SAT-solving: The impact of number of givens and symmetry on the computational complexity of Sudokus

David Rau (11725184), Mirthe van Diepen (10327428)

University of Amsterdam

david.rau@student.uva.nl, mmvandiepen@gmail.com

Abstract

Sudoku is a very well known puzzle that many people consider as a fun game. It can be translated into a propositional SAT-Problem and subsequently solved by a computer. The goal of this paper is to examine the impact of the number of givens and rotational-symmetry on the computational complexity of Sudokus. In the course of this research several encodings (extended-, optimal-, minimal encoding) were examined. In this research the minimal encoding *was chosen to convert the Sudoku into the Conjunctive Normal Form (CNF). The resulting set of clauses was fed to a SAT-Solver and statistics were collected about the solving process in order to gain insights about the complexity. We propose the hypothesis that the number of givens is negatively correlated with the computational complexity, whereas the symmetry has no impact. This can be confirmed in the course of the work.*

I. Introduction

In many Sudoku booklets Sudoku puzzles are ordered by difficulty. But how is the difficulty of a Sudoku determined? Some people state for example that it is easier for them to solve symmetric Sudokus. Anyhow, there is no general way of measuring the difficulty. In this research we try to give some insights into the relation of the properties of Sudokus to the computational complexity.

The most common Sudokus differ in the number of *givens* as well as in the *location* where the givens are placed in the grid. Although, the complexity of solving a Sudoku in the classical sense is measured with respect to humans skills, we examine the complexity concerning computers.

Whether the number of givens and the location of them is actual relevant for the computational complexity of the Sudoku remains unclear. In this research we examine both.

We will analyse the impact of the *location* for two types of Sudokus having symmetric and non-symmetric properties respectively:

First, Sudokus that show no symmetry. Second, a special type of which its givens show a two-fold rotational symmetry: the givens are 180-degree rotated within the horizontal axis. For both types we will also examine effect of the *number of givens* on the computational complexity. Therefore we will solve Sudokus with a different range of givens.

Our hypothesis is that the number of givens is depending on the computational complexity on such a way, that for a computer it is easier to solve a Sudoku with a bigger number of givens, whereas symmetry has no impact on the complexity.

This is in contrast to the complexity for humans. In many Sudoku booklets the difficulty is not based on the number of givens since some *hard* puzzles have more givens than some *easy* ones. Although, it seems plausible to us that the computational complexity decreases while the number of givens increases, because the increase of givens comes with an increase in logical constraints.

In this paper we will first clarify how we generated the data set. Second, we will describe the methods that are necessary to turn a Sudoku into a SAT-Problem. Third, we will

present the results of the research. And eventually we will discuss whether the hypothesis holds for the results we obtained.

II. Materials & Methods

For solving a Sudoku there are different techniques. In this approach, however, the Sudoku is formulated as a SAT-Problem consisting of constrains in a logical form [1]. Those can be given to a SAT-solver which is then able to find a solution for a given puzzle. This solution is normally obtained within a few milliseconds even though the 9x9 Sudoku is known to be NP-Complete problem. The SAT-solver used in this research is the *64-bit zChaff* [2]. According [3] zChaff is using three strong methods from the field of Knowledge Representation. The first important method is *The watched literals scheme*, which continuously watches two literals of all not satisfied clauses, these watched literals are non-false. The second method is *Fast backjumping* which "lets a solver jump directly to a lower decision level d when even one branch leads to a conflict involving variables at levels d or lower only"[3]. The third important method is *Randomized restarts* which randomly restart at decision level zero.

The puzzles that were examined in this research, were generated by Native QQWing [4] built in C++. Using the QQWing commandline tool 60.000 Sudokus were generated. These consist of 30.000 non-symmetric and 30.000 180-Degree rotated symmetric Puzzels. Both types of Sudokus were generated with the same difficulty level: 'any'. All sample Sudokus are proper Sudokus. This means they have by definition exactly *one* solution.
In the following step the samples were sorted by the number of givens by the use of the *filter_num.sh* script [5]. Subsequently looking at the number, that each subset was comprised of, it was striking that the distribution highly uneven. For the non-symmetric data set the number of obtained Sudokus peaked around 25 givens. For the symmetric data set it peaked around 28 givens. For the range 30-32 number of givens of the 180-degrees rotated Sudokus there were less then 100 Sudokus in each Subset. For the non-symmetric Sudokus this holds for range 22-23.

In order to make the results comparable, a cutoff for the size of all subsets, was chosen. Subsets with size < 100 were excluded and 100 Sudokus of each Subset were randomly picked.

As stated earlier a Sudoku can be considered a SAT-Problem. Therefore, it is represented by n propositional variables that can be either assigned to the value true (1) or false (0). In general a Sudoku consists of a grid of $n \times n$ cells. The value of each cell can be assigned to a number from $1, ..., n$. Therefore we need n variables to describe each field of the grid. In total $9 \cdot 9 \cdot 9 = 729$ were needed to describe the 9×9-Sudoku. We define a 3-tuple (r, c, v) to be true to be a variable which is true iff cell in row r and column c is assigned to a number [6].

However, before the Sudokus can be fed into a SAT-Solver they need to be in a specific Boolean logic form: The Conjunctive Normal Form (CNF). It consists of conjunction of clauses, where the literals within the clauses are connected with disjunctions e.g.

$$(A_1 \vee B_1) \wedge (\neg A_1 \vee B_2) \wedge ... \wedge (A_n \vee B_n)$$

Therefore the whole set of clauses evaluates to true if at least one literal in each clause is true.

In the special case of Sudokus different kinds of encoding can be used to translate the general constraints into CNF. We examined several different encodings (optimal-, minimal encoding). However, in this research the *minimal encoding* was chosen to convert the Sudoku into the Conjunctive Normal Form (CNF). For all the other encodings the number of added conflict clauses was very small and wouldn't have give strong indications about the computational complexity.
Below is a brief explanation of this type of encoding that we implemented [5] according to *Optimized CNF Encoding for Sudoku Puzzles* [6]. For the explanation denote $a \pmod 3$ with $\overline{a}$.

This set of clauses examine if each cell has

at least one number $1, \ldots, 9$:

$$Ce := \bigwedge_{r=1}^{9} \bigwedge_{c=1}^{9} \bigvee_{v=1}^{9} (r, c, v). \tag{1}$$

This set of clauses examine if each row has at most one number $1, \ldots, 9$:

$$R := \bigwedge_{r=1}^{9} \bigwedge_{v=1}^{9} \bigwedge_{c_i=1}^{8} \bigwedge_{c_j=c_i+1}^{9} \neg(r, c_i, v) \vee \neg(r, c_j, v). \tag{2}$$

This set of clauses examine if each column has at most one number $1, \ldots, 9$:

$$Co := \bigwedge_{c=1}^{9} \bigwedge_{v=1}^{9} \bigwedge_{r_i=1}^{8} \bigwedge_{r_j=c_i+1}^{9} \neg(r_i, c, v) \vee \neg(r_j, c, v). \tag{3}$$

This set of clauses examine if each block has at most one number $1, \ldots, 9$:

$$B := \bigwedge_{\tilde{r}=1}^{3} \bigwedge_{\tilde{c}=1}^{3} \bigwedge_{v=1}^{9} \bigwedge_{r=1}^{9} \bigwedge_{c=r+1}^{9} \neg(3\tilde{r} + \bar{r}, 3\tilde{c} + \bar{r}, v) \vee \neg(3\tilde{r} + \bar{r}, 3\tilde{c} + \bar{c}, v). \tag{4}$$

At this stage the general Sudoku constraints are represented by a Propositional Logic that can be processed by a SAT-Solver. However, an actual Sudoku, consisting of pre-given numbers in a grid, needs to be added to the set of clauses. Let $G := \{(r, c, v) \in \{1, 2, \ldots, 9\} \mid$ the givens with row r, column c and value $v\}$ be the set of givens. The set of clauses for the givens of a particular Sudoku is:

$$Gi := \bigwedge_{g \in G} g. \tag{5}$$

Taking the conjunction of the equations (1)-(5) one may obtain the minimal encoding which is as follows:

$$Ce \wedge R \wedge Co \wedge B \wedge Gi.$$

Since zChaff needs the input in the machine readable DIMACS/CNF format the resulting CNFs were converted to DIMACS by the use of the package *hyperactive* [7]. DIMACS is a simplified version of CNF. Each row in DIMACS represents one clause where conjunction between literals and disjunction between clauses are implicit. Negated literals are indicated by a minus previous to the number. The corresponding DIMACS to the CNF example above is given by:

```
c example
p cnf 2 4
 1  2
-1  2
...
 3  4
```

As the final step the encoded Sudokus were fed into the zChaff SAT-Solver and statistics about the solving process collected. As Features of complexity Number of Decisions, Number of Implications, Number of Added Conflict Clauses and the Total Run Time. Making a *decision* means assigning a value to one cell in the grid. But not in all cases of assigning it is necessarily a decision since in the cases the assigning is done as a logical consequence from the CNF, it is not a decision. The feature *Number of Decision* counts the total number of decisions which were made to solve the Sudoku. Assigning a value to one cell in the grid which is done as a logical consequence from the CNF of the Sudoku, is an implication. *Number of Implications* counts the total implications which where made during the solving process. Every time the algorithm makes a decision, a conflict clause can arise, i.e. one of the cells in the grid is assigned to some value such that there is a clause in the constrains for which all literals are assigned to be false. Since we consider the conjunction of all clauses, such a conflict clause will lead to a contradiction. Hence the algorithm needs to change the decision and its administered the conflict clause, so it will not get the same contradiction. *Number of Added Conflict Clauses* is the number of conflict classes which are administered. The Total Run Time is the time the solving process took to solve the Sudoku.

Table 1: *Statistics given by zChaff solving **non-symmetric** Sudokus. For each # Givens, 100 Sudokus were solved. The values in this table corresponds to the averaged sum respectively. The values underneath the horizontal line mark the values could be compared to the 180-degrees rotated Sudokus.*

# Givens	# Decisions	# Implications	# Added Conflict Clauses	Total Run Time (in secs)
29	15.37	1191.75	5.25	0.0003311
28	22.85	1395.64	7.82	0.0003913
27	28.6	1545.46	9.39	0.0004909
26	34.17	1676.02	11.3	0.0005108
25	41.75	2014.96	14.8	0.0006118
24	55.13	2374.37	19.02	0.0007234
23	65.32	2674.02	23.66	0.0008006
22	110.76	3927.0	37.12	0.0011463

Table 2: *Statistics given by zChaff solving **180-degrees rotated** Sudokus. For each # Givens, 100 Sudokus were solved. The values in this table corresponds to the averaged sum respectively. The values underneath the horizontal line mark the values could be compared to the non-symmetric Sudokus.*

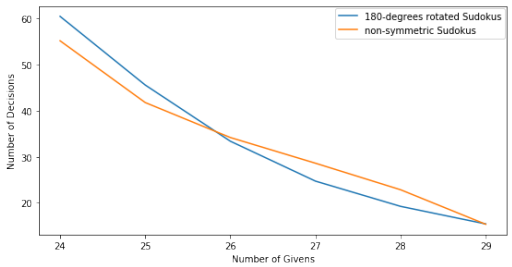
# Givens	# Decisions	# Implications	# Added Conflict Clauses	Total Run Time (in secs)
32	8.99	972.84	3.07	0.0002795
31	9.03	966.44	3.17	0.0002654
30	15.14	1199.05	5.61	0.0003413
29	15.44	1135.62	5.03	0.0003334
28	19.25	1280.01	6.95	0.0003915
27	24.72	1525.76	9.06	0.0004208
26	33.37	1790.89	11.93	0.0005400
25	45.55	2174.34	16.77	0.0006938
24	60.42	2657.32	22.7	0.0007693

III. Results

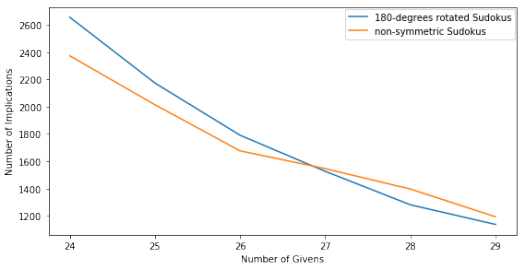
While solving the Sudokus statistics were collected from the solving process, the most useful features are shown in Table 1. In the first column the number of givens are shown. The remaining columns in the table 1 are Number of Decisions, Number of Implications, Number of Added Conflict Clauses and the Total Run Time.

In Table 1 the statistics of the non-symmetric Sudoku are shown. We may observe that the values of all features increase if the Number of Givens decreases. For the number of decisions this means if there are more givens, more cells can be assigned by only using the constrains, but if there are less givens the algorithm will assign more cells randomly since the constrains do not conflict directly with this decision. This does not contradict with the increasing values of the number of implications since if there are less givens then there are more constrain-free cells in the grid, i.e. there is more space for making implications. Considering the Total Run Time as a valid feature is questionable because the Run Time is sensitive to many external influences that can distort the values. However, we can see that the Run Time behaves similar to the other features since it increases when the number of givens decreases.

In Table 2 the statistics of the 180-degrees rotated Sudoku are given we can see that the most features increase if the number of givens decrease. For 29 and 30 givens only the num-



(a) ***Number of Decisions*** *of 180-degrees rotated (blue) and non-symmetric (orange) Sudokus plotted against the number of givens (24-29).*



(b) ***Number of Implications*** *of 180-degrees rotated (blue) and non-symmetric (orange) Sudokus plotted against the number of givens (24-29).*

ber of decisions increases, the other values of the features decrease.

In Figure 1a the number of decisions of the 180-degrees rotated Sudokus and the non-symmetric Sudokus are plotted against the number of givens. Note that the graphs only compares the number of givens from 24-29 this corresponds to the numbers above and underneath the line in the graphs 1 and 2 respectively. There are two lines in the graph which represent the average number of decisions comparing to the number of givens. The distance between the lines is smaller than 6 decisions on each point. There are two intersection points.

In Figure 1b one may obtain the number of implications of the 180-degrees rotated Sudokus and the non-symmetric Sudokus. The difference between the number of implication is big for lower number of givens. However the distance of the line is not monotone; after the intersection point the distance decreases and increases.

IV. Discussion

The results show that the values of the features of non-symmetric Sudokus increases while the number of givens decreases. Therefore, using these features as indicators for computational complexity we state that solving a Sudoku with more givens is less complex compared to a Sudoku with less givens. Thus, our hypothesis that the computational complexity is negative correlated with a increasing number of givens can be confirmed.

In our case finding symmetric and non-symmetric Sudokus with a range of number of givens caused an obstacle. To cope with the lack of data we had to fall back to a Sudoku generator. Since it is not clear how the difficulty levels of QQwing effect the computational complexity, we didn't restrict our data generation to any difficulty level.

Due to the inequality in the size of the resulting subsets, that are comprised of puzzles with k-givens, we were to forced to pick maximally 100 Sudokus of each subset. The slight bigger values for symmetry we interpret as a biased result. Therefore we also can confirm our hypothesis that symmetry has no effect on the computational complexity. We conclude symmetry remains something nice for the eye of the puzzler but does not really make a difference in complexity when examined by the means of SAT-Solvers.

This research does not examine how the complexity for humans corresponds to the computational complexity for SAT-Solvers. Due to the small size of the subsets the distribution of (human) difficulties within each subset could be highly biased. Given the hypothesis that human difficulty of a puzzle corresponds to computational complexity this could additionally distort our results.

For future research it would be interesting to see if our hypothesis still can be confirmed for larger data sets, using different encodings and other types of symmetries .

References

[1] Tjark Weber, A SAT-based Sudoku Solver, Institute for Computer Science, Technische University Munich, Avalible: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.5591&rep=rep1&type=pdf

[2] Y. Mahajan, 'zChaff', Version 2007.3.12. Available: https://www.princeton.edu/~chaff/zchaff.html.

[3] C.P. Gomes, H. Kautz, A. Sabharwal & B. Selman, (2008). 'Satisfiability Solvers'. In F. van Harmelen, V. Lifschitz & B. Porter (Eds.), *Handbook of Knowledge Representation* (pp. 89-100). Elsevier.

[4] QQWing (Version 1.3.4) is Sudoku generating and solving software that has been ported to 3 programming languages: C++, Java, and JavaScript. Avalible: https://qqwing.com/. [Accessed: 6- Sept- 2017]

[5] David Rau, Mirthe Van Diepen, scripts for obtaining the minimal encoding for a 9x9 Sudoku, sorting Sudokus after number of givens: https://github.com/davidmrau/SAT-Solving

[6] Gihwon Kwon and Himanshu Jain, *Optimized CNF Encoding for Sudoku Puzzles*

[7] M. Procope, 'hyperactive', Availible: https://github.com/Procope/hyperactive

A. Appendix

Link to the presentation:
https://youtu.be/qB0EQhn8czs.