CS404-Assignment2

Alper Kaan Odabaşoğlu

March 2023

1 CSP Model

As we discussed in the lecture, CSPs can be represented as a vector of feature values. There are variables, corresponding domain of values for variables and constraints. The goal is to seek for the configuration of values for the variables so that all of the constraints are satisfied. As it was specified in the slant puzzle problem definition, there are mxn intersect points including numbers and "*" and there is a (m-1)x(n-1) grid, filling the gaps of intersect points and contains '\' or '/'. In the subsections, the variables, constraints, domain of values for variables and goal state will be specified.

For the ease of constraint definition, variables are determined double for each grid cell. 2 variable set is defined for each cell (left_vars and right_vars). So in total, if there are (m-1)(n-1) grid cell, there are 2(m-1)(n-1) variables exist.

The values that each variable can take is determined as 0 and 1. 1 for right_vars list means there exists a '\' and 0 means there is not '\' for that cell. 1 for left_vars means there exist a '/' for particular grid cell and vice-versa for 0.

There is 3 constraint specified for the problem. First constraint is only one of the left_vars variable or right_vars variable can equal to 1 and other should be equal to 0 for particular grid cell. In other words, only one of the '\' or '/' can be selected for a particular cell. Second constraint enforces that if there is a number at the intersection point, the number of '/' or ' \' entering this intersection point from the surrounding grid cells must be equal to that number. This constraint enforced by adding up the values of surrounding cell variables (left-right) and checking whether this sum equal to the number on the intersect point. Last constraint for the problem is loop constraint which emphasizes that the '\' and '/' symbols cannot form a loop on the grid which means that there shouldn't be an infinite path exist from one intersection point to another, or itself. This constraint is introduced as a post-check. First, grid that ensures the first 2 constraints extracted and converted to a graph using intersect points as a node and '\', '/' as an edge. On this extracted graph, cycle check is applied using DFS algorithm. Solutions passed the first 2 constraint, is eliminated with this check if it includes a loop.

The entire placement of '\' and '/' values on each grid cell (variable) without a gap that does not violate any constraint is a goal state. There can be multiple goal states for a particular solution.

2 Benchmarks

The benchmarks are the intersect point matrices specified in the assignment. Easy benchmarks are 3x3 intersect point matrix with 2x2 grid. Easy benchmarks include less than or equal to 10 (for each grid cell they cannot be equal and they should force the second condition for each cell) constraint condition added to the CP-SAT solver model and include less loop formation rate due to the fact that it contains lesser grids. Normal Instances are 5x5 intersect matrix with 4x4 grid. There is more chance to form a loop in normal instances. Compared to the easy instances there is more than 20 (even just enforcing all the grid cells include one of the value from domain gives 4x4=16 conditional constraint check) and less than 40 conditional constraint checks which increases the difficulty obviously. In the hard instances, 7x7 intersect matrix and corresponding 6x6 grid is given. There is obviously more combination of loop formation in hard instances. Furthermore, it has more than 40 conditional constraint checks which increases the difficulty even more. Below are the three examples for each difficulty class respectively (only the intersect matrix is given).

*	*	1
*	*	*
*	2	*

*	*	*	*	1
2	*	*	3	1
*	2	*	*	*
2	*	*	3	1
*	2	*	*	*

1	*	2	*	*	1	*
*	*	*	*	*	*	2
*	3	*	*	3	*	*
*	*	*	*	*	2	*
*	*	*	3	*	*	*
1	*	*	*	*	2	*
*	2	*	1	*	*	*

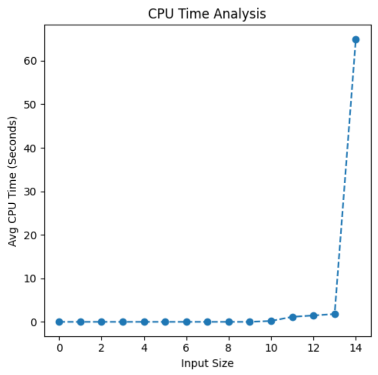
3 Modelling Decision

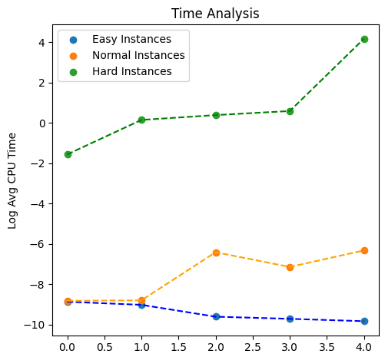
CSP is more appropriate for solving this problem than A* search since we are not trying to find an optimal path to goal state. Instead, we are just trying to extract the goal state with given constraints. Furthermore, it is hard to find a heuristic that can guess how far is the current configuration of slashes on grid cells is to the goal state. Furthermore, as we discussed in the lecture, A* search can be inefficient when there are a lot of solutions exist or the search space is large (n·n) which can be a case for Slant Puzzle. Since CSP models can apply backtrack search which is a depth-first strategy search method, it can be quite efficient on large search spaces. Last and more importantly, the natural structure of the Slant puzzle problem directly conform to the CSP since it includes constraints, variables and domains.

4 Scalability

Instance	Number of Variables	Number of Constraints	Backtrack Count	Avg Time(s)
Easy 1	8	8	0	0.00014
Easy 2	8	7	0	0.00012
Easy 3	8	5	0	0.00007
Easy 4	8	5	0	0.00006
Easy 5	8	9	0	0.00005
Normal 1	32	22	0	0.00015
Normal 2	32	20	0	0.00015
Normal 3	32	20	16	0.00163
Normal 4	32	21	8	0.00079
Normal 5	32	21	16	0.00179
Hard 1	72	48	1,164	0.20929
Hard 2	72	49	6,920	1.15613
Hard 3	72	45	8,768	1.47078
Hard 4	72	48	10,888	1.79873
Hard 5	72	46	393,216	64.97175

Table 1: Performance Metrics of the Sudoku Solver





Even though google ortools cp solver is a state of art solver, it cannot be scalable when the search space is getting very large. As we can see from Table 1, while the number of variables increasing linearly from easy to hard instances, backtrack counts and average time complexity increases rapidly. For instance, in instance 1 with easy difficulty level, there is 8 variable, 0 backtrack count (the solver finds directly) and averagely 1e-4 seconds taken, however in hard instance 5, while the variable count becomes 72 (9 times more), the backtrack count becomes 393216 and average time taken becomes 64.971 (4640785.7 times more). Just in the 7x7 grid, we can encounter such rapid increase. Thus, when the size approaching NxN (very large integers) the computational power cannot be enough to find the feasible solutions. Furthermore, when we look at the average time plots, in the first figure, when the instance difficulty increase towards easy to hard, we can observe insane jump in the scatters (especially in the last one). When we examine the second figure, we can see in logarithmic scale, there is jumps between difficulty levels. These jumps contextualize our notion on scalability discussed above.