CS404 Homework 2

Alperen Yıldız - Sabancı University FENS

April 2023

1 Model

1.1 Variable

The variable in this constraint satisfaction problem correspond to the cells which make up the game board itself since it is possible to assign them different values to indicate whether or not they do not hold any symbol, or a slash or a backslash. Two sets of variables are kept, one for representing backslashes and one for representing slashes. Therefore, each cell in the game board corresponds to two variables.

1.2 Domain

The domain of each variable is basically them being empty or holding a symbol. Thus, the domain is comprised of three two values, namely 0 indicating an empty cell, 1 indicating a slash or a backslash.

1.3 Constraints

If we look at this constraint satisfaction problem from a high level perspective there are only two constraints which are:

- Number of slates adjunct to intersections should be exactly equal to the number given in the intersection inside of a circle if there are any.
- No combination of slates should form a loop in the grid.
- Each cell in the game board must contain either a slash or a backslash

These constraints can be explored further in detail using mathematical specifications:

Suppose $(i_r, j_r) \subseteq RightSlantGrid(n, n)$ where RightSlantGrid denotes the grid of variables denoting the right slashes in the playing field, (i_r, j_r) is a variable and $n \times n$ is the size of the matrix where $i_r <= n$ and $j_r <= n$. Each variable denoted this way has a domain of {0,1} therefore $(i_r, j_r) \epsilon \{0, 1\}$

Next, suppose $(i_l, j_l) \subseteq LefttSlantGrid(n, n)$ where LeftSlantGrid which as opposed to RightSlantGrid is made up of variables denoting left slashes in the playing field , (i_l, j_l) is a variable and $n \times n$ is the size of the matrix where $i_l <= n$ and $j_l <= n$. Each variable denoted this way has a domain of {0,1}. Thus, $(i_l, j_l) \epsilon \{0, 1\}$

In order to represent numbers in the intersections of the game board, constraints are generated automatically. If a slash is added to the playing field within the coordinates of (x, y) where $(x, y) \epsilon n \times n$, it will effect the constraints concerning (x, y) and $(x + 1, y + 1)$. Conversely, if a backslash is added, then it is going to effect $(x, y + 1)$ and $(x + 1, y)$. Therefore the full constraint would be specified as:

Let c be a constraint where $0 \leq c \leq 4$ and let the coordinates of c be (x, y). Then the constraint can be mathematically formulated as:

$$c = RightSlantGrid(x, y+1) + RighSlantGrid(x+1, y) + LeftSlantGrid(x, y) + LeftSlantGrid(x+1, y+1)$$

Additionally, since every cell in the puzzle needs to be filled a new constraint is added:

For each i_{slant}, j_{slant} where $(i_{slant}, j_{slant}) \epsilon \; SlantGrid(n, n)$

Since two sets of variables make up the SlantMatrix, an additional constraint is added:

For every $var_{left} \; \epsilon \; LeftSlantGrid(n, n)$ and $var_{right} \; \epsilon \; RightSlantGrid(n, n)$, $var_{left} \; \neq \; var_{right}$ Since the domains of both kinds of variables is [0,1], this constraint also makes sure each cell in the game board is filled in with either a slash or a backslash.

The final constraint is that the solution does not contain any cycles. Which is checked as via post-processing the presented solutions using DFS cycle checking algorithm.

2 Implementation

Google OR-Tools CP-SAT with Python is used. The final constraint which is that the solution should not contain any cycles is done as a post-processing phase. The results are generated and fed into a Depth-First Search algorithm implementation that checks whether or not there is a cycle within the solution and the first solution that does not contain a cycle is returned as the solution.

3 Evaluation

Easy, medium and hard categories are generated with inferences about how many backtracks would be required during the solving process based on the size of the matrix. Puzzles such as this where there are solid constraints seem to be significantly more suited for CSP solvers than search algorithms, even the solution-optimal ones such as A* Search. The reason why is that during the backtracking, inferences allow the implementation to reduce the search space depending on constraints imposed upon the solver. On top of that, backtracking search keeps only one instance of the state and modifies it therefore the space complexity of the implementation reduces. Therefore, CSP is the more appropriate for solving this problem. The algorithm seems to be very scalable in accordance with the difficulty of the puzzle but can be made more scalable using parallel and concurrent programming.

Table 1: Performance Table Per Matrix

	easy1	easy2	easy3	easy4	easy5	medium1	medium2	medium3	medium4	medium5	hard1	hard2	hard3	hard4	hard5
backtracks	8	8	8	8	8	50	50	50	50	50	72	72	72	72	72
variables	8	8	8	8	8	50	50	50	50	50	72	72	72	72	72
constraints	2	3	4	3	2	11	10	12	10	10	11	16	14	14	13
time	0.0428	0.0106	0.0111	0.0112	0.0112	0.0394	0.1570	0.0236	0.0427	0.0427	21.6608	1.9140	8.4046	8.7799	8.6008