SAT solving and dispersion of given numbers in Sudokus

Sarantos Tzortzis
Dennis Ulmer

October 3, 2017

1 Hypothesis

In this project, we explore how the spatial distribution of numbers, especially the degree to which they are concentrated or spread out over the sudoku influences the difficulty for a satisfiability solver solving them. We therefore formulate our hypothesis in the following way:

Hypothesis 1 (H1): *The spatial distribution of given numbers for a Sudoku does not have any impact on the difficulty for a SAT solver solving it.*

Potentially, sudokus could be easier to solve in case the given number lie close to each other and more difficult in the opposite case. This assumption will be tested using a variety of experiments, using metrics to make a quantified statement about the degree of spatial distribution of given numbers in a sudoku, as many sudokus deemed hard by humans seem to correspond to this observation. The code used in this work is freely available on `GitHub`[1]. The presentation of this project has been hosted on `Youtube`[2].

2 Experiments

To provide a foundation for the experiments, the data used will be briefly described in section 2.1. Afterwards, appropriate metrics to measure the difficulty for a SAT solver (section 2.2.1) and to measure the spatial distribution of given numbers (section 2.2.2) are introduced. We then illustrate the way we encoded sudokus as a satisfiability problem (section 2.2.3) before combining dispersion and difficulty in a final measurement testing our hypothesis (2.3).

2.1 Data

For our experiments, we obtained three different data sets from an online sudoku archive[3]. All data sets thereby consist of 9×9 sudokus, either solved or unsolved. More information about the different sets is given in fig. 1.

Name	**Size**	**#Givens**
`10k_25`	10.000	25
`49k_17`	49.151	17

Figure 1: Sudoku data sets used in this work, where **#Givens** denotes the number of given numbers.

Data sets containing sudokus with a low amount of given were chosen by intention to generate expressive results.

[1] https://github.com/Kaleidophon/shiny-robot
[2] https://youtu.be/F7jyUXiT6VM
[3] Available under http://www.printable-sudoku-puzzles.com/wfiles/.

2.2 Metrics

In this section different metrics to measure the difficulty of a sudoku and the spatial distribution of its given numbers will be defined.

2.2.1 Difficulty for SAT solvers

When a satisfiability solver does not have sufficient knowledge to make assign a definitive value to a variable in a logical clause, it has to "gamble" (make a random guess). After this decision, the problem is branched in regard to this condition. If the solver ends up with a contradiction case while trying to resolve a branch (e.g. $\neg x \wedge x$), it must backtrack to the point at which the decision was made and *re-learn* a conflict clause [MDL62].

Following this intuition, we can assume that problems that are hard for SAT solvers will require a higher amount of random variable assignments and therefore, generally speaking, conflicts. This provides a better measure as e.g. runtime, as it isn't dependent on the host machine the solver is run on. Therefore, it was chosen as our metric to measure the difficulty of sudokus.

As we were using `Python 3` to implement the code necessary for this work, we chose `Pycosat`[4] as our SAT solving library, which in turn is just a wrapper for `Picosat` [5] ([Bie08]).

2.2.2 Dispersion metrics

We will now define some metrics to measure the spatial distribution of given numbers in a sudoku, which we will call *dispersion* in the following sections. We will denote the degree of dispersion Δ of a sudoku $\mathcal{S}$ as $\Delta(\mathcal{S})$. Subscript will be used to indicate that the dispersion was measured with a specific metric. We will define a sudoku as a set of cells $\mathcal{S} = \{\kappa_1, \ldots, \kappa_{81}\}$[6]. For convenience, we also define the set of given numbers in a sudoku $\mathcal{S}$ as $\mathcal{S}_g = \{\kappa_n | \forall \kappa_n \in \mathcal{S} \wedge \kappa_n \neq \varnothing\}$ as all the cells of a sudoku wich are not empty and denote them with $\mathcal{S}_g = \{g_1, \ldots, g_N\}$. All κ and g also have a x- and a y-coordinate to specify their position within the grid.

Three metrics have been chosen in this work to assign every sudoku a degree of dispersion:

- Average euclidean distance between all given numbers $\Delta_\varnothing$
- Average distance to the centroid of all given numbers Δ_C
- Shannon entropy of sudoku Δ_H

To compute $\Delta_\varnothing$ for a sudoku $\mathcal{S}$, we first define the distance between two given numbers of a sudoku as the vector norm of the difference between their respective x- and y-components. Given e.g. two given numbers g_a and g_b with $g_a, g_b \in \mathcal{S}_g$, we define

$$dist(g_a, g_b) = ||g_b - g_a||_2 = || \begin{bmatrix} x_{g_b} \\ y_{g_b} \end{bmatrix} - \begin{bmatrix} x_{g_a} \\ y_{g_a} \end{bmatrix} ||_2 = \sqrt{(x_{g_b} - x_{g_a})^2 + (y_{g_b} - y_{g_a})^2} \tag{1}$$

Thereby leading us to the intuitive definition of $\Delta_\varnothing$, where $|\mathcal{S}_g|$ denotes the number of given numbers in sudoku $\mathcal{S}$:

$$\Delta_\varnothing(\mathcal{S}) = \frac{1}{|\mathcal{S}_g|^2} \sum_{g_i, g_j \in \mathcal{S}_g} dist(g_i, g_j) \tag{2}$$

To calculate Δ_C, we first determine the centroid C of the sudoku, which is the average of all the positions of given numbers:

$$c_\mathcal{S} = \frac{1}{|\mathcal{S}_g|} \sum_{g \in \mathcal{S}_g} g \tag{3}$$

Afterwards we calculate the average distance of all given numbers to this centroid:

[4] https://github.com/ContinuumIO/pycosat
[5] http://fmv.jku.at/picosat/
[6] For sudokus with size 9×9.

$$\Delta_C(\mathcal{S}) = \frac{1}{|\mathcal{S}_g|} \sum_{g \in \mathcal{S}_g} dist(g, c_{\mathcal{S}}) \tag{4}$$

For Δ_H, we will use a slightly adjusted version of the standard formulation of Shannon entropy[7]

$$\Delta_H(\mathcal{S}) = -\sum_{\kappa \in \mathcal{S}} p(\kappa) log_2(p(\kappa)) \text{ with} \tag{5}$$

$$p(\kappa) = \begin{cases} p(g|x_\kappa, y_\kappa) & \text{if } \kappa \in \mathcal{S}_g \\ 1 - p(g|x_\kappa, y_\kappa) & \text{if } \kappa \notin \mathcal{S}_g \end{cases} \tag{6}$$

where $p(\kappa)$ denotes the probability of a cell in a sudoku. If the cell is occupied by a given number ($\kappa \in \mathcal{S}_g$), the probability of a given number at this position of the grid ($p(g|x_\kappa, y_\kappa)$) is used and the inverse probability in the opposite case. This probability is calculated by looking at all the sudokus in a data set and dividing the number of times a given number number is found at this specific position of the grid divided by the total number of sudokus.

Intuitively, we can interpret these metrics the following way: As $\Delta_\varnothing$ and Δ_C measure the average distance between all given numbers, the value is expected to decrease in case where all the givens are concentrated close to each other and increase when they are spread out over the sudoku grid. However, Δ_H follows a different intuition, as it follows the degree to which a sudoku is deviating from the expected sudoku based on our observations over whole data set, i.e. the degree of "information" it contains.

Using our metrics defined in section 2.2.2, we can use them to compute the respective value of every sudoku in each data set and plot the distribution.

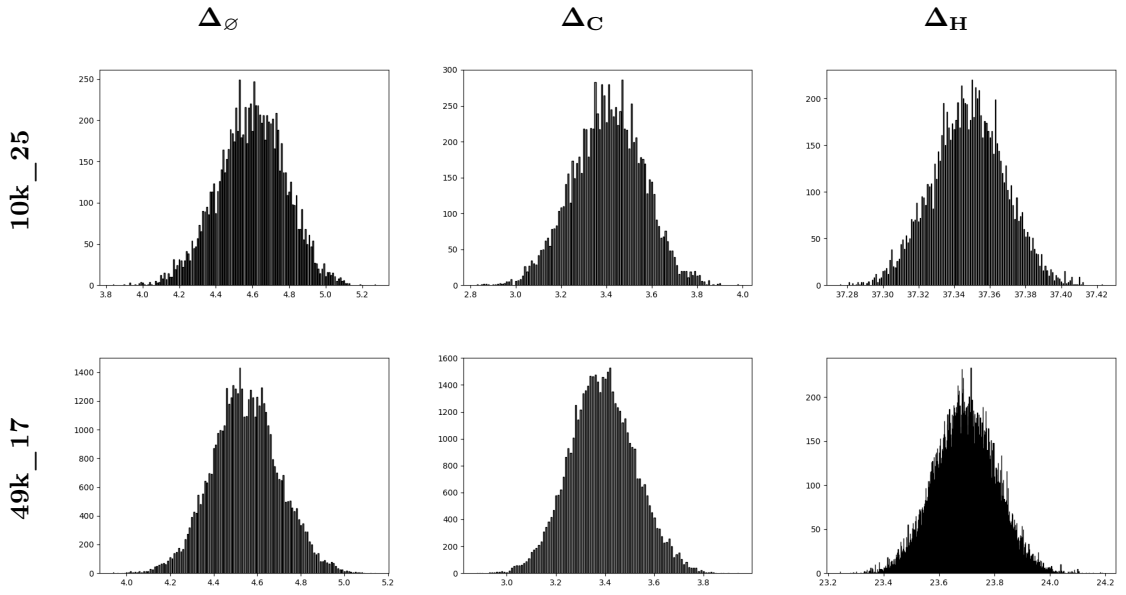

Figure 2: Distribution of values for different dispersion metrics grouped by two decimal points for different data sets. The x-axis denotes the measured value of the metric, the y-axis the number of times this value was observed within the data set.

As apparent in fig. 2, these values are approximately distributed following a normal distribution.

To make sure that the same concept is measured twice in a slightly different way, those distribution were normalized in terms of range and value. Subsequently, the correlation between the

[7] E.g. refer to https://en.wikipedia.org/wiki/Entropy_(information_theory) for further reading.

metrics for the two data set was measured using Spearman's ρ[8]. The results can be seen in appendix A.

Using these correlation values, we can see that regardless of the data set, measuring dispersion using $\Delta_\varnothing$ and Δ_C has a nearly 100 % correlation ($\approx 0.96/0.97$). Therefore we will drop $\Delta_\varnothing$ in the following section and only use Δ_C and Δ_H as our main metrics, as Δ_C is computationally cheaper.

To give an intuition on how spread out or dense the sudokus in our datasets are in regard to the given numbers, three heat maps were created for each dataset and metric. (see fig. 3).
It can immediately be seen how the distribution of given numbers is represented for the lowest ranking sudokus measured with Δ_C for the `10k_25` data set: This seems plausible, as sudokus which are maximizing or minimizing this metric have to either try to "push" the given number out into the corners of the sudoku or collect them into a single part of the grid, respectively. The heat map for all sudokus of a given data set or metric just shows noise, as one could expect.

Interpreting the heat maps for the same data set given the Δ_H metric seems much harder: All three maps seem to show mostly noise. We therefore have to question its value to proof our hypothesis, although we can still keep it in our evaluation as a baseline.

For the `49k_17` data set the heat maps reveal surprising insights: The average grids show a repeating pattern. Finding reasons for this is beyond the scope of this work, but may lie in the number of givens: Potentially, the construction of sudokus with the minimal possible number of givens only leaves room for certain spatial arrangements. This could provide a reason for the more complex patterns that can be spotted for the highest and lowest maps for both metrics. For this bigger data set, Δ_H seems to yield better results, e.g. compare *Lowest 1k* / `49k_17` / Δ_H and *Lowest 1k* / `49k_17` / Δ_C which show some resemblance. This also explains why Pearson's ρ was slightly higher for these data in regard to Δ_C and Δ_H.

2.2.3 Sudoku Encoding

In order to transform a sudoku into a SAT problem, an encoding procedure is necessary. Our research used the *Efficient Encoding* by [Web05], which is faster than the intuitive approach by [LO06]. It contains N^3 variables and $N*N+(N*N*(\frac{N*(N-1)}{2}))*3+k$ clauses[9]. That means a 9×9 sudoku will contain 729 variables and 11775 clauses in total.

In order to convert the problem to CNF, the following rules to create a proper sudoku need to be taken into account. How they are realized in propositional logic is further explained in Appendix B.

- A number appears only once in each cell.
- A number appears only once in each row.
- A number appears only once in each column.
- A number appears only once in each block.

After having encoded the sudoku, it is now possible to feed it to the SAT solver.

2.3 Measuring complexity

After feeding every sudoku in `10k_25` and `49k_17` to the SAT solver and measuring Δ_C and Δ_H for every one of them, we obtain the number of conflicts to get to the solution with respect to their dispersion metric. To disprove our hypothesis, we have assume that there is a (linear or proportional) correlation between those two measurements. In other words: A sudoku scoring a high value using our two metrics is expected to also display a high number of conflicts and vice versa.

[8] Spearman's rank correlation coefficient takes on a value $\rho \in [-1, 1]$, where -1 indicates a strong negative correlation, 1 a positive one and 0 no correlation at all. More information can be found under https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient.

[9] See http://swtv.kaist.ac.kr/courses/cs453-fall12/sudoku.pdf for a more intuitive explanation for this.

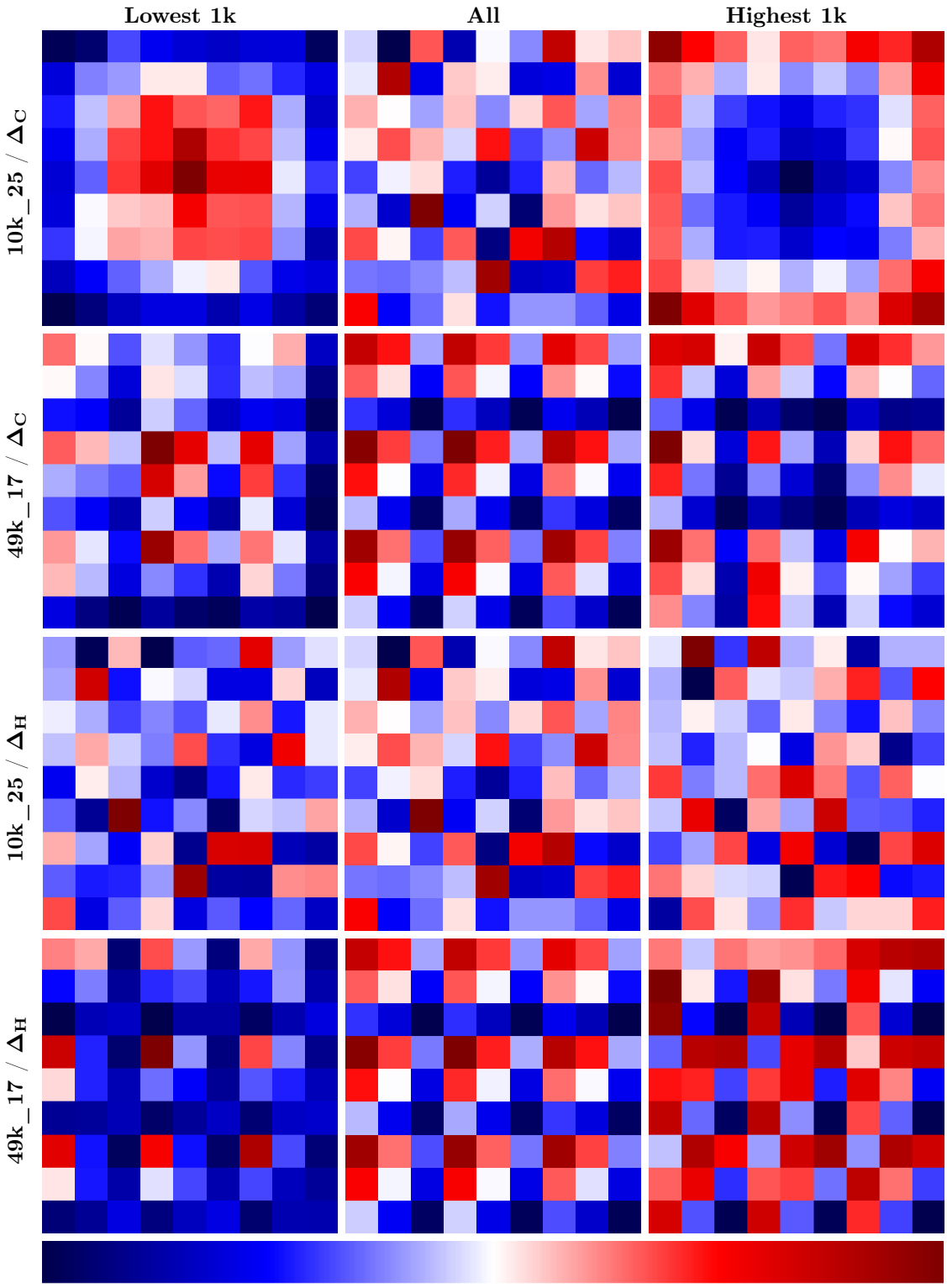

Figure 3: Heat maps depicting the the number of times a given number falls onto a cell of the 9x9 grid. A red color indicates a higher frequency, a blue tone the opposite.

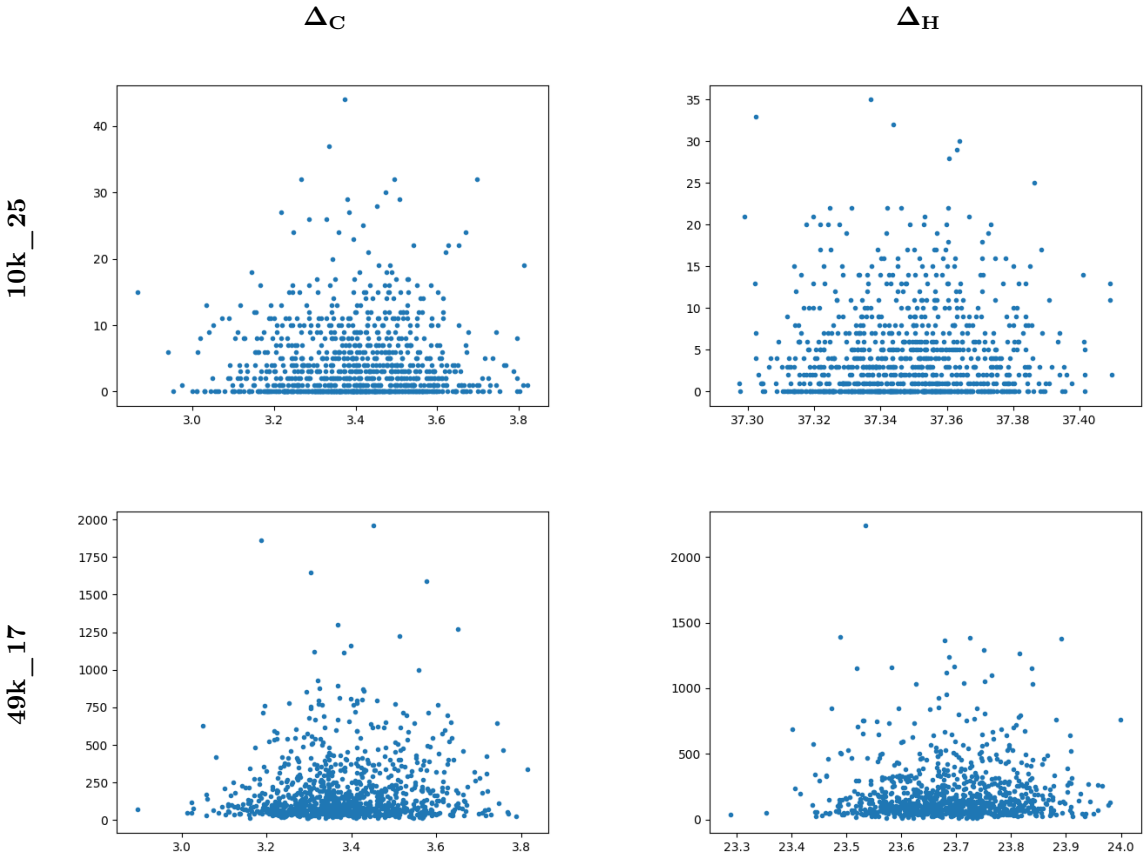

Figure 4: Scatter plots showing 1000 sampled sudokus from each dataset plotted by the number of conflicts using SAT solving and their respective dispersion metric. The x-axis is showing the measured value for the dispersion metric, the y-axis the observed number of conflicts.

However, the results draw a different picture. As shown in fig. 4, it is easy to observe that the number of conflicts fluctuates highly and more importantly independently from the dispersion value for both metrics. This renders the initial **hypothesis** to be true.

3 Discussion

The results from section 2.3 in form of fig. 4 clearly indicate that the degree of dispersion does not show any correlation with the difficulty solving them. This could have multiple reasons:

- *There truly is no correlation between dispersion and difficulty for SAT solvers.* While this is certainly a possibility, it is hard to definitively prove or disprove this statement in the scope of this work just based on *our* experimental set up. The question is whether the position of a number matters at all if they merely appear as unit clauses inside a conjunctive normal form, for which the rule of commutativity applies.
- *Our dispersion metrics are insufficient or not expressive enough to capture correlation.* The dispersion metrics didn't take the actual numeric value of every given number into account, which may have provided additional insights.
- *The number of conflicts is an insufficient measure to make a statement of the difficulty a SAT solver encounters solving the sudoku.* Although the number of conflicts seems to be an intuitive metric, further measures like CPU time, number of restarts and learned clauses could be considered. Also, the number of conflicts could be measured multiple times and then averaged per sudoku in case the SAT solver uses non-deterministic heuristics.

- *The correlation is not detectable in* 9×9 *sudokus*: Potentially, a correlation exists, but only is apparent in sudokus of bigger scale.

4 Summary

In this work, we apply two different metrics, namely the average distance of given numbers of a sudoku to their *centroid* as well as the sudoku's *Shannon entropy* to two data sets of 9×9 sudokus with 17 and 25 given numbers per grid. Observing the number of conflicts a satisfiability solver encounters solving this sudokus encoded by the *Efficient Encoding* of [Web05], we conclude that the spatial distribution of given numbers has no influence on their difficulty for a solver in our experimental setup.

The reasons for this outcome could, among others, lie in insufficient metrics used in this work, the phenomenon having a minuscule effect on 9×9 sudokus or simply the absence of such correlation entirely.

5 Future Work

Future research questions building on top of our results could lead into the following directions:

- *Generate Sudoku with intentionally extreme values given dispersion metrics*: The idea is to generate extremes cases of sudokus - based on the degree of the spatial distribution - so we can repeat the experiment compensating for the lack of data at the corners of our distributions back in fig. 2[10].
- *Entropy as a metric for dispersion*: Entropy seems to have captured at least some information in regard to dispersion, looking at the `49k_17` dataset in fig. 3. If these regularities arise from the number of givens, the size of the data set or an unconsidered factor is unknown at this point.
- *Including the numeric value of given numbers*: This work focused merely on the position of given numbers within in the sudoku, not their respective values. Extending our dispersion metrics to also capture this information may change the conclusion reached.
- *Behavior for bigger sudokus*: The data sets utilized in this project were all comprised of standard 9×9 sudokus. It could have been the case that dispersion indeed *does* play a role in difficulty for SAT solver, but that these effects are negligeable on our scale.

References

[Bie08] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.

[LO06] Inês Lynce and Joël Ouaknine. Sudoku as a sat problem. In *ISAIM*, 2006.

[MDL62] G. Logemann M. Davis and D. Loveland. A machine program for theorem proving. *Comm. ACM*, page 5:394–397, 1962.

[Web05] T. Weber. A sat-based sudoku solver. *The Proceedings of LPAR'05*, 2005.

[10] Note: Within the scope of this work, efforts were put into the conception of such algorithm, where based on already solved sudokus numbers were iteratively removed s.t. they maximized or minimized one of our metrics. After each step, the resulting sudoku was tested to still be proper. As it turns out, this seems to be a challenging problem, as removing the given numbers in regard to the dispersion metrics seems to quickly lead to improper sudokus.

A Correlation between dispersion metrics on data sets

		$\Delta_{\varnothing}$	$\Delta_{\mathbf{C}}$	$\Delta_{\mathbf{H}}$
$\Delta_{\varnothing}$	10k_25	-	0.96	0.08
	49k_17		0.97	0.42
$\Delta_{\mathbf{C}}$	10k_25	0.96	-	0.09
	49k_17	0.97		0.35
$\Delta_{\mathbf{H}}$	10k_25	0.08	0.09	-
	49k_17	0.42	0.35	

Figure 5: Correlation between metrics for both the 10k_25 and 49k_17 dataset, measured for the normalized distributions of metric values using Spearman's ρ.

B Efficient encoding of sudokus as satisfiability problems

In order to access a cell, a 3-tuple is used, where (r, c, v) stands for the $cell[r, c]$ with value v.

Cell Rule

- At *least* one number in each cell (*definedness*)
  $Cell_d = \wedge_{r=1}^{N} \wedge_{c=1}^{N} \wedge_{v=1}^{N} (r, c, v)$
- At *most* one number in each cell (*definedness*)
  $Cell_u = \wedge_{r=1}^{N} \wedge_{c=1}^{N} \wedge_{v_i=1}^{N} \wedge_{v_j=v_i+1}^{N} \neg((r, c, v_i) \wedge (r, c, v_j))$

Row Rule

- Each number at *least* one in each row (*definedness*)
  $Row_d = \wedge_{r=1}^{N} \wedge_{c=1}^{N} \wedge_{v=1}^{N} (r, c, v)$
- Each number at *most* one in each row (*definedness*)
  $Row_u = \wedge_{r=1}^{N} \wedge_{c=1}^{N} \wedge_{v_i=1}^{N} \wedge_{v_j=v_i+1}^{N} \neg((r, c, v_i) \wedge (r, c, v_j))$

Column Rule

- Each number at *least* one in each column (*definedness*)
  $Col_d = \wedge_{r=1}^{N} \wedge_{c=1}^{N} \wedge_{v=1}^{N} (r, c, v)$
- Each number at *most* one in each column (*definedness*)
  $Col_u = \wedge_{r=1}^{N} \wedge_{c=1}^{N} \wedge_{v_i=1}^{N} \wedge_{v_j=v_i+1}^{N} \neg((r, c, v_i) \wedge (r, c, v_j))$

Block Rule

- Each number at *least* once in each block(*definedness*)
  $Block_d = \wedge_{r_{offs}=1}^{subN} \wedge_{c_{offs}=1}^{subN} \wedge_{v_{offs}=1}^{N} \vee_{r=1}^{subN} \vee_{c=1}^{subN} (r_{offs} * subN + r, c_{offs} * subN + c, v)$
- Each number at *most* once in each block(*uniqueness*)
  $\wedge_{r_{offs}=1}^{subN} \wedge_{c_{offs}=1}^{subN} \wedge_{v_{offs}=1}^{N} \wedge_{r=1}^{N} \wedge_{c=r+1}^{N}$
  $\neg((r_{offs} * subN + (rmodsubN), c_{offs} * subN + (rmodsubN), v)$
  $\wedge(r_{offs} * subN + (rmodsubN), c_{offs} * subN + (rmodsubN), v))$

Given Numbers

- An initial given number is actually a constant.Thus, it is a unit clause.
  $GivenNo = \wedge_{i=1}^{k} (r, c, g) | \exists_{1 \leq a \leq N} \bullet [r, c] = g$, where k the amount of given numbers.

Figure 6: Rules utilized to encode a sudoku into a satisfiability problem, following the *Efficient Encoding* of [Web05].