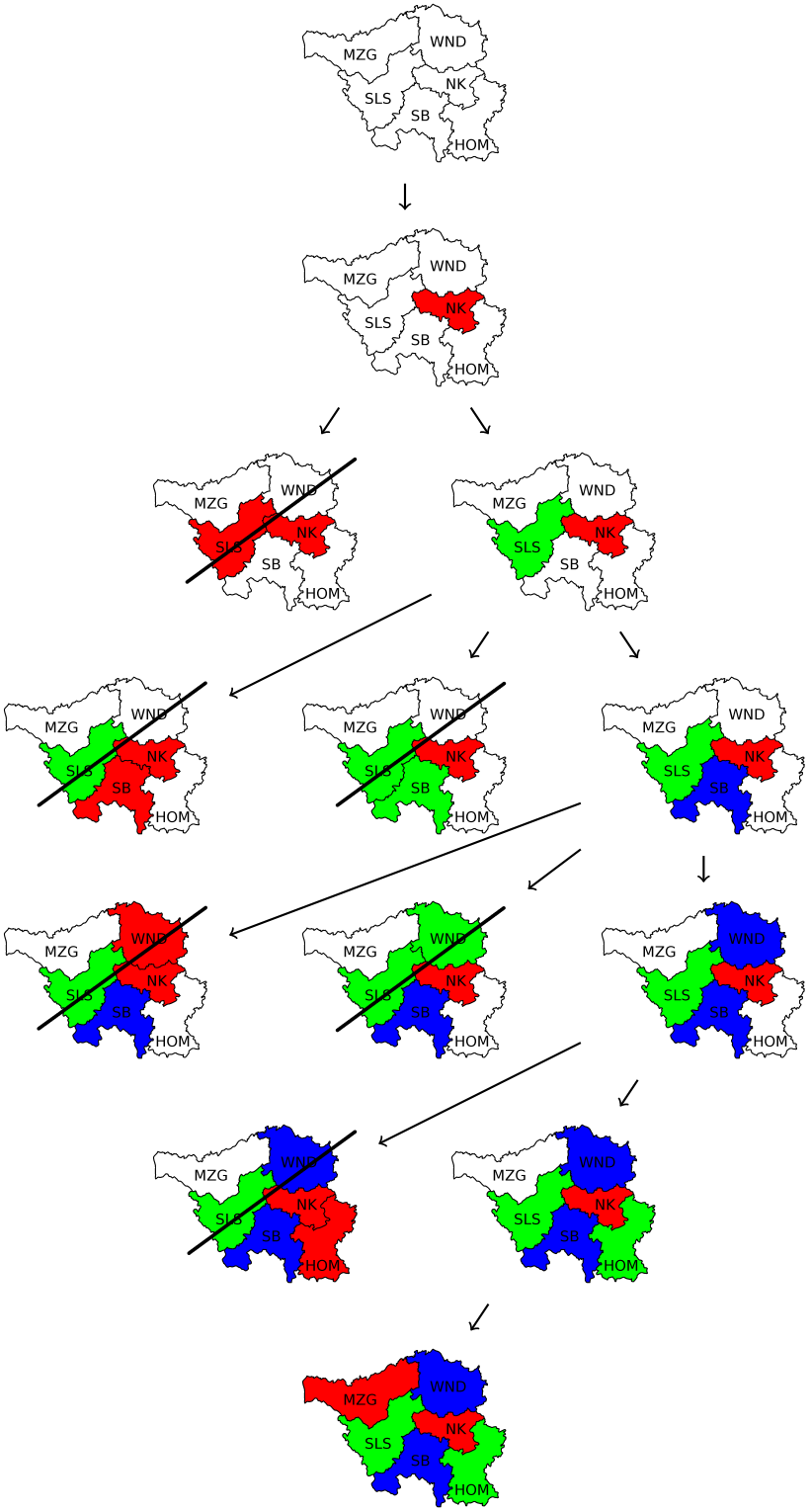


Figure 5: Solution of Exercise 6(a)

Exercise 7. (2 Points)

Consider the crossword puzzle in Figure 6. Even numbers stand for words to be inserted vertically, odd numbers stand for words to be inserted horizontally. Each word can be added only once.

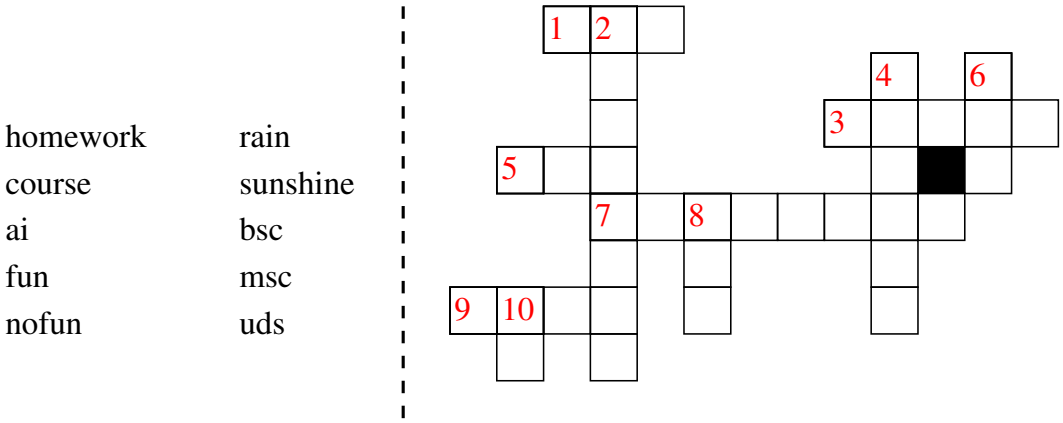


Figure 6: Crossword puzzle to be formalized in Exercise 7.

Formulate this puzzle as a constraint network whose solutions are the solutions to the puzzle. Use the set of variables $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$. Each variable domain will be a subset of the words to be filled into the puzzle.

(a) Specify the variable domains, i.e., give D_{v_i}, for each variable $v_i \in V$.

(b) Specify the constraints, i.e., give $C_{v_i v_j} \subseteq D_{v_i} \times D_{v_j}$, for all pairs of variables where the set of feasible value pairs is not equal to the entire set $D_{v_i} \times D_{v_j}$ of such pairs.

(Solution)

(a) It suffices to include, into the domain of each variable, the subset of words that have the correct length. Hence we get:

- $D_{v_1} = \{\text{fun, bsc, msc, uds}\}$
- $D_{v_2} = \{\text{homework, sunshine}\}$
- $D_{v_3} = \{\text{nofun}\}$
- $D_{v_4} = \{\text{course}\}$

- $D_{v_5} = \{\text{fun, bsc, msc, uds}\}$
- $D_{v_6} = \{\text{fun, bsc, msc, uds}\}$
- $D_{v_7} = \{\text{homework, sunshine}\}$
- $D_{v_8} = \{\text{fun, bsc, msc, uds}\}$
- $D_{v_9} = \{\text{rain}\}$
- $D_{v_{10}} = \{\text{ai}\}$

(b) The variable pairs with non-trivial constraints (feasible value pairs not equal to the entire set $D_{v_i} \times D_{v_j}$ of such pairs) are exactly those pairs of variables whose word-placeholders intersect, and not all combinations of words are compatible: The pairs of feasible values are exactly those where the intersecting fields have the same letters. Hence we get the constraints:

- $C_{v_1 v_2} = \{\text{(bsc,sunshine), (msc,sunshine)}\}$
- $C_{v_1 v_5} = \{\text{(fun, bsc), (fun, msc), (fun, uds), (bsc, fun), (bsc, msc), (bsc, uds), (msc, fun), (msc, bsc), (msc, uds), (uds, fun), (uds, bsc), (uds, msc)}\}$
- $C_{v_1 v_6} = \{\text{(fun, bsc), (fun, msc), (fun, uds), (bsc, fun), (bsc, msc), (bsc, uds), (msc, fun), (msc, bsc), (msc, uds), (uds, fun), (uds, bsc), (uds, msc)}\}$
- $C_{v_1 v_8} = \{\text{(fun, bsc), (fun, msc), (fun, uds), (bsc, fun), (bsc, msc), (bsc, uds), (msc, fun), (msc, bsc), (msc, uds), (uds, fun), (uds, bsc), (uds, msc)}\}$
- $C_{v_2 v_5} = \{\text{(sunshine,uds)}\}$
- $C_{v_2 v_7} = \{\text{(sunshine, homework)}\}$
- $C_{v_2 v_9} = \{\text{(sunshine,rain)}\}$
- $C_{v_3 v_6} = \{\text{(nofun,fun)}\}$
- $C_{v_4 v_7} = \{\text{(course,homework)}\}$
- $C_{v_5 v_6} = \{\text{(fun, bsc), (fun, msc), (fun, uds), (bsc, fun), (bsc, msc), (bsc, uds), (msc, fun), (msc, bsc), (msc, uds), (uds, fun), (uds, bsc), (uds, msc)}\}$
- $C_{v_5 v_8} = \{\text{(fun, bsc), (fun, msc), (fun, uds), (bsc, fun), (bsc, msc), (bsc, uds), (msc, fun), (msc, bsc), (msc, uds), (uds, fun), (uds, bsc), (uds, msc)}\}$
- $C_{v_6 v_8} = \{\text{(fun, bsc), (fun, msc), (fun, uds), (bsc, fun), (bsc, msc), (bsc, uds), (msc, fun), (msc, bsc), (msc, uds), (uds, fun), (uds, bsc), (uds, msc)}\}$
- $C_{v_7 v_8} = \{\text{(homework,msc)}\}$

Note: The word-placeholders of v_3 and v_4 intersect, but each has only a single possible value (word to be filled in), and these values are compatible. Same for v_9 and v_{10}.

(/Solution)