Efficiency of popular DPLL heuristics for increasing N^2 size Sudoku's

Demi Kruijer, Rick van Slobbe, and Seth van der Bijl

Vrije Universiteit, Amsterdam

1 Introduction

The problem statement of the present research around Boolean satisfiability problems (SAT) situates itself in the field of constraint satisfaction problems. This active area of research involves itself in the theoretical complexity and practical solving of problems in propositional form. A practical instance of this SAT problem is the satisfaction of a Sudoku. Sudoku's have the objective to fill a $n^2 \times n^2$ grid with numbers 1 to n, so that each row, column and each of the n different $n \times n$ subgrids contain every number only once. The player is provided with a partially filled grid that requires one arrangement of the numbers to solve the puzzle.

While simple, Sudoku provides an excellent practical instance of a SAT problem. A host of different techniques for solving these problems has come to existence in the last few decennia with the DPLL-algorithm as one of the classic solutions. The vanilla DPLL can be integrated with custom variable selection heuristics. The present research attempts to determine the efficiency of different heuristics in the DPLL algorithm for solving the particular SAT-problem of Sudoku. The size of the SAT problem, in this case the size of the Sudoku grid, might affect which heuristics is most efficient in solving that particular SAT problem.

The following research question originates from this problem-statement: *What is the effect of varying Sudoku sizes on the efficiency of popular branching heuristics implemented in a DPLL-algorithm?* Efficiency will be determined in terms of running time, number of backtracks and number of decisions taken by the algorithm before finding a solution. 6 different methods for variable selection will be considered. In light of this research question we hypothesize that the differences in efficiency found between the different heuristics generalize to different Sudoku sizes.

2 Background

The background history and state of the art pertaining to the research in the present paper can be divided in three categories.

Firstly, the Sudoku problem is an instance of a constraint satisfaction problem (CSP). CSPs are problems of combinatorial nature where a set of constraints must be satisfied. The main formalization and pioneering work was done by Waltz [3] and Montanari. [4] [6] The boolean satisfiability problem (SAT) is a particular category of constraint satisfaction problems.

Secondly, DPLL-solvers are a branch of algorithms attempting to solve satisfiability problems. Being introduced by Davis-Putnam as a relatively fast SAT-solver in practice[1] the DPLL [2] proposed by Davis, Putnam, Logeman and Loveland is an algorithm that has proven to be a complete, sound and a performance-efficient algorithm in practice for establishing the satisfiability of propositional formulas in clausal normal from (CNF). The DPLL algorithm does this through simplification, variable selection and backtracking. This allows for fast determinations of the satisfiability of a CNF formula. Some of the most powerful variations of DPLL algorithms employ clause learning [7] or applying knowledge to the SMT domain. [11] Sudoku itself has been researched as SAT instance.[5][12][9][10][13]

3 Methods

3.1 The DPLL-algorithm

As discussed in the background in research into the DPLL-algorithm our implementation of this algorithm followed the original DPLL algorithm with addition of various heuristics. The various elements of the DPLL algorithm are listed in order below.
1. Assigning true to all clauses containing only one literal and all other clauses in which this literal is present.
2. Assigning true or false to all literals which are only present in the non negated or negated form respectively. These literals are called 'pure' literals.
3. If the formula is not satisfied after the simplification, a variable is selected by a heuristic to be assigned True. If the true assignment causes an unsatisfiable formula, its negated form will be tried.
4. Backtracking is applied when the currently assigned literals cause an unsatisfied formula, but not all literal assignments are explored yet. The algorithm will backtrack to the last literal chosen by the heuristic and branch upon the opposite assignment. Backtracking returns 'unsatisfiable' when both assignments of all literals has been tried without finding a satisfied formula, or 'satisfiable' when a set of literal assignments has been found which satisfies the formula.

3.2 Propositionalizing Sudoku and rules

Both the starting set-up of the Sudoku being solved and the rules for solving it have to be propositionalized by putting them in dimacs terms in conjunctive normal form.

Encoding to dimacs The starting positions of the given Sudoku's were encoded by simply creating an integer of the x and y coordinate of the cell in the Sudoku and a direct mapping of the value. Employing this system a 5 on the 2nd row of the 3rd column was encoded as 235. The negation of any variable was encoded by taking the negative of the integer: -111 explicating that the number 1 is not the number in the first row in the first column of the Sudoku. For $n = 4$ and $n = 9$ size Sudoku's these dimac encodings were given.

If different propositional variables map to the same dimacs variable this is disastrous for the resulting formula and causes inconsistencies. Noticeably, converting $n = 16$ Sudoku's to dimacs format requires one to either encode them as hexadecimal dimacs variables or as integer dimacs variables of a length of minimum $3 \times \lceil \log_{10}(n+1) \rceil$. This denotes that when encoding $n = 16$ size Sudoku variables to dimacs the dimacs terms need to be at least $6 = 3 \times 2$ digits long, since the numeral 16 is two digits long and as such requires two digits for both row, column and value to be encoded correctly when not encoding in hexadecimals. For example, encoding the value 11 in the first row of the first column maps to the same dimacs integer as encoding the value 1 in the 11th row of the first column, both map to 1111. This illustrates that clashes happen when plainly mapping $n > 9$ Sudoku's without padding. The variables can be padded with the symbol 9 to make it $\lceil \log_{10}(n+1) \rceil = 2$.[1] This padding only happens when it is necessary. For example, the 11th row maps to 11 where the first, second and fifth row map to respectively 91, 92 and 95. This system ensures no collisions between the dimacs variables generated from actually different propositions occur.

Arranging dimacs terms in a CNF The terms generated have to be arranged in such an order that they correctly denote the rules for filling in a Sudoku and the set-up of the numbers of a given Sudoku. For $n = 4$ and $n = 9$ sized Sudoku's both the dimacs-encodings and their CNF-arrangement where given. The Sudoku's were translated from the given dots format into the aforementioned dimacs format.

Similar to the propositionalizing of the $n = 16$ puzzles, the rules also had to be padded with 9s in order to be read correctly. As such, the rules where generated by explicating several paradigms. Every cell in the Sudoku had a number (could not be empty), for example, $919191 \vee 919192 \vee 919193 \vee 919194 \vee 919199 \vee 919110 \vee 919111$, *etc.* for every cell. Furthermore, every cell only has one number and could not contain more numbers for example $\neg 919191 \vee \neg 919192$ for every cell for every value. Consequently every number could appear only once in every row and every column, respectively denoted by iterations of for example, $\neg 919191 \vee \neg 919291$ and $\neg 919191 \vee \neg 929191$ for every number for every column or row. Finally every number could appear only once in it's local area of size $n \times n$.[2]

3.3 Heuristics for variable selection

To analyze the efficiency of different heuristics on the number of backtracks performed, decisions made and on the running time for Sudoku's of varying sizes, multiple heuristics have been implemented. These heuristics vary in complexity and range from being all-purpose heuristics to Sudoku-specific variable selection.

[1] Padding with a zero would lead to numbers like 021406 which when read as an integer is read as 21406 and loses the zero, leading to information loss and non-injectivity.

[2] That is, for regular-shaped subareas. So Sudoku's could only be $n = 4$, $n = 9$, $n = 16$ or $n = 25$ and for example, not $n = 10$.

Random variable selection Random variable selection is implemented by selecting a random clause from the CNF-formula and selecting a random variable from this clause This method serves as an initial baseline to compare the other heuristics against. It is noted that this mechanic causes variables in long clauses to have a very small chance to be selected. Furthermore, one notes that in the specific CNF for Sudoku's the chance to select a negative variable is multitudes higher than the chance to select a positive variable.

Absolute random variable selection Considering the shortcomings of random variable selection, namely the tendency to select a negative variable and the small selection chance for variables from longer clauses we implemented an absolute version of the random variable. This heuristic returns the positive version of a random variable. This is hypothesized to benefit the search process since in the case of Sudoku's, positive variables are more 'information rich' than negative variables (i.e. saying number 5 is not in coordinate 4,8 does not give very much information).

Jeroslow-Wang heuristic The main principle of the two sided Jeroslow-Wang heuristic is that variables present in shorter clauses are considered more important compared to variables present in larger clauses. This heuristic assigns weights to all variables according to the formula below.

$$J(l) = \sum_{l \in \omega, \omega \in \varphi} 2^{-|\omega|}$$

A score J is assigned to every variable, by adding a value of $2^{-|\omega|}$ to its score for every clause it appears in, where ω represents the length of the clause. The variable with the highest score is selected to be branched upon. This behavior has been called a weighted branching rule (Xing, Z Zhang, W. 2004).

MOM's heuristic For the MOM's heuristic, short for Maximum Occurrences in clauses of Minimal size, selection is only based upon the clauses of the smallest size in the current state of the CNF. Occurrences of literals are counted only if they appear in the clauses of minimal length. The variable that maximizes the formula below is selected.

$$S(x) = (f(x) + f(x')) * 2^k + f(x) * f(x')$$

Here, $f(x)$ represents the number of occurrences of variable x in the smallest non-satisfied clauses, $f(x')$ represents the number of occurrences of its negated form and k is a tuning parameter. The parameter k determines the dependence of the score on the sum of the two occurrences, $f(x) + f(x')$, relative to the dependence on the product of the occurrences, $f(x) * f(x')$. For a higher value of k, the score depends more strongly on the sum, while for $k = 0$, the score equally depends on both the sum and the product. The value of the product is higher for literals for which the occurrences of x approximately balance the occurrences of x'. Therefore, literals that meet this requirement are advantaged for low values of k. The value of k was set to $k = 2$ to minimize this advantage.

The MOM's heuristic was included in the experiments, because it is unique in only focusing on the clauses of the shortest length, eliminating a large part of the search space related to variable selection. This property makes this heuristic an efficient method for variable selection and relatively low in computational costs.

Shortest-pos The shortest-pos heuristic for variable selection is also based on the principle that variables present in shorter clauses are more important. However, this heuristic combines this principle with the idea that positive literals contain more information compared to negative literals, as discussed in the absolute random variable selection section. First the shortest clause with all positive variables is found, the first variable within this clause is then selected to be branched upon.

Longest-pos Finally, while variables of the longest clauses are considered to be less functional for general CNF-problems for the Sudoku rules encoding we know that the long clauses consist of variables stating that every cell should have any number. While the choice of the variables from these clauses only renders a very long clause true, it interestingly converts a large number of shorter clauses to unit clauses. This in relation to the information richness. The longest-pos heuristic selects a random variable from the longest clause to branch upon.

3.4 Experimental design

Six different heuristics are implemented in a DPLL algorithm and used to solve 50 Sudoku's with grid sizes of 4x4, 9x9 and 16x16. Performance of the heuristics is measured as time to completely solve the Sudoku's, and number of backtracking and decision steps required to find the complete solution to the Sudoku's. The collection of Sudoku puzzles was provided by S. Schlobach from the Vrije Universiteit as part of the Knowledge Representation course. From this collection, 50 random 4x4 and 9x9 puzzles are selected to be tested upon. The DPLL algorithm experiences difficulty when trying to solve the 16x16 puzzles with any heuristic, therefore these puzzles are modified as to contain a 60% filled in grid.

4 Results

The results of the experiments are depicted in the boxplots in figure 1. Every row shows the results for a particular Sudoku size, ranging from 4×4 to 16×16 and every column shows a different performance measure. Performance is measured in terms of running time, number of backtracking and decision steps required to find the complete solution to the Sudoku's.

A one-way ANOVA test is used to test for significant differences between groups. This is done between all heuristics, within groups 9 different groups determined by size and measure. P smaller than 0.05 is considered statistically significant.

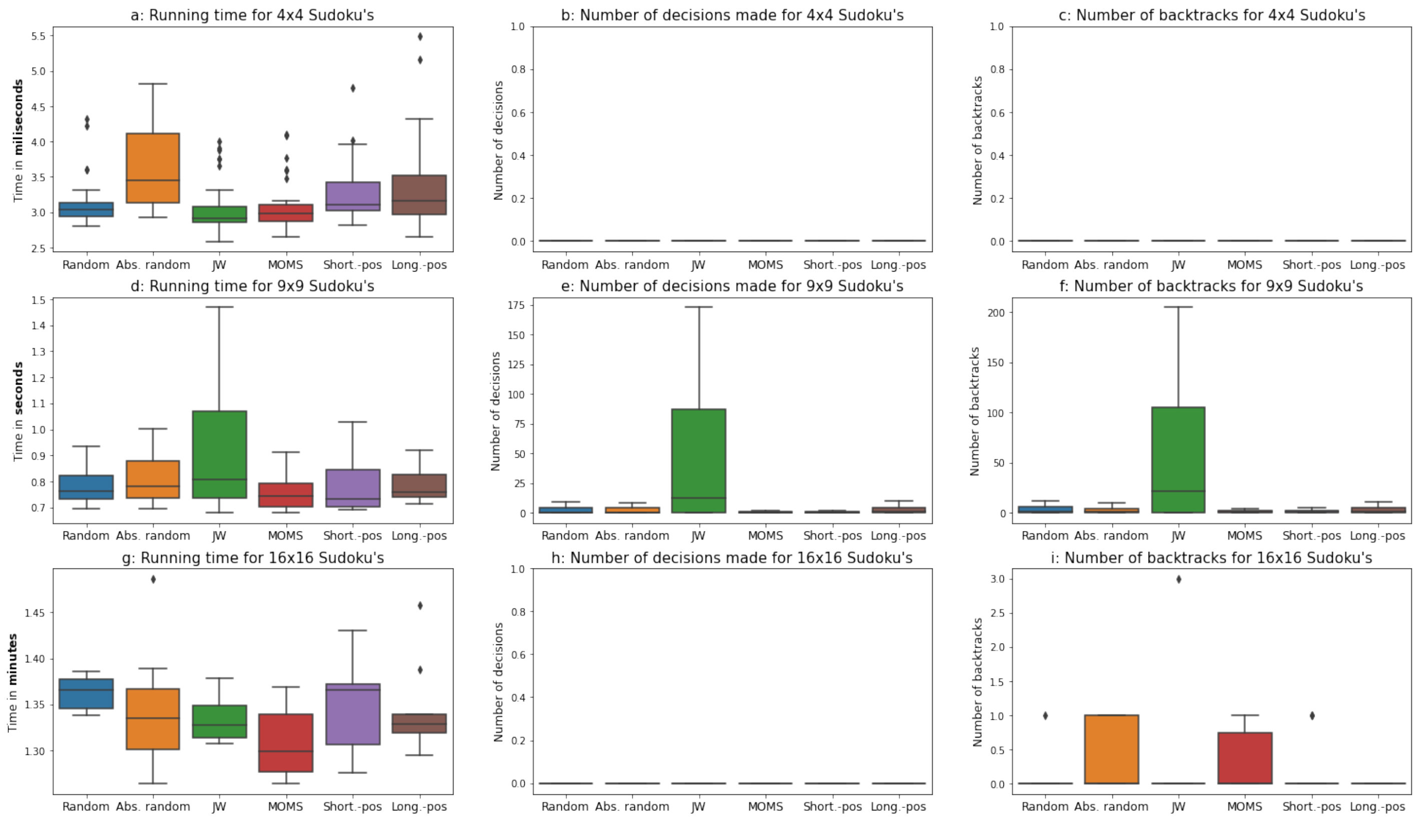

Fig. 1. Boxplots depicting the results on the efficiency of different heuristics

4.1 4×4 Sudoku's

Figure 1a represents the running time for the experiments on the 4×4 Sudoku's in milliseconds. The method of random variable selection, as well as the MOM's heuristic are significantly faster than the random absolute version, the shortest-pos and longest-pos. The random absolute method for variable selection runs significantly slower than all other heuristics. Speed increases for the Jeroslow-Wang heurisic, which runs significantly quicker than the MOM's, longest-pos and random absolute methods on 4×4 Sudoku's. Both the shortest-pos and the longest-pos heuristic are significantly slower than Jeroslow-Wang, MOM's and random and perform significantly faster than the random absolute method. Figure 1b and 1c show that all heuristics did not perform any backtracking and also needed zero decisions to find a solution to the puzzles.

4.2 9×9 Sudoku's

For all three boxplots on the 9×9 Sudoku's, the outliers are hidden to make the plots more readable but included in the mentioned averages. The boxplot in figure 1d displays the results on the running time for these Sudoku's in seconds. The method of random variable selection is significantly faster than the Jeroslow-Wang heuristic for 9×9 Sudoku's. The Jeroslow-Wang has the slowest mean performance in this category and runs significantly slower than random, MOM's and the shortest-pos heuristic. MOM's performs significantly quicker than the Jeroslow-Wang heuristic. The shortest-pos was also significantly quicker than the Jeroslow-Wang heuristic. For random absolute and the longest-pos heuristic, no significant differences were found for running time as compared to other heuristics.

For the number of decisions made to reach a solution of the 9×9 Sudoku's, several significant differences are found as well. It is easy to read from figure 1e that Jeroslow-Wang makes most decisions. It takes on average 150 decisions before reaching a solution and has a significantly higher score than all other heuristics. Both the random and random absolute (6 and 5 decisions respectively) methods made significantly less decisions than Jeroslow-Wang, but significantly more than shortest-pos (1 decision). MOM's (2 decisions) made significantly less decisions than Jeroslow-Wang (150 decisions) and the longest-pos heuristic (12 decisions). The shortest-pos heuristic made significantly less decisions, namely only 1 on average, with respect to all heuristics but MOM's. The longest-pos heuristic (12 decisions) made significantly less decisions than Jeroslow-Wang (150 decisions) and significantly more than MOM's and shortest-pos (2 and 1 decisions respectively. The significance relationships of the number of backtracks, shown in figure 1f, corresponds to those of the number of decisions for the 9×9 Sudoku's, with the number of backtracks being approximately equal to the number of decisions. There was one additional significant difference: the MOM's made significantly more backtracks than the random method.

4.3 16×16 Sudoku's

For the running time of the largest size Sudoku's, only two significant differences are found. The distribution of those running times are displayed in figure 1g. For 16×16 Sudoku's, the method of random variable selection is found to be significantly slower than the Jeroslow-Wang and MOM's heuristics. For the number of decisions, no significant differences are found. The random absolute method takes significantly more decisions than the longest-pos heuristic. This is the only significant difference found for the number of decision steps required to find a solution to the 16×16 Sudoku's.

4.4 Qualitative analysis

For the 4×4 Sudoku's, Jeroslow-Wang, MOM's and random are all significantly faster than the other heuristics. These three methods can therefore be judged as the best performing methods for variable selection for 4×4 Sudoku's. The method of random absolute variable selection is the absolute worst performing algorithm for this Sudoku size.

However, Jeroslow-Wang is clearly performing worst in terms of efficiency for 9×9 Sudoku's, as it is the slowest option and it has by far the most decisions and backtracks. A clear winner is harder to spot in this category, but the shortest-pos heuristic is best performing in terms of backtracks and decisions, taking significantly less than all heuristics, except MOM's.

For the 16×16 Sudoku's, a best performing heuristic is also hard to distinguish, although it is clear that the MOM's and Jeroslow-Wang heuristic perform significantly better than just random variable selection in terms of running time.

5 Discussion

The pattern of Jeroslow-Wang heuristic occasionally producing large outlying numbers of variable assignments, backtracks and solving times is notably obvious for 9×9 Sudoku's which required significantly more numbers of variable assignments. Besides the means being significantly higher are also the worst-case scenario metrics magnitudes higher. The cause of this is suspected to be found by the interaction of the heuristic with the encoding of the Sudoku rules. A vast majority of the shortest Sudoku rules clauses contains only negative literals and negative literals make up most of the literals in the rules by far. As discussed before, negative literals do not bring the same 'information-richness' to the current exploration of the solution as positive literals do.

Future research would benefit from a continued[8] [5] analysis and description of the clausal structure of a particular problem and the corresponding performance of certain heuristics. This in connection with the asymmetric properties of the clauses for this problem (many short fully negative clauses and a few very long fully positive clauses) and the subsequent effectiveness certain heuristics appear to exhibit over others.

The present research indicates that different heuristics might perform better or worse depending on the size of the problem. An interesting implementation of this could be to utilize multiple heuristics at different stages of the search, starting off with one heuristic and switching to another once the problem size has been reduced sufficiently.

6 Conclusion

The aim of this paper was to evaluate the performance of popular all-purpose and Sudoku-specific branching heuristics within a DPLL algorithm on Sudoku puzzles of varying sizes. The complexity of solving Sudoku puzzles increases exponentially as the size of the grid goes up, which might favor certain heuristics more so. The results show that Jeroslow-Wang yields good results for 4×4 and 16×16 Sudoku's, but performs worst for 9×9 Sudoku's. MOM's yield promising performances for all sizes. On the contrary, the random method and shortest-pos heuristic turn out to be very efficient for respectively 4×4 and 9×9 Sudoku's, while yielding mediocre results for 16×16 Sudoku sizes. These findings are in line with our hypothesis that certain heuristics may be more efficient in solving constraint satisfaction problems of different sizes.

References

[1] Martin Davis and Hilary Putnam. "A Computing Procedure for Quantification Theory". In: *Journal of the ACM* 7.3 (July 1960), pp. 201–215. ISSN: 0004-5411. DOI: 10.1145/321033.321034. URL: https://doi.org/10.1145/321033.321034 (visited on 11/24/2021).

[2] Martin Davis, George Logemann, and Donald Loveland. "A machine program for theorem-proving". In: *Communications of the ACM* 5.7 (July 1962), pp. 394–397. ISSN: 0001-0782. DOI: 10.1145/368273.368557. URL: https://doi.org/10.1145/368273.368557 (visited on 11/24/2021).

[3] David L. Waltz. "Generating Semantic Descriptions From Drawings of Scenes With Shadows". en_US. In: (Nov. 1972). Accepted: 2004-10-20T20:06:16Z. URL: https://dspace.mit.edu/handle/1721.1/6911 (visited on 11/27/2021).

[4] Ugo Montanari. "Networks of constraints: Fundamental properties and applications to picture processing". en. In: *Information Sciences* 7 (Jan. 1974), pp. 95–132. ISSN: 0020-0255. DOI: 10.1016/0020-0255(74)90008-5. URL: https://www.sciencedirect.com/science/article/pii/0020025574900085 (visited on 11/27/2021).

[5] David Mitchell, Bart Selman, and Hector Levesque. "Hard and Easy Distributions of SAT Problems". In: 1992, pp. 459–465.

[6] Sally C. Brailsford, Chris N. Potts, and Barbara M. Smith. "Constraint satisfaction problems: Algorithms and applications". en. In: *European Journal of Operational Research* 119.3 (Dec. 1999), pp. 557–581. ISSN: 0377-2217. DOI: 10.1016/S0377-2217(98)00364-6. URL: https://www.sciencedirect.com/science/article/pii/S0377221798003646 (visited on 11/27/2021).

[7] Lawrence Ryan. *Efficient Algorithms for Clause-Learning Sat Solvers*. 2004.

[8] Yacine Boufkhad et al. "Regular Random k-SAT: Properties of Balanced Formulas". en. In: *Journal of Automated Reasoning* 35.1 (Oct. 2005), pp. 181–200. ISSN: 1573-0670. DOI: 10.1007/s10817-005-9012-z. URL: https://doi.org/10.1007/s10817-005-9012-z (visited on 11/27/2021).

[9] Laurent Théry. "Sudoku in Coq". en. report. INRIA Sophia Antipolis - Méditerranée, Feb. 2006. URL: https://hal.inria.fr/hal-03277886 (visited on 11/27/2021).

[10] Gi-Hwon Kwon. "Optimized Encoding of Sudoku Puzzle for SAT Solvers". kor. In: *Journal of KIISE:Software and Applications* 34.7 (2007). Publisher: Korean Institute of Information Scientists and Engineers, pp. 616–624. ISSN: 1229-6848. URL: https://www.koreascience.or.kr/article/JAKO200734513400911.page (visited on 11/27/2021).

[11] Leonardo de Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver". en. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 337–340. ISBN: 978-3-540-78800-3. DOI: 10.1007/978-3-540-78800-3_24.

[12] Martin Henz and Hoang-Minh Truong. "SudokuSat—A Tool for Analyzing Difficult Sudoku Puzzles". en. In: *Tools and Applications with Artificial*

Intelligence. Ed. by Constantinos Koutsojannis and Spiros Sirmakessis. Studies in Computational Intelligence. Berlin, Heidelberg: Springer, 2009, pp. 25–35. ISBN: 978-3-540-88069-1. DOI: 10.1007/978-3-540-88069-1_3. URL: https://doi.org/10.1007/978-3-540-88069-1_3 (visited on 11/27/2021).

[13] David Eppstein. "Solving Single-Digit Sudoku Subproblems". en. In: *Fun with Algorithms*. Ed. by Evangelos Kranakis, Danny Krizanc, and Flaminia Luccio. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 142–153. ISBN: 978-3-642-30347-0. DOI: 10.1007/978-3-642-30347-0_16.