Solving Sudokus like the Experts: Augmenting DPLL with Problem-Specific Heuristics

Kim de Bie
2656827

1 Introduction

The Davis-Putnam-Logemann-Loveland (DPLL) algorithm, which provides a strategy for solving SAT problems, is agnostic to the nature of the problem it is solving [2]. Existing literature on DPLL mostly proposes improvements that aim to perform well across the spectrum of SAT problems [7], and such general improvements seem to be favoured over approaches explicitly tailored to specific problems. However, it seems likely that the structure of SAT problems may have a large impact on the performance of different heuristics. Newer methods, such as conflict-driven clause learning (CDCL) [7], implicitly address this by augmenting DPLL with the ability to learn from its own mistakes. However, this approach to exploiting problem structure to solve SAT problems efficiently is fairly indirect, as this structural information is only implicitly learned during runtime. In addition, it seems likely that implicit methods such as CDCL fail to capture some relevant structural information (for example, the selection of a new variable to assign is still performed by a general heuristic). Therefore, this paper hypothesizes that the structure of a SAT problem can *a priori* provide information about the applicability of different solving strategies.

To prove that problem structure matters for performance, and that problem-specific strategies may therefore outperform general heuristics, this paper focuses on SAT problems that are instances of Sudoku puzzles (encoded following the extended encoding proposed by [5]). A DPLL implementation augmented with a problem-specific heuristic, *Nishio*, is proposed. *Nishio* is shown to significantly outperform a DPLL solver with random branching on Sudoku problems, as well as DPLL augmented with the MOMS [3] and Jeroslow-Wang heuristic [4].

Lastly, the differential performance of the heuristics within the set of Sudokus is investigated. It is shown that subsets of puzzles which are relatively hard under one heuristic may be among the easier puzzles for another heuristic, which again indicates that the structure of a problem matters to the optimal solving strategy.

2 The DPLL algorithm for solving SAT problems

DPLL generally consists of three simplification steps [2]: the removal of *tautological clauses*, as they are always true; the identification of *pure literals*, which only occur in either positive or negated form, and may therefore be assigned without any risk; and lastly the identification of *unit clauses*, whose literal must be set to true. When no further simplifications can be made, the algorithm assigns a new literal and then recursively calls itself, first attempting to simplify (albeit the tautology step is skipped in consecutive iterations) and then assigning new literals again - and backtracking if this assignment leads to conflicts. The algorithm terminates when all clauses are satisfied, or when it encounters an empty clause that cannot be undone (indicating that the problem is unsatisfiable).

The algorithm is implemented in Python. Despite using a simple data structure (the CNF formula is a list of sets and no counters are kept), the algorithm performs reasonably fast, and solving the Sudokus is a matter of a few seconds for nearly all puzzles, using regular consumer hardware.

3 Heuristics for the DPLL algorithm

The outlined DPLL algorithm is ambiguous with respect to its branching strategy, i.e. the selection of a new literal to assign. The baseline strategy is to select this literal randomly, but it is obvious that improvement of the selection strategy may aid the performance of the algorithm. An optimal branching heuristic helps to find a short path to the solution (for satisfiable problems) or reaches conflicts quickly (for unsatisfiable problems),

and must be reasonably cheap to compute. Three branching heuristics are implemented: MOMS heuristic [3] and (two variations of) the Jeroslow-Wang heuristic [4], which are generally reported to outperform the random branching strategy, and *Nishio*, a heuristic based on a strategy commonly used by human Sudoku experts.

3.1 General heuristic: MOMS

The Maximum Occurrence in clauses of Minimal Size (MOMS) heuristic [3] selects the most-occurring literal from the shortest clauses only. Intuitively, assigning such literals may lead to a large number of unit literals, and a consecutive cascading effect. As such, MOMS balances concerns for selecting a literal that occurs often (as such a literal will affect the largest number of clauses and thus lead to solutions or failures most quickly), and selecting a literal that occurs in the shortest clauses (as these can reveal contradictions and enable unit propagations quickly).

3.2 General heuristic: Jeroslow-Wang One- and Two-Sided

The Jeroslow-Wang heuristic [4] has a similar intuition to MOMS heuristic, but looks at all clauses instead of the shortest only. Literal scores are weighted: a larger weight is assigned to a literal if it occurs in smaller clauses. In the one-sided variant, Jeroslow-Wang treats a literal and its negated version as two distinct literals, simply selecting and assigning the most frequently occurring one. In the two-sided version, the total score for the literal and its negation determines which literal gets selected (but the most-frequently occurring polarity still is assigned).

3.3 Problem-based heuristic: Nishio for fewest-options

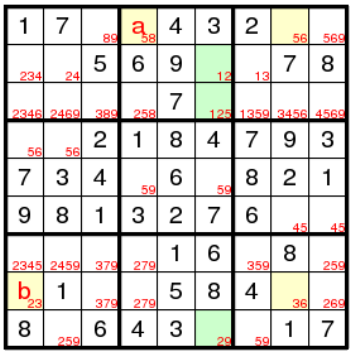

Figure 1: Example sudoku [1]

The Sudoku literature knows an extensive family of solving strategies with interesting names, such as Broken Wing and Finned Fish, most of which are implicitly implemented by the SAT solver. However, in situations where reasoning is not sufficient to derive another number assignment, the Sudoku solver (whether human or computer) is forced to make a – preferably educated – guess.

The *Nishio* strategy [1] chooses a value for a cell in the puzzle, and then considers what happens as a consequence. If this leads to contradictions, the chosen value is eliminated as a possibility. Moreover, literature suggests that the optimal literal to select is one within a cell that has the fewest remaining options.

In the SAT context, "choosing a value for a cell" simply implies assigning a positive literal. An equivalent for "the cell with the fewest remaining options" is less trivial to specify. We consider the example in figure 1. For the cell in location **a**, all literals for numbers *other than 5 or 8* (so 141 to 144, 146, 147 and 149 for this location on the board) have been removed from their clauses, as they must be false. In addition, the clauses which contain the negated variants of these numbers have been satisfied, and therefore removed. The number of negated literals removed increases as the number of constraints on a cell's value increases. Therefore, literals from within a cell that has the fewest options (here only 145 and 148 remain in relation to cell **a**) have the smallest number of negated literals still present in the CNF formula. Thus, translating *Nishio* to the SAT framework, the *Nishio* heuristic assigns the literal that has the lowest number of negated occurrences, and always sets it to True.

4 Hypothesis and rationale for the *Nishio* heuristic

To reiterate, this paper investigates how the structure of SAT problems matters for the performance of a solving strategy. This is analyzed by comparing the performance of DPLL with branching heuristics that are commonly claimed to outperform the random strategy, and DPLL with a problem-specific heuristic, that is

inspired by a real-life Sudoku strategy. If the hypothesis proves true, the present research shows that SAT problems cannot be understood as a general problem class and that problem-specific strategies are valuable to find solutions most efficiently.

In particular, this paper tests whether the *Nishio* strategy is better tailored to solve Sudokus than other common heuristic strategies. Compared to general DPLL heuristics such as Jeroslow-Wang and MOMs, the *Nishio* approach is similar in terms of complexity (and therefore comparable). However, its approach is somewhat odd. Rather than selecting the literal that occurs the *most* often (perhaps weighted by factors such as clause length), *Nishio* selects the negative literal that occurs the *fewest* number of times, and always sets it to positive.

Besides being analogous with a real-life strategy, the Nishio approach is sensible from the SAT perspective. Firstly, the preference for assigning positive literals is justified by the cascading effect of assigning a positive versus a negative literal. In a blank Sudoku puzzle (i.e. the rules are specified but no values for cells are assigned), when one *positive* literal is assigned, 765 clauses are instantly removed by the DPLL simplification steps. In contrast, if one *negative* literal is assigned, only 33 clauses are removed. This effect is of a similar size in subsequent rounds, and explains why there is a large interest in assigning positive rather than negative literals. This cascading occurs because all negated literals are within clauses of length 2. Thus, as a positive literal is assigned, its negated counterpart cannot be satisfied and is removed from the length-2 clause. This instantly creates many unit clauses, which can be removed. Conversely, assigning a negative literal leads to the removal of the (length-2) clauses in which it appears, but we are still undecided about the truth value of the other literal in the clause – so the cascading effect does not occur. This effect can also be viewed from the perspective of the real Sudoku: if a number in a cell *must* be a 9, it cannot have any value between 1 and 8. Conversely, if a number *cannot* be an 9, all values between 1 and 8 remain an option.

If assigning a positive literal seems a valid strategy, it must still be decided which literal is assigned. In general, either a literal is selected that can lead to removal of the largest number of clauses (i.e. the search tree is pruned as quickly as possible), or a literal is selected that is true with the largest probability (so that backtracking is limited). The *Nishio* approach favours the latter. Again, this seems reasonable: for cell **a** in figure 1, if either of the two values is assigned, it is likely that conflicts are reached quickly – much more quickly than when the value of a cell is assigned about which less is known (i.e. a high number of negated occurrences). In addition, if backtracking is necessary, for cell **a** the other literal is assigned to positive, which must now be correct. In contrast, for a cell with more than 2 options, still multiple options remain.

5 Experimental design

The DPLL algorithm was tested on a set of 3.000 Sudokus. The procedure was repeated once for every heuristic (i.e. 5 tests in total). For 1.457 Sudokus, only one simplification step and no guessing of literals was required; these puzzles therefore provide no information about the relative performance of the heuristics, and are discarded from the results. Several metrics are kept, including the number of literals the algorithm assigns for each puzzle (i.e. the number of 'guesses'), and the number of times the algorithm has to backtrack. In addition, a *difficulty score* is recorded for each heuristic, which is a simple summation of the number of literals assigned and the number of backtracks. The difficulty score aims to express how challenging a Sudoku is to a DPLL heuristic (of which the literal assignment and the backtrack count both are important indicators).

6 Experimental results and analysis

6.1 Visualization of results

Figure 2 shows the distribution of the number of literal assignments and the number of backtracks for each heuristic. As distributions are very skewed to the right, numbers are logged to be visualized appropriately (and to satisfy the normality-assumption for the ANOVA test that follows). As can be observed, only the *Nishio* heuristic outperforms the random branching strategy, as fewer literal assignments and backtracks are needed, while the other heuristics in fact perform worse than the baseline. Moreover, it seems that there is hardly any difference between the Jeroslow-Wang and MOMS heuristic. This makes sense: in the

Sudoku, the shortest clauses are always of length 2, and contain only negative literals (MOMS thus selects from among these literals). Jeroslow-Wang One-Sided is therefore by definition identical to MOMS, and Jeroslow-Wang Two-Sided in practice: the very few times that a literal occurs in a positive phrase are only given very small weight (especially because these clauses are longer).

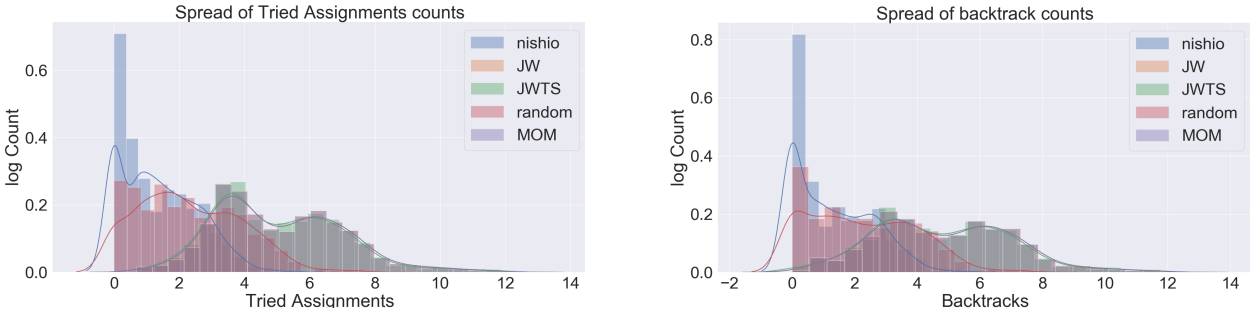

Figure 2: Distribution of the assignment and backtrack count for each heuristic over 1543 Sudokus

Figure 3 shows the relative difficulty score of a puzzle for all pairs of heuristics. (The random strategy is omitted because its difficulty score is also random, and therefore not meaningful at the level of individual puzzles.) If all puzzles are relatively equally difficult to each heuristic, these values would lie on a straight line. However, this is clearly not the case: some puzzles that score high on one axis (i.e. are among the more difficult puzzles for that heuristic) score low on the other axis (i.e. are among the easier puzzles for the other heuristic), and vice versa. This shows that a heuristic may work relatively well on some problems, but worse on others, again indicating that a heuristic cannot be said to be 'generally better' than another, but more factors (such as problem structure) determine the performance of a strategy.

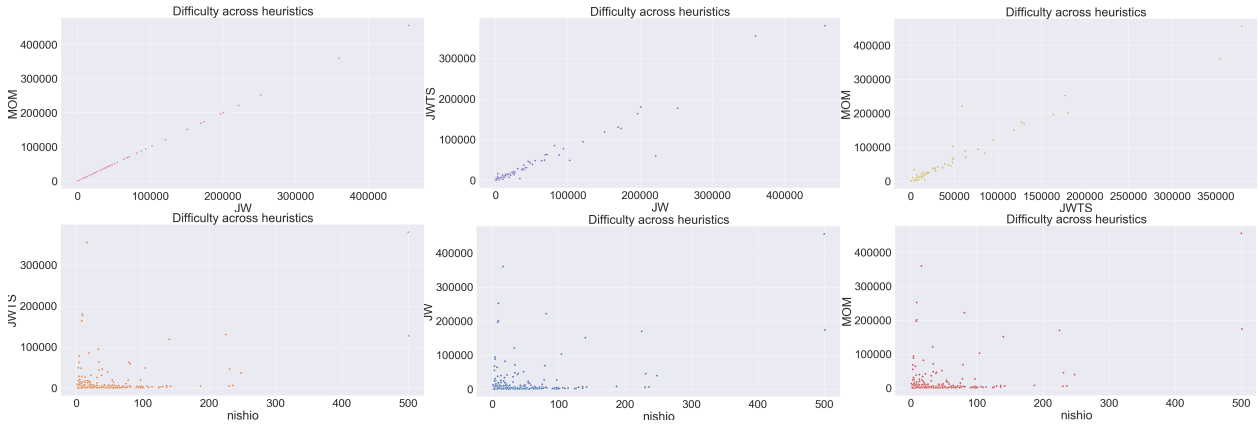

Figure 3: Difficulty scores of 1543 Sudokus over pairs of heuristics

6.2 Significance tests

An ANOVA test [6] is performed to analyze whether the mean difficulty score is significantly different between heuristics. Difficulty scores are logged to satisfy the assumption that these are normally distributed. Indeed, the ANOVA indicates that the heuristics are significantly different from each other (at the $p \leq 0.05$ significance level; F-score ≈ 1839.08).

Next, Tukey's HSD test is applied to assess the nature of this significant difference: which heuristics are different from each other, and in what direction? Results are reported in table 1. All pairs of heuristics differ significantly from each other (except Jeroslow-Wang One- and Two-sided, as well as Jeroslow-Wang and MOMS, which have a near-identical performance). All general heuristics perform worse than the random strategy on the Sudoku problems, while the *Nishio* heuristic performs better.

group1	group2	meandiff	lower	upper	reject
JW	JWTS	-0.127	-0.309	0.055	False
JW	MOMS	0.0	-0.182	0.182	False
JW	**nishio**	-3.884	-4.066	-3.702	True
JW	**random**	-2.8933	-3.0753	-2.7113	True
JWTS	MOMS	0.127	-0.055	0.309	False
JWTS	**nishio**	-3.757	-3.939	-3.5749	True
JWTS	**random**	-2.7663	-2.9483	-2.5843	True
MOMS	**nishio**	-3.884	-4.066	-3.702	True
MOMS	**random**	-2.8933	-3.0753	-2.7113	True
nishio	random	0.9907	0.8087	1.1727	True

Table 1: Results of Tukey's HSD test. Significantly better heuristic emphasised in bold.

7 Conclusion

The present research has shown that heuristics for the DPLL that are commonly reported to outperform a random branching strategy in fact perform worse than this benchmark for Sudoku puzzles. In contrast, the *Nishio* heuristic, which is inspired by human solution strategies, is a significant improvement to the random strategy. In the present case, the increased performance by the *Nishio* heuristic seems to be due to the fact that the problem consists of a few longer clauses which contain only positive literals, and many length-2 clauses which contain only negative literals. Assigning positive literals has a large cascading effect in this setup by the reduction of many clauses to unit clauses. As such, the research shows that the structure of SAT problems matters in finding the optimal solution strategy. This also indicates that it is problematic to describe heuristics as generally 'good' at solving SAT problems.

In future research, an increased understanding of the structure of SAT problems and a better vocabulary to describe them (i.e. a SAT problem typology) would aid in the development of problem-specific heuristics. The present research provides some pointers to the relevant indicators of problem structure, such as clause length and relative positive/negative occurrence. Lastly, it seems valuable to take inspiration from real-world analogous solving strategies for particular problem types, as these might help to derive heuristics that are less obvious from the SAT context alone.

References

[1] Andries Brouwer. Solving sudokus, 2019. URL `https://homepages.cwi.nl/~aeb/games/sudoku/index.html`.

[2] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[3] Jun Gu, Paul W Purdom, John Franco, and Benjamin W Wah. Algorithms for the satisfiability (sat) problem: A survey. *DIMACS Series in Discrete Mathematics, American Mathematical Society*, 1999.

[4] Robert G Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of mathematics and Artificial Intelligence*, 1(1-4):167–187, 1990.

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

[6] Scott E Maxwell and Harold D Delaney. *Designing experiments and analyzing data: A model comparison perspective*. Routledge, 2003.

[7] Frank Van Harmelen, Vladimir Lifschitz, and Bruce Porter. *Handbook of knowledge representation*, volume 1. Elsevier, 2008.