Solving Sudoku's with extended constraints using SAT solvers

Simone van Bruggen & Mark Romme
https://hahamark1.github.io/KRthelastkillersamurai/

October 3, 2017

Greater-than and colour Sudoku's are special types of Sudoku's in which an extra set of constraints is added to the normal base of Sudoku constraints. In this report, a study is presented in which is checked if adding more constraints to the regular constraints of Sudoku makes the puzzle easier to solve. Adding more constraints is measured by literal per clause rate and difficulty is measured by decision level, number of added clauses and solving time. An evaluation on each puzzle type ($N = 1800$) is given from which is concluded that for a higher literal per clause rate, the difficulty of the Sudoku puzzle is higher. However, some points of discussion are raised. Finally, an option for improving the experiment is given.

1 INTRODUCTION

Sudoku puzzles are an old type of Japanese number games. The puzzle consists of a $n \times n$ grid which should be filled with numbers $n \in N = \{1,...9\}$ and may have several pre-filled digits. Each cell should have exactly one number and every number must appear exactly once in each row, column and $\sqrt{n} \times \sqrt{n}$ block. These rules give the mathematical constraints which have to be satisfied in order to solve the puzzle. Besides these constraints, more constraints can be added to change the character of the puzzle.

One of these variations on regular Sudoku's is the **coloured Sudoku**. This type of Sudoku has coloured cells with nine different colours. Every 3x3 grid in the Sudoku has exactly one cell for each colour. Next to satisfying the constraints of a regular Sudoku, each number can only appear once in the nine cells with the same colour.

Another form of adding constraints to alter the challenge of a Sudoku is the **greater-than** (GT) Sudoku. These Sudoku's don't have pre-filled cells but an greater-than (>) or smaller-than (<) symbol is added between the cells of each $\sqrt{n} \times \sqrt{n}$ block. This adds the constraint that for two aligning cells A and B and the line between the cells depict >, it follows that $A > B$. This works similarly for <.

Due to the large solution space of Sudoku puzzles (6×10^{21}), a naive search algorithm is not feasible for solving. Therefore a SAT approach is used for solving the Sudoku puzzles. In order to do this, the constraints of the Sudoku have to formalized to CNF-form, the format which the SAT-solver can solve. After solving, the outcome has to be decoded to find the solution to the Sudoku puzzle.

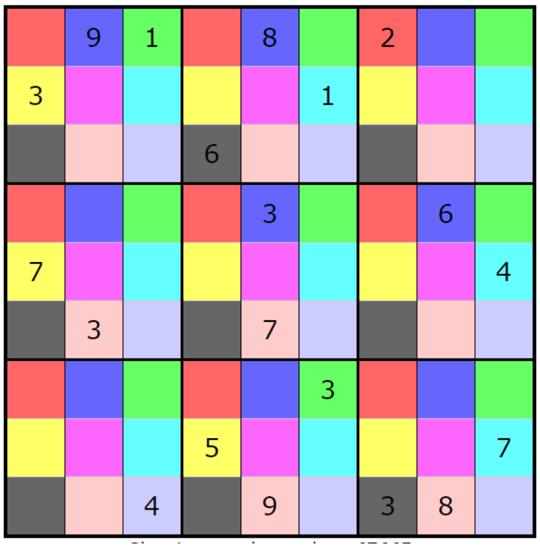

Figure 1.1: Example of a coloured Sudoku

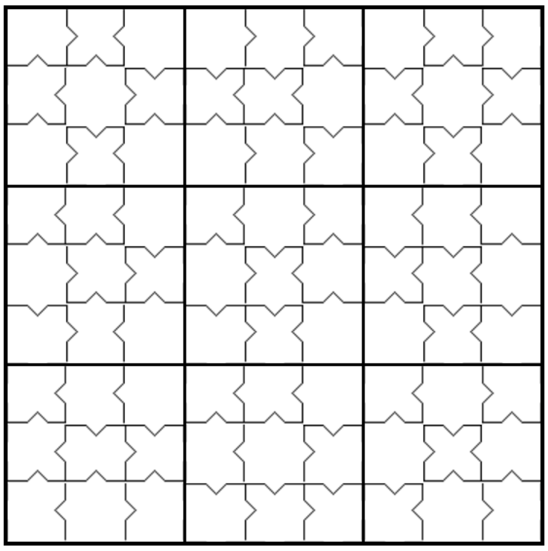

Figure 1.2: Example of a Sudoku

1.1 Hypothesis

Since having more constraints leads to less options to choose from during the solving of the puzzle, we propose the following hypothesis:

Adding more constraints to the Sudoku puzzle makes the puzzle easier to solve for a SAT solver.

We are going to test this hypothesis by studying the two variations of Sudoku we mentioned in the introduction, and using the idea of modeling Sudoku's as a propositional satisfiability (SAT) problem [5]. This way, we can use a SAT solver to solve and analyze the puzzles. Difficulty of solving the Sudoku will be measured by the decisions that have to made, conflicts added and the total CPU run time. If these measures are higher, the puzzles is more difficult to solve. Studying the differences between solving of regular Sudoku's and more constrained Sudoku's is an interesting subject to investigate, since this can give us more insight in the performance of SAT solvers on constraint-based problems.

2 Experimental setup

2.1 Data generation

To perform this research, a need for different types of Sudoku puzzles arose. A set of coloured, GT, and regular Sudoku puzzles was taken from a Norwegian website with different Sudoku samples[3]. This was done with the use of a web scraper. For all three Sudoku types, an equal number of puzzles for each of the websites difficulty levels was taken to ensure the equivalence in puzzle difficulty. In total a sample of 1000 puzzles per puzzle type was taken over 6 different difficulty levels. The puzzles themselves were chosen randomly.

2.2 Metrics

Metrics used to evaluate the difficulty of the puzzles are the maximum decision level and the number of decisions, the number of added conflict clauses and the total run time. The measure of added constraints will be measured by the average number of literals per clause. From these metrics, a conclusion on the change in difficulty considering added constraints can be found.

2.3 SAT Solver

The Sudoku puzzles were solved with the use of zChaff, a robust and popular SAT solver. To use it, a Python wrapper was created, so the use of zChaff was automated. zChaff is designed with both performance and capacity in mind and furthermore produces a wide range of evaluation statistics, such as number of conflict clauses, number of learned clauses and run time. We also solved the Sudoku puzzles using Picosat[1]. While this solved our puzzles efficiently, it did not provide the statistics we were looking for.

2.4 Encoding

2.4.1 Regular Sudoku's: naive vs. efficient

To solve Sudoku's using SAT solvers, we need to rewrite our constraints into a logic-based format and convert this to Conjunctive Normal Form (CNF).

To encode the constraints for regular Sudoku's, the following rules need to be implemented[5]:

- There is at least one number in each entry: $\bigwedge_{x=1}^{9}\bigwedge_{y=1}^{9}\bigvee_{z=1}^{9} s_{xyz}$
- Each number appears at most once in each row: $\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{9}\bigwedge_{x=1}^{8}\bigwedge_{i=x+1}^{9}(\neg s_{xyz} \vee \neg s_{iyz})$
- Each number appears at most once in each column: $\bigwedge_{x=1}^{9}\bigwedge_{z=1}^{9}\bigwedge_{y=1}^{8}\bigwedge_{i=y+1}^{9}(\neg s_{xyz} \vee \neg s_{xiz})$
- Each number appears at most once in each 3x3 sub-grid:
  $\bigwedge_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3}\bigwedge_{k=y+1}^{3}(\neg s_{(3i+x)(3j+y)z} \vee s_{(3i+x)(3j+k)z})$
  $\bigwedge_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3}\bigwedge_{k=y+1}^{3}\bigwedge_{l=1}^{3}(\neg s_{(3i+x)(3j+y)z} \vee s_{(3i+k)(3j+l)z})$

Using these constraints, we arrive at the *minimal encoding*. However, we need to extend these clauses to also account for the 'exactly one' constraint.

- There is at most one number in each entry: $\bigwedge_{x=1}^{9}\bigwedge_{y=1}^{9}\bigwedge_{z-1}^{8}\bigwedge_{i=z+1}^{9}(\neg s_{xyz} \vee \neg s_{xyi})$
- Each number appears at least once in each row: $\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{9}\bigvee_{x=1}^{9} s_{xyz}$
- Each number appears at least once in each column: $\bigwedge_{x=1}^{9}\bigwedge_{z=1}^{9}\bigvee_{y=1}^{9} s_{xyz}$
- Each number appears at least once in each 3x3 sub-grid: $\bigvee_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3} s_{(3i+x)(3j+y)z}$

In total, this leads to the *extended encoding* which consists of a formula with approximately 13k clauses. Apart from this, a unit clause is added for every pre-filled number.

2.4.2 Encoding Sudoku variations

To solve the Sudoku variations, some rules need to be added to the encoding. In the case of the coloured Sudoku's, next to the clauses for regular Sudoku's, similar clauses are added for every colour in the Sudoku. More specific, we add the same clauses as before for 'at least' and 'at most', for all cells which have the same colour.

To encode the GT Sudoku's, clauses which correspond to the greater-than and smaller-than signs in the grid need to be added. For two cells A and B next to each other and $A > B$, we want to add the logical consequence $(A9 \wedge (B8 \vee B7 \vee B6..)) \vee (A8 \wedge (B7 \vee B6 \vee B5...))$....[6]. As this is not yet in CNF, it needs to be converted this using DeMorgan's laws. Given the following toy formula:

$$(A2 \wedge B1) \vee (A3 \wedge (B2 \vee B1)) \vee (A4 \wedge (B3 \vee B2 \vee B1)$$

When rewritten to CNF, this leads to the following formula:

$$(A2 \vee A3 \vee A4) \wedge (B1 \vee A3 \vee A4) \wedge (B1 \vee B2 \vee A4) \wedge (B1 \vee B2 \vee B3))$$

Now, to generalize this result in the encoding, the following clauses are added: For $A > B$, with $A = xy$ and $B = vw$: $\{\bigvee_{z=2}^{10} = s_{xyz}, \bigwedge_{i=1}^{9}\bigvee_{z=1}^{i+1} s_{vwz}, \bigwedge_{i=1}^{9}\bigvee_{l=i+2}^{10} s_{xyl}\}$

3 Results

3.1 Experimental Results

The encoded SAT-problems were solved with the use of zChaff [2]. Puzzles were solved efficiently, in less then a 0.01 second [1]. The evaluation of 1800 normal Sudoku's, 1800 color Sudoku's and 1800 GT Sudoku's was averaged to get to comparable results. The solution and solving statistics returned by zChaff can be found in Appendix A. The relevant outcomes can be found in Table 3.1 and will be discussed below.

	Literals per Clause	Max. dec. level	Num. of dec.	Add. confl. clauses	Total runtime
Regular	2.05	5.76	21.62	8.51	0.0001067 s
Colour	2.04	7.87	34.89	14.75	0.0004177 s
GT	2.50	36.76	1364.98	776.91	0.0228844 s

Table 3.1: Experimental results from zChaff: average statistics after solving ($N = 1800$)

3.2 Interpretation

The results found in our experiment reject our hypothesis. When comparing the regular Sudoku with the coloured and GT Sudoku, it can be seen that for all the measures used to express difficulty, the regular Sudoku has a lower score than the Sudoku's with extended constraints. This does directly reject our claim that the addition of constraints makes it easier to solve a Sudoku puzzle. This may be caused by the fact that the problems become more complex by adding more constraints (and thus more clauses). More clauses does not necessarily make the problem easier to solve for a SAT solver. As can be seen from Table 3.1, the number of literals per clause is higher for the GT puzzle. As the number is still lower than 4.3, from the 3-SAT hardness problem we expect the complexity of the GT to be higher.

Furthermore, it seems that this result corresponds to the way most humans would solve these puzzles. Adding more constraints makes puzzles harder rather than easier for a human being to reason about, as there are more factors to take into account.

4 Discussion

The hypothesis proposed in this research was based on Sudoku's with more constraints in general. In our experiment, we focused only on coloured Sudoku's and GT Sudoku's and did not give a general proof. However, it does not seem necessary as our examples of variations gave a clear result.

In Table 3.1, it is clear that the GT Sudoku is actually much harder (given the metrics we used to define 'hardness' of a puzzle).

For the coloured Sudoku, only a few extra clauses of regular length are added. However, the GT Sudoku adds many more clauses which are also very long, as explained in section 2.4.2. In Figure 4.1, we plotted the average number of literals per clause. The data used in this figure can be found in the appendix. It shows that the average number of literals is significantly larger for the GT Sudoku than for regular and coloured Sudoku's. This makes the comparison between these Sudoku's more difficult and less reliable.

When comparing the difference between regular Sudoku's and Sudoku's with extended constraints, most clauses in the CNF problem come from the regular Sudoku. The constraints we add for the Sudoku variations only add a relatively small amount of clauses. This makes the comparison somewhat more complicated, as we are still mainly looking at the clauses for regular Sudoku's.

It would be a good idea to look into making this same comparison using a more advanced encoding. An example of such an encoding is explained in Section 5.

As a final remark, it should be noted that while run time is used as one of the measures for difficulty of solving, this is a not considered a good metric and should not be used on its own. Here it is mainly added as it does clarify our results.

[1] PC specs: Intel® Core™ i7-7700HQ CPU @ 2.80GHz × 8, RAM 15,4 Gb

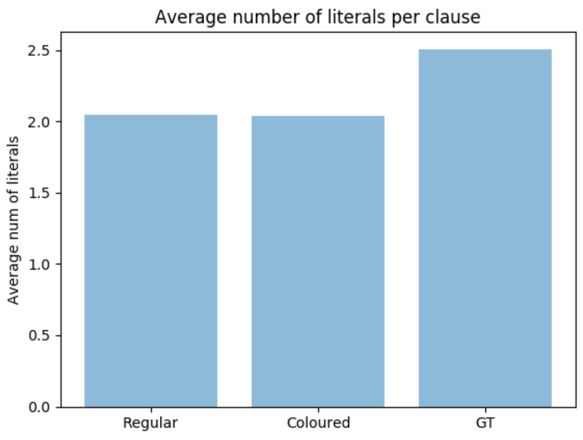

Figure 4.1: Plot of average number of literals per clause

5 Future work

Since the naive encoding leads to n^3 clauses, it is useful to try to use a more efficient encoding for the standard Sudoku encoding. Kwon and Jain [4] proposed an encoding that potentially leads to a 90% decrease of clauses for the $9x9$ size Sudoku. The main idea of making the coding more efficient is the removal of redundant literals and clauses, based on the pre-filled cells. When a certain cell is filled with a number, this has consequences for many clauses concerning that block, row and column. Removing these redundancies is done by partitioning the variables V in three sub classes:

$$V = V^+ \cup V^- \cup V' \tag{5.1}$$

where V^+ corresponds to the true variables, V^- corresponds to the false variables, given the Sudoku rules and the pre-filled cells, and V' to remaining variables. Variables representing other numbers associated with a pre-filled number[2], and variables representing values which are duplicated in a row or column are removed. The complete CNF formula now looks like this:

$$\phi' = Assigned \Downarrow V^+ \cup (Cell_u \cup Row_u \cup Block_u) \Downarrow V^- \cup (Cell_d \cup Row_d \cup Block_d) \downarrow V^- \tag{5.2}$$

In which:

$$\begin{aligned} \phi \Downarrow V^+ &= \left\{C \in \phi | \neg\exists_{L\in C} \cdot (L = x \land x \in V^+)\right\} \\ \phi \Downarrow V^- &= \left\{C \in \phi | \neg\exists_{L\in C} \cdot (L = \neg x \land x \in V^-)\right\} \\ C \downarrow V^+ &= \left\{L \in C | \neg(L = \neg x \Rightarrow x \in V^+)\right\} \\ C \downarrow V^- &= \{L \in C | \neg(L = x \Rightarrow x \in V^-)\} \end{aligned}$$

Where ϕ is the total set of clauses, C is a clause and L is a literal. Which was proven to have a satisfiability equivalence relation with the naive-SAT-encoding. Therefore, this encoding could be used to encode the basic part of the normal and the color Sudoku. For the GT Sudoku, this however does not work. However, given certain block in a GT Sudoku, one could argue that certain values can be filled as all other values have to be either bigger or larger.

This encoding is relevant in finding testing the hypothesis as the proportion of clauses in the 'basic' Sudoku encoding to the added number of clauses from the color and GT constraints becomes closer to 1 if the number of clauses in the basic encoding shrinks. The outcome should therefore become clearer in the results.

For the rules for this encoding and the proof that this encoding is correct, we refer to the paper[4]. The implementation of the encoding can be found on our project page [3].

[2] Kwon and Jain[4] refer to these pre-filled cells as fixed cells, but these two names refer to the same concept of a cell which has a value assigned before the solving of the puzzle.

[3] See Sudoku_decoder.py at `https://github.com/hahamark1/KRthelastkillersamurai`

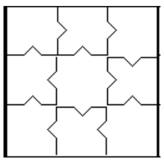

Figure 5.1: GT block

Furthermore, it would be interesting to investigate how adding constraints to regular Sudoku problems behaves when we scale the Sudoku's to for example 16x16 or even bigger grids.

6 Conclusion and Acknowledgement

This report presents an experiment to test the hypothesis: Adding more constraints to the Sudoku puzzle makes the puzzle easier to solve for a SAT solver. The experiment rejects the hypothesis. For both color and GT puzzles it is shown that they are more difficult to solve than normal Sudoku's. Both the database and source code can be found on Github. The authors would like to thank Prof. F. Van Harmelen and MSc. Finn Potason for their information and feedback.

References

[1] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, pages 75–97, 2008.

[2] SAT Research Group et al. Princeton university. zchaff. *Sıtio: http://ee. princeton. edu/˜ chaff/zchaff. php*, 2007.

[3] Vegard Hanssen. Vegard hanssens hjemmeside, 2017.

[4] Gihwon Kwon and Himanshu Jain. Optimized cnf encoding for sudoku puzzles. *Hermann M (ed.) Proceedings of the 13th International Conference on Logic Programming for Artificial Intelligence and Reasoning*, 2006.

[5] Inês Lynce and Joël Ouaknine. Sudoku as a sat problem. *International Symposium on Artificial Intelligence and Mathematics*, January 2006.

[6] Gao Shan and Roland Yap. Solving puzzles (eg. sudoku) and other combinatorial problems with sat. 2008.

7 Appendix

7.1 Results experiment

	Max. dec. level	Num. of dec.	Orig. num. vars	Orig. num. clauses	Orig. num. lit.
Regular	5.76	21.62	729	11774	24085
Colour	7.87	34.89	729	14683	29911
GT	36.76	1364.98	729	12717	31833

Table 7.1: Experimental results from zChaff: average statistics before solving

	Add. confl. clauses	Del. confl. clauses	Del. lit.	Add. confl. lit.	Num. of implic.	Total runtime
Regular	8.51	0	0	475	1238.2	0.0001067
Colour	14.75	0	0	852	1798.8	0.0004177
GT	776.91	0.065	0.065	15034	72763.9	0.0228844

Table 7.2: Experimental results from zChaff: average statistics after solving