UNIVERSITY OF AMSTERDAM

Analysing the correlation of symmetrical sudokus with their solving time using SAT solvers

Ruben Blom - 10684980
Luca Simonetto - 11413522

September 22, 2016

1 Introduction

Satisfiability solvers, or SAT solvers are complete methods that can prove or disprove the satisfiability of formulas written in propositional logic. For every literal in a formula the algorithms strategically assigns truth values. If the formula can be satisfied, then the truth values for all the propositional variables are returned, otherwise the problem is considered unsatisfiable.

This paper focuses on SAT solving of sudoku puzzles. A sudoku is a logic-based puzzle of 'placing numbers' that is contained in a 9-by-9 grid divided in nine, 3-by-3 grid sections. An example of a sudoku puzzle is shown in Figure 1. The goal in solving a sudoku is placing number in the grid in such a matter that all columns and rows, as well as each 3-by-3 grid contain the digits from 1 to 9 without repetition of any digit. For this research, each sudoku constraint is encoded into a propositional logic formula in order to make it solvable with the use of a SAT solver.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

(a) Empty sudoku

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

(b) Solved sudoku

Figure 1: Example of a empty sudoku and it's solved version.

Sudokus can have different symmetrical properties. These properties rise when the position of each given on the board is mirrored by another one. The symmetrical proper-

ties that are researched in this paper are presented in Table 1 and four example puzzles are given in Figure 2.

Label	Description
Anti-diagonal	Symmetry over \ diagonal
Diagonal	Symmetry over / diagonal
Horizontal	Mirror symmetry over x-axis
Vertical	Mirror symmetry over y-axis
Horiz_vert	Mirror symmetry over both x and y-axis
Pi Rotational	Rotational symmetry over 180°
Normal	No symmetrical properties

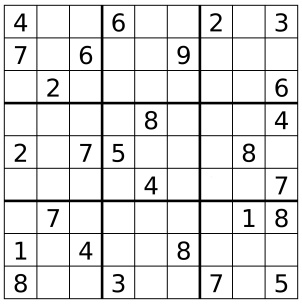
Table 1: Symmetry labels with their descriptions

In this paper, the following hypothesis will be researched:

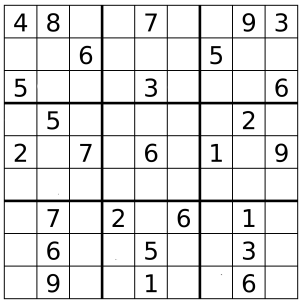
Will the symmetry property of a sudoku puzzle affect the time required for a computer to solve it?

The answer is non trivial as usually some pattern in a problem helps to solve it in the case of a human. To prove or falsify this hypothesis, the research focuses on the analysis of the correlation between the symmetrical properties of sudokus (with various level of difficulty) and the time spend by a SAT solver to solve them.

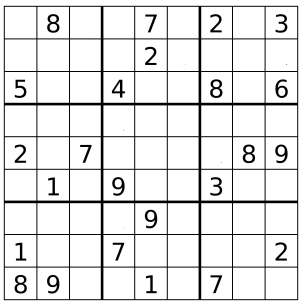
This paper consists of four parts. In Section 2 the methodology of testing this hypothesis is explained, in Section 3 the results are presented and in Section 4 the conclusion will be discussed and some possible future works are proposed.



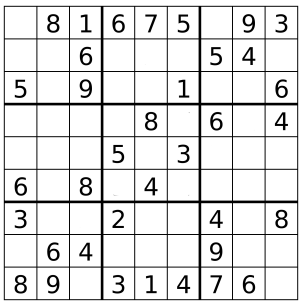
(a) Horizontal symmetry



(b) Vertical symmetry



(c) Diagonal symmetry



(d) Pi symmetry (180°)

Figure 2: Four examples of symmetry in sudokus

2 Method

In order to test the hypothesis, a custom dataset of sudokus has been created. Each sudoku has then been encoded as a propositional logic problem, in order to be solved by an existing SAT solver executed by a custom Python program that evaluates the times used in the search of a solution for each problem. The time spent to find each solution is recorded and grouped by difficulty and symmetry, elaborated and analyzed.

2.1 Dataset generation

As the testing of the hypothesis requires a dataset of sudokus with many types of symmetries, all the existing datasets had to be discarded, forcing the choice to the search of a specialised sudoku generator. The one used in this paper has been developed by Glenn Fowler[1] and has the ability to create and validate sudukos in various difficulties and with specific traits. The command used to generate the dataset is the following:

```
./generator -g -sS -e "E" -nN -f%v -o O
```

Flags:

`-g` generate a dataset

`-sS` desired symmetry `S` (horizontal, vertical, diagonal, pi rotational etc.)

`-e "E"` use the expression `E` for the generation. In this case `E`=valid&rating>n1& rating<n2 where `valid` indicates the generation of a valid sudoku and `rating` sets the difficulty level of the generated sudoku (on a scale of 0 to 9999)

`-nN` the number N of generated sudokus

`-f%v` formats the output to remove unnecessary data

`-o O` specifies the output file

Rating	Lower bound	Upper bound
Easy	0	2000
Medium	2001	4000
Hard	4001	6000
Very Hard	6001	8000
Extereme	8001	9999

Table 2: Sudoku difficulties and their rating boundaries

Using the boundaries, sudokus of five different difficulties have been created. The difficulties and their rating boundaries are shown in Table 2. For each difficulty, 50 sudokus have been generated resulting in a total of $5 \times 8 \times 50 = 2000$ sudokus. Of these 2000 sudokus, 250 don't have any symmetry in order to make possible to do comparisons.

2.2 SAT solver

To analyse the time spent by a SAT solver on each symmetry, it is necessary to use a deterministic solver, as a non deterministic approach could result in big variations in elapsed times. For this reason, PycoSat was chosen as a valid candidate. PycoSat[2] is a

[1] http://gsf.cococlyde.org/

[2] https://pypi.python.org/pypi/pycosat

Python wrapper for PicoSat[3], a deterministic SAT solver written in C. In this work, only the computation time is measured and there was no necessity for any statistical output from the solver, therefore PycoSat has been a straightforward choice.

2.3 Metric and experimental procedure

The chosen metric for testing the hypothesis is the elapsed time for each SAT solving run. Because this metric could be seen as non-optimal and prone to misreadings, a great effort has been undertaken in order to limit the variance.

This project's experimental procedure follows a rigorous sequence of actions, created ad hoc in order to filter from the results all the factors that otherwise could affect their reliability.

Firstly, the experiments have been done on the same computer, in order to get the same hardware configuration on each run thus reducing big time fluctuations.

Secondly, the elapsed time of each SAT solving session was not recorded using traditional methods that use system clock, as they are hugely influenced by the running processes and user operations: instead, CPU time was used. CPU time gives a more accurate reading and is not dependent on a defined metric, as it is a pure number. Furthermore, it is not highly biased by other processes and is not linked to a specific hardware configuration. However, CPU time is still influenced by the operating system tasks, something the user has no control over. Therefore, other steps have been implemented in order to reduce the noise with various degrees of precision.

The Python garbage collector has been disabled before each SAT run, in order to avoid unnecessary CPU cycles. This decreased the variances on the total times by around 12%.

Every dataset is run five times and the results are averaged. The highest and lowest outlier is removed from the total times for each dataset, increasing the precision in the measurement for each dataset. The process for a whole dataset is repeated five times and the outliers from those results are removed. The remaining three global results are averaged into the final result for the processed dataset. This process is illustrated in Figure 3 and Figure 4.

These steps greatly reduced the standard deviation of the final results, giving more precise and reliable readings to work with.

[3]http://fmv.jku.at/picosat/

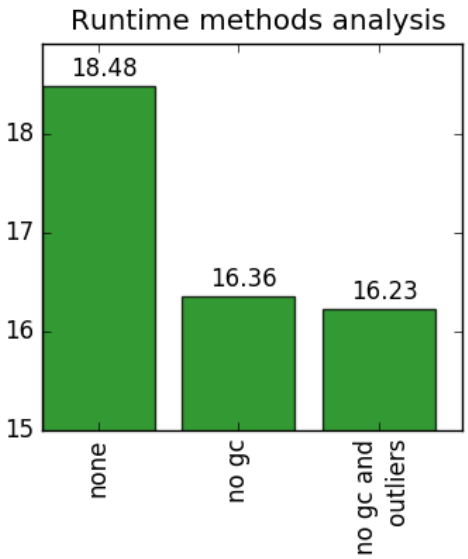


Figure 3: Comparison between different methods used to lower the noise in the readings

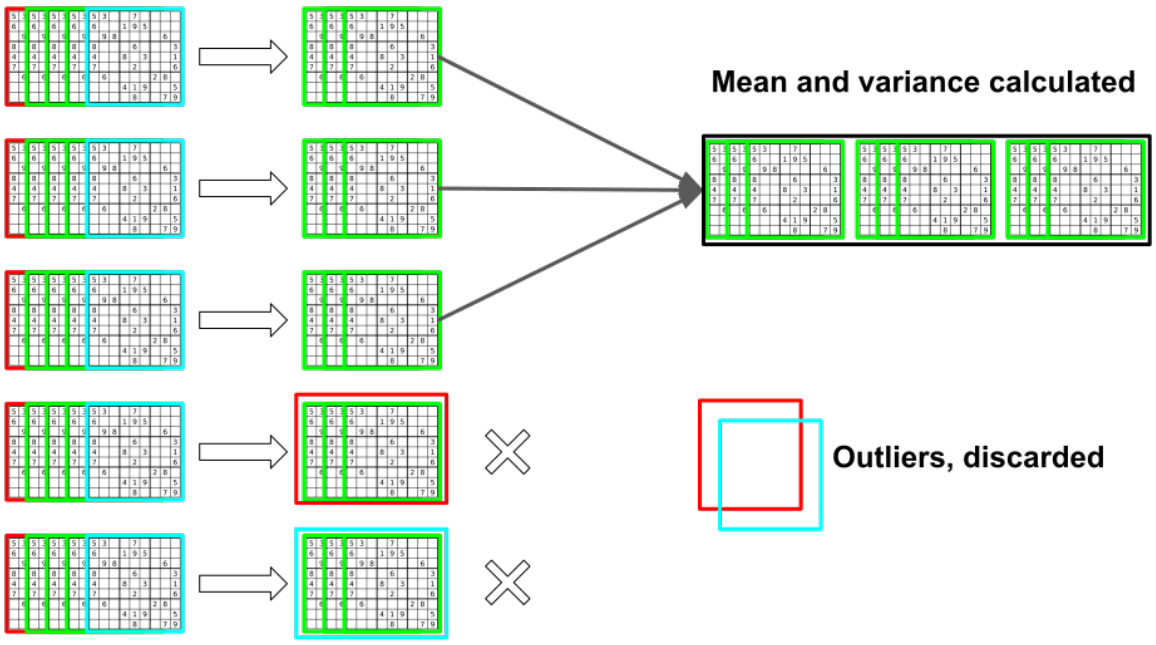


Figure 4: Procedure followed to remove short and medium length process interferences.

3 Experimental results

After testing our approach for consistency, doing multiple runs and averaging the times, the following results emerged:

Looking at Figure 5, and using the plotted data as reference it can be seen that, taking into consideration each difficulty separately, there is no visible correlation of symmetries and elapsed time. The deviation for each run (indicated by the vertical line above the bar) indicates that the fluctuations in the same symmetry are comparable to the fluctuations in different symmetries, thus rejecting the formulated hypothesis.

Comparing different dataset difficulties shows that the easy dataset has an average elapsed time that is noticeably lower than all the other difficulties. This pattern is not present between the higher difficulties, meaning that the effort taken by the SAT-solver does not differ, for example, between medium and extreme difficulty.

Finally, a slight difference of 0.3 can be seen when comparing the antidiagonal and vertical symmetry from the easy dataset. Although promising, this difference is not substantial enough to confirm the initial hypothesis, as a value of 0.3 roughly translates in three tenths of millisecond.

For a better comparison of the results, Figure 6 shows the same data divided by dataset difficulty.

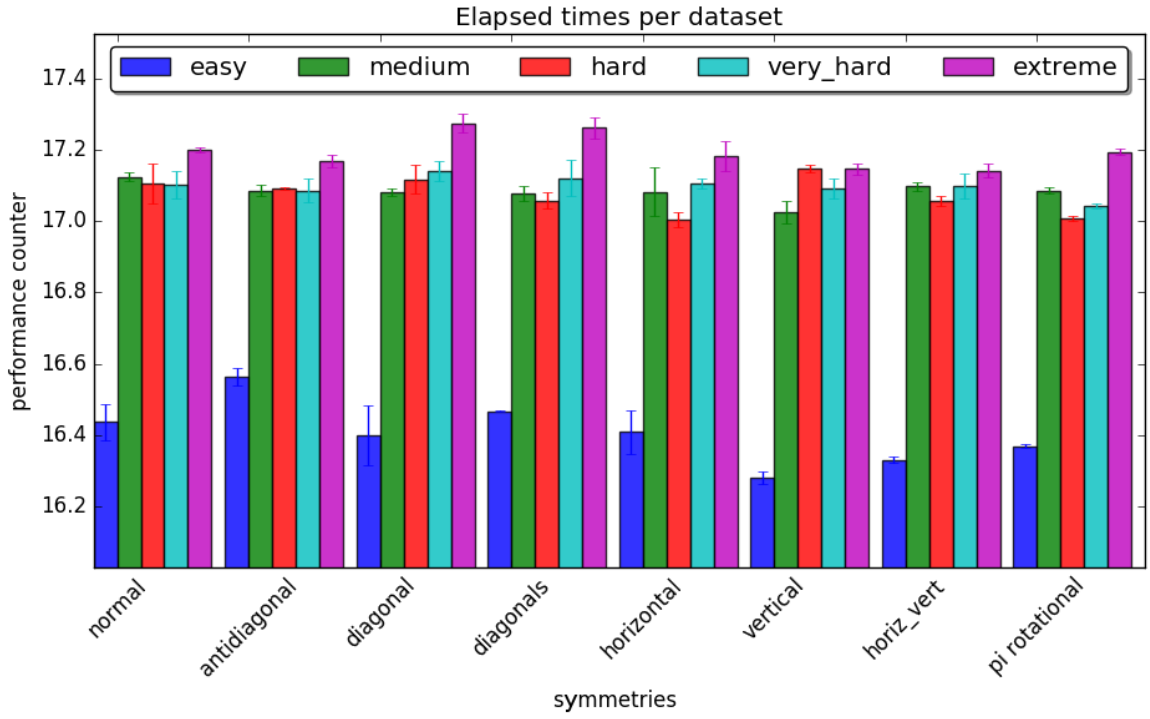


Figure 5: Summary of the results for a single run. Notice that the y-axis is defined by the CPU time, not elapsed time. Vertical lines over the bars show the standard deviation of the various runs with respect to te mean

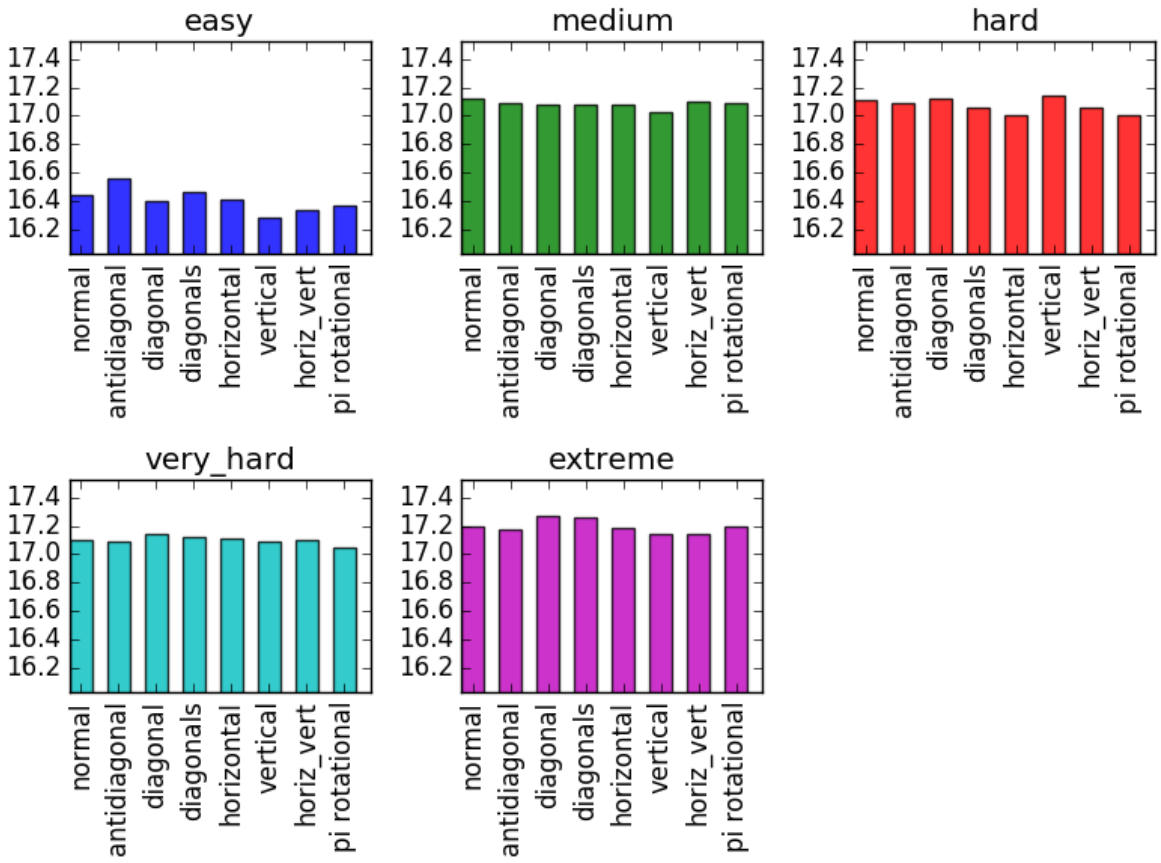


Figure 6: Plot for each difficulty, no patterns emerge, falsifying the hypothesis

4 Conclusions and Future Work

In conclusion, the hypothesis is falsified: solving a sudoku with symmetrical properties doesn't influence the time required by a SAT solver to find a solution.

Although this research resulted in the rejection of the hypothesis, valuable insights in the correlation between the time required to solve a sudoku and it's difficulty was obtained.

Furthermore, this research provided a gain in knowledge about sudokus, and more future work should be done in order to get more accurate results and find other, unseen correlations.

In the future this work could be continued with different metrics and techniques, in order to determine for example, if the presence of symmetry in a given sudoku will result in an different number of backtracks for the SAT solver, a higher number of decisions to make or higher number of clauses that are created. Also, using a non deterministic SAT solver could generate other unexpected results, like the ones seen in this paper.