Killer Sudoku Solver

František Dostál

April 11, 2022

1 Problem Description

Killer Sudoku is a variant on classical Sudoku, where so called "cages" are added to the problem. Each cage has a target sum to which numbers in it's cells should add up. Furthermore all numbers in one cage should be different from each other. The rules as described here are sourced form CSPLibrary. The cages are usually continuous sets of sudoku cells, however since the rules are not particulary clear on this point, we will assume general set of cells to be a cage, the decision being left to the puzzle's author.
The numbers in cages have to differ, just like they have to differ in rows and classic sudoku brackets but they also have to add up to a pre-set sum.
The size of a cage should be smaller than classic sudoku bracket but again it's not enforced rule and we will grant agency in this matter to the author of the puzzle.
We will assume that the input sudoku can be non-standard, meaning that the square bracket can have arbitrary size $B \times B$, therefore the whole puzzle being of size $N \times N$ where $N = B^2$.
Some sources claim that one of the rules of Killer Sudoku is that no "hints" (pre-filled cells) are provided to the solver, others do not say, therefore we will treat this as more of convention than rule and provide a way to give the solver hints in our model.
Some Sudoku rules add constraint on main diagonals however as it is not used in CSP Library definition or official Sudoku rules we will not use this constraint.

2 Solution

To compare different aproaches we will compare three models, loosely based on each other, attempting improvement on each other: naive implementation, overconstrained implementation, channeling implementation.

2.1 Naive Implementation

This implementation simply uses general rules of the game as constraints. Each cell is assigned a cage it belongs to and may be assigned hint via proper arrays. This ensures no two cages share a cell.
The enforcement of difference rules is simply done via alldifferent global constraint.
Base denotes the size of the bracket.

2.2 Overconstrained Implementation

Basicaly a naive implementation with added constraints to the sum over rows, columns and brackets on top of the alldifferent constraints in hopes it will produces better result in conjunction with the summation rules over cages.

2.3 Channeling Implementation

This implementation add channeling constraints to the overconstrained implementation. Two arrays are added to the model to provide inversions to rows and columns in a hope that the added interaction will help prune the searchspace faster.

3 Experimental Results

3.1 Puzzle Generation

For tests on classical Sudoku was used the example puzzle provided on CSP Library page with some alternation to make it unsatisfiable for the negative test.
However for tests on Sudokus with larger base there had to be developed a program gen_sudoku.py generating these puzzles. It's using python Minizinc library (and the script SudokuGeneration.mzn) to obtain valid Sudoku puzzle, then generates cages over it, aggregates their sums and from this information generates *.dzn* file for the models.

3.1.1 Valid vs. Invalid generation

The cages are generated by randomly spreading "seeds" of pre-set number of cages, in the next step for each unclaimed cell we choose which of its neighbour's cages will claim it and interate until no cell is unclaimed.
This method of generation however poses a problem, because some cages may, contrary to the rules, contain in its cells some number more than once. This means the resulting puzzle will become unsolvable. Therefore we will call this method "invalid" and puzzles generated by it "invalid" although we cannot claim apriori that all puzzles produced by this method will be nessessarily unsolvable.
The valid method that is sure to produce valid and solvable puzzles was implemented much like the invalid method but during the spreading phase, if the diffence rule in a cage should be broken by adding a cell to a cage, a new cage is created for the offending cell. This cage may then spread in next iterations much like the original cages. Puzzles gebnerated by this method will be called valid puzzles and are, again, sure to be solvable.

3.2 Positive Comparative Test

In this test we will run two valid puzzles of $base = 3$ and $base = 4$ through the scripts and see if there are any significant differences. We will report statistics measured by Minizinc - running time and explored nodes. Because the fully unhinted puzzle failed to be solved in $base = 4$ the tests for this setup were conducted on hinted puzzles, starting at 80 hints from 256 possible going down until the solution could not be found within 5 minutes. In each category the same puzzle is used in run of all three models.

Name	Nodes	Time[s]
Naive	1	0.169
Overconstrained	1	0.140
Channeling	1	0.149

Table 1: Positive experiment with base=3

Name	Hints	Nodes	Time[s]
Naive	80	4	0.175
	70	6	0.187
	60	4	0.171
	55	5	0.185
	50	253	0.176
	45	28	0.193
	40	5775	0.446
	35	12682	0.516
Overconstrained	80	4	0.176
	70	6	0.185
	60	4	0.172
	55	5	0.191
	50	253	0.187
	45	28	0.189
	40	5775	0.331
	35	12682	0.818
Channeling	80	4	0.18
	70	6	0.18
	60	4	0.186
	55	5	0.182
	50	253	0.192
	45	28	0.179
	40	5775	0.443
	35	12682	0.762

Table 2: Experiment with base=4

3.3 Negative Comparative Test

In this test we will run two invalid puzzles of $base = 3$ and $base = 4$ through the scripts and see if there are any significant differences. Although we can be certain only that $base = 3$ puzzle will is unsolvable apriori we can have a reasonable confidence that even in the event of the test running longer than 5 minutes, the puzzle indeed was generated unsolvable especialy in comparison to the positive test.

Name	Nodes	Time[s]
Naive	1	0.169
Overconstrained	1	0.140
Channeling	1	0.149

Table 3: Experiment with base=3

Name	Hints	Nodes	Time[s]
Naive	80	0	0.153
	70	0	0.172
	60	0	0.16
	50	0	0.174
	40	0	0.155
	35	Unfinished	
Overconstrained	80	0	0.171
	70	0	0.182
	60	0	0.172
	50	0	0.192
	40	0	0.178
	35	Unfinished	
Channeling	80	0	0.172
	70	0	0.172
	60	0	0.172
	50	0	0.172
	40	0	0.174
	35	Unfinished	

Table 4: Negative experiment with base=4

4 Discussion

We have attepmted to formulate three models, one naive, second based on the human solving tactic of overconstraining the expected sums, third adding channeling constraints to the overconstraining model. The methods used unfortunately failed to significantly improve performance of the model which are only fully usable on 9x9 puzzle. The volatility of positive experiments with 16x16 puzzles suggest that placemant of hints may have played bigger role in bringing up the solution quicker than the number of these hints by itself. A better model may therefore benefit from utilising clever preprocessing on the puzzle, perhaps generating individualy relevant constraints. This however should be subject of further study.