Comparison of Different Heuristics Using the DPLL Algorithm to Implement a SAT Solver for Sudoku Puzzles

Vrije Universiteit Amsterdam

Abstract. The implementation of a satisfiability (SAT) solver based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm is presented. Next to the naive DPLL approach, the the maximum occurrences in clauses of minimum size (MOMs) and one-sided Jeroslow-Wang (JW) are utilised with the intention of improving the performance of the SAT solver. Two different hypotheses are then evaluated across different Sudoku difficulty levels. The first one studies how the k-value in MOMs heuristic impacts the number of splits, while the second one compares the performance of MOMs and JW.

Keywords: Sudoku · SAT solver · DPLL algorithm · MOMs heuristic · Jeroslaw-Wang heuristic · Proposition logic · Satisfiability

1 Introduction

In the present work, the implementation of a satisfiability (SAT) solver along with two heuristics and hypotheses will be looked at with the aim of studying the behaviour of the SAT solver. The SAT solver can be used for any SAT problem that involves propositional logic. So as to look at the behaviour of the SAT solver, Sudoku puzzles that vary in level of difficulty will be utilised. A Sudoku is made up from a grid that consists of nine by nine cells. There are nine blocks in total with each block consisting of nine cells. With some cells already containing a digit, the aim is to complete the Sudoku by filling in the rest of the grid, such that each block, row and column contain each digit only once. These rules have been encoded in clause normal form (CNF) and are, next to the Sudoku, given as input to the SAT solver. Hereafter, the SAT solver tells whether the given Sudoku has a solution or not. The SAT solver satisfies the property of soundness as it presents a solution in case the Sudoku is satisfiable, however it does not satisfy the property of completeness, since only one solution is presented in case of multiple solutions.

2 SAT Solver Design

To build the SAT solver, the Davis-Putnam-Logemann-Loveland (DPLL) algorithm (see Algorithm 1 below) has been used. The algorithm takes as input a

problem encoded as a CNF formula and outputs whether the given problem is satisfiable or not. The given problem is satisfiable if there exists an assignment for all literals in the formula, such that the formula is satisfied. If no such assignment exists, then the problem is unsatisfiable. An assignment satisfying the formula is outputted in case of a satisfiable formula. The algorithm starts with a CNF formula F and an assignment ρ, which is initially empty. While F is not empty, the algorithm proceeds recursively by choosing a literal l from F that is not in ρ. The clauses in which ls occurs are removed from F and l is added to ρ. $\neg l$ is removed from all clauses that contain it. If this leads to an empty clause in F, then the previous step is repeated with $\neg l$ instead. In each iteration, unit clauses, which are clauses consisting of only one literal, are sought in F with the aim of satisfying those first. Hereafter, pure literals, which are literals whose negation is not contained in F, are sought. If F does not contain any clauses anymore, then F is satisfiable and the assignment ρ is outputted. However, if F contains the empty clause, then F is unsatisfiable.

Algorithm 1 Davis-Putnam-Logemann-Loveland Algorithm

```
1: Input: CNF formula F and assignment ρ = ∅
2: procedure DPLL(F, ρ)
3:     (F, ρ) ← unitPropagation(F, ρ)
4:     (F, ρ) ← pureLiteralElimination(F, ρ)
5:     if F contains no clauses then
6:         Output ρ
7:         return SATISFIABLE
8:     if Empty clause in F then return UNSATISFIABLE
9:     l ← literal not in ρ
10:    if DPLL(F|l, ρ ∪ {l}) then return SATISFIABLE
       return DPLL(F|¬l, ρ ∪ {¬l})
11: procedure unitPropagation(F, ρ)
12:    while F has a unit clause x do
13:        F ← F|x
14:        ρ ← ρ ∪ {x}
15:    return (F, ρ)
16: procedure pureLiteralElimination(F, ρ)
17:    while F has a pure literal x do
18:        F ← F|x
19:        ρ ← ρ ∪ {x}
20:    return (F, ρ)
```

The implementation of the SAT solver is based on Algorithm 1. A function called `DP` executes the main procedure. This function takes as input a CNF formula, which is a list containing lists with each list storing the literals in one clause of the formula, and an assignment, which is an initially empty list. While there is a unit clause found in the formula, the function `unitPropagation` is

called to search for a unit clause. Likewise, while there is a pure literal found in the formula, `pureLiteralRemoval` is called to search for a pure literal. In case of a unit clause or pure literal, `unitPropagation` and `pureLiteralRemoval`, respectively, call a function named `assignValue` by passing the CNF clauses, assignment and found literal as parameters. `assignValue` iterates through the clauses and deletes the clauses in which the literal is present. If the negation of the literal is present in a clause, it is deleted from the clause. `DP` proceeds by checking if the list of clauses is empty. If this is the case, then the solution is printed and the function return `True`. If the list contains an empty list, which represent an empty clause, then `DP` returns `False`. In case the function does not return, it continues by calling `chooseLiteral` to choose a literal for the assignment according to the strategy. `chooseLiteral` returns the first literal in the first clause from the formula if the naive DPLL strategy is used. If a heuristic is used, `chooseLiteral` calls a function according to the heuristic to choose the literal that is best to split on. `DP` calls itself after splitting on the chosen literal. Before splitting, `assignValue` is called by passing on the formula, assignment and chosen literal. If the iterative call does not return `True`, then the formula, assignment and negation of the chosen literal are passed on to `assignValue`. Another iterative call to `DP` is made.

3 Approach

In addition to the basic DPLL algorithm, the maximum occurrences in clauses of minimum size (MOMs) and one-sided Jeroslow-Wang (JW) heuristic have been implemented. Both belong to the early branching heuristics and can be regarded as greedy algorithms. This means that the algorithms are making the locally optimal choice in each phase. A disadvantage of this is that they usually do not find optimal solutions [1, 9]. On the other hand, branching on these heuristics is more likely to lead to unit propagations, formula simplification and with a little more time a globally optimal solution will be found [7]. Even though both heuristics belong to the early branching heuristics, they both have a different approach to a problem. This is the reason these two have been chosen. In the following subsections the heuristics will be discussed in more depth.

3.1 MOMs

This heuristic is the most used SAT heuristic, because of its accuracy, ease of implementation, and problem-independence. According to other researchers, this heuristic seems to work best of the known heuristics [2, 6, 8]. As the name implies, it chooses to split on the literal that occurs most among clauses of minimal length and does not include all clauses that are greater than the minimal length. Intuitively, it gives preference to literals that occur in a large number of clauses as x or in $\neg x$ [2].

Let $f^*(x)$ be the number of unresolved smallest clauses that contain x. x will be chosen to maximise the following:

$$((f^*(x) + f^*(\neg x)) \cdot 2^k + f^*(x) \cdot f^*(\neg x)$$

The value for k has to been chosen heuristically [4] and in this case, it is set to 1.8, since this resulted in the fastest run time of finding the solution of the example Sudoku.

In this paper, the implementation of this heuristic has the following approach:

- A dictionary is used as a counter with the keys being the literals and the values being the scores.
- Only literals in the clauses of minimal length(`minClause`) will be picked and when this happens the counter increases by 1.
- A score is assigned to each key stored in this counter by calculating the value $((f^*(x) + f^*(\neg x)) \cdot 2^k + f^*(x) \cdot f^*(\neg x)$.
- A built-in condition is used to make sure that the key with a higher score will be stored, which means that the key with the current highest score is overwritten by a key with a higher score.
- The literal with the highest score will be the one that is split on

3.2 JW

Likewise to the MOMs heuristic, the JW heuristic is based on the idea that literals appearing in clauses of shorter length are preferred. However, compared to MOMs heuristic, the JW heuristic requires less complex computations. The algorithm calculates a score for each literal in the formula and chooses the literal with the maximal score to split on. For a literal l, the score $f(l)$ is defined as $f(l) = \sum_{i:l,\neg l \in C_i} 2^{-|C_i|}$, where $|C_i|$ denotes the number of literals in clause C_i and the sum is over all clauses C_i which contain the literal l or $\neg l$. If $f(l) > f(\bar{l})$, the heuristic branches on l, because the higher value has to be chosen. The longer clauses should contribute exponentially smaller weights to the literals they contain [3, 7].

The implementation of this heuristic has the following approach:

1. A dictionary is used as a counter for the key-value pairs.
2. Each clause is then looped and in the loop each literal is looped.
3. For each literal, the score of the matching (positive) variable is increased with $2^{-|C_i|}$.
4. The maximum value will be chosen with the `max()` function.

4 Hypotheses

The following two hypotheses have been selected to be tested experimentally.

1. The k-value has a different effect on the performance of MOMS heuristic when applied to various difficulty levels of the Sudoku.

2. The strategy using the MOMs heuristic continues to outperform the one using JW with respect to the run time and number of splits, irrespective of the difficulty of the Sudoku.

The first hypothesis will give some inside about the function of this k-value with regards to solving Sudokus of various difficulty levels. The expectation is that the best k-value will differ depending on the difficulty level of the Sudoku. The level of difficulty depends on the amount of numbers that are already given, where a Sudoku with many numbers given is considered as more easy than a Sudoku with fewer numbers given. A lower difficulty level requires less computation and, thus, might also require a lower amount of splits than Sudokus with a higher difficulty level. Understanding the effect of adjusting this k-value can be seen as beneficial for optimising the SAT solver.

The second hypothesis can also be seen as a very interesting case, since the performance of both heuristics is compared. As the hypothesis states, the expectation is that the SAT solver with MOMs heuristic will have a better performance than the one with JW heuristic with respect to the run time and number of splits, irrespective of the difficulty level of the Sudoku.

5 Experimental Design

In this section, the experimental designs of the two different hypotheses will be described. Each experimental design is divided into the experimental conditions, metrics and the statistics.

5.1 K-Value Optimisation

Experimental Conditions The experiment will consist of three different types of test sets:

1. Easy Sudokus: 25 numbers are given
2. Medium Sudokus: 21 numbers are given
3. Hard Sudokus: 17 numbers are given

All sets consist of forty Sudoku puzzles. Before using the Sudoku puzzles, they were transformed to a DIMACS format through a script.

To check whether the optimal k-value differs significantly for different difficulty levels, the number of splits of each Sudoku have been counted. When a Sudoku reported 0 splits, it means the algorithm was able to solve the Sudoku with pure literal elimination and unit propagation. When this happens the MOMs, heuristic is not even used. To measure the best performance of this heuristic, these Sudokus have been taken from the sets, such that there are 20 easy, 25 medium and 38 difficult Sudokus left. We ran the leftovers on 12 different k-values ranging from 0 to 4 with intervals of 0.5. Python 3.8 was used to run this experiment and data were analysed using IBM SPSS Statistics 26.

Metrics The metric that is utilised is the number of splits that are encountered while solving the Sudoku. This metric is chosen as it is a very suitable measure for testing the best value of k for the MOMs heuristic.

Statistics The central limit theorem cannot be applied to this case, because by removing a number of Sudoku the data does not exceed thirty [5]. So, before the statistical test will be chosen, it must first be tested whether the data is normally distributed. The Shapiro-Wilk test will be used for this. Based on the outcome of this test, a parametric test (if the data is normally distributed) or a non-parametric test (if the data is not normally distributed) will be performed.

5.2 MOMs vs. JW

Experimental Conditions The dataset used was the same one as the previous hypothesis, with three different groups of Sudokus (easy, medium, hard, with, respectively, 25, 21, and 17 givens) used for testing. Each heuristic received as input the three different datasets and its performance was later on evaluated.

Metrics To evaluate the performances of the two algorithms, two metrics were selected. The first one is run time (in seconds) and the second one is the number of splits performed by the algorithm. Although run time might not be a very informative measure, all the trials were performed on the same device under the same conditions, to ensure the possibility to compare relative run times. The number of splits was once again chosen to evaluate the algorithms' capacity of performing good choices (resulting in a lower amount of total splits).

Statistics The central limit theorem was applied, since in all three datasets the number of variables exceeded thirty [5]. In addition, Levene's test and F test for heteroskedasticity were run to rule out the presence of possible biases in the dataset. An ANOVA test was performed on the collected data, selecting first run time as the dependent variable and then number of splits. The predictors were always two (Type of Heuristic and Difficulty). The experiment was performed using Python 3.8 and the data were analysed using IBM SPSS Statistics 26.

6 Experimental Results

6.1 K-Value Optimisation

Since the Shapiro-Wilk test has shown that the data of all sets is not normally distributed ($P < .05$), a non-parametric test will have to be performed to check for significance.

As can be seen in figure 1, the number of splits is increasing when the k-value is also increasing. The k-value differs for all three sets, but compared to the easy and difficult Sudokus, the medium Sudokus have reported many more splits.

After performing a Bonferroni-adjusted pairwise comparison, it resulted that the easy and hard datasets did not differ in mean number of splits when accounting for k-value ($P = .774$). Both the easy and hard datasets reported a significantly lower number of splits compared the medium dataset, when accounting for k-value ($P < .001$).

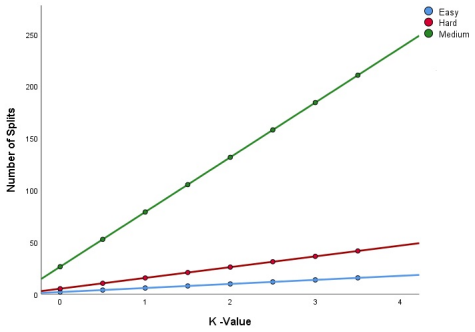

Fig. 1. Number of splits regarding easy, medium and hard Sudokus

6.2 MOMs vs. JW

None of three datasets reported 0 splits for the Sudokus in it, so no values were excluded. The analysis of run time revealed the presence of little differences between the two heuristics. There were non-significant differences in run time for both the easy and the hard Sudokus (overlapping confidence intervals), while there was a significant difference in the medium Sudokus dataset (overlapping confidence intervals).

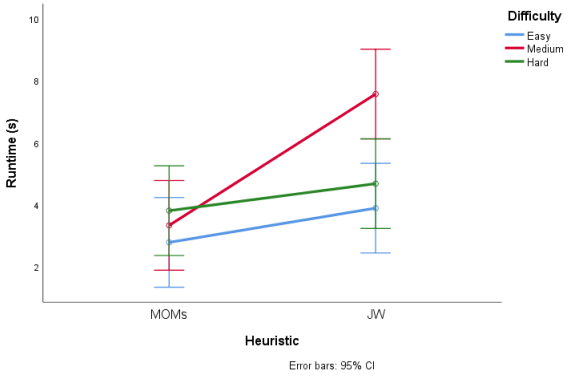

Fig. 2. Runtime of heuristics for Sudokus of different difficulty

The same fact proved to be true also for the number of splits, in which the only significant difference between the two heuristics was found in the dataset of medium difficulty. The other two sets showed, on average, a comparable number of splits.

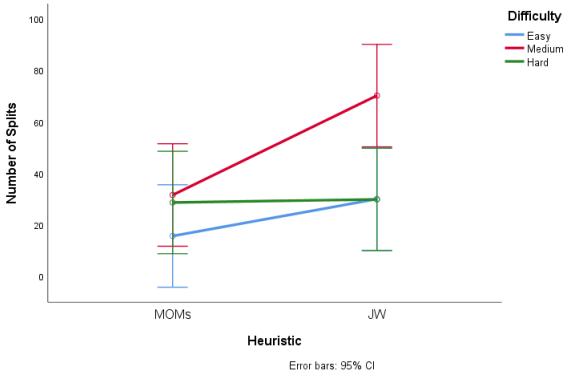

Fig. 3. Number of splits of heuristics for Sudokus of different difficulty

7 Conclusion

When comparing the different k-values between easy, medium and hard Sudokus, with regards to the number of splits, a significant difference was found between the medium Sudokus and the easy and hard Sudokus. This means that the optimal k-value depends on the various difficulty levels of the Sudoku. If the optimal k-value is found for each level of difficulty, the MOMs heuristic can be improved.

There is a significant difference found when comparing the number of splits and the run time of both heuristics for Sudokus with a medium difficulty level. Therefore, there can be said that the SAT solver with MOMs heuristic has a better performance than the one with JW heuristic in solving the Sudokus where 21 numbers are given. For the other two difficulty levels the difference between the two heuristics was not this clearly visible.

For future research, there can be investigated what might be the reason behind the deviating number of splits for Sudokus of medium difficulty level in comparison to the other two difficulty levels. A potential reason behind this is that a Sudoku with a high difficulty level could have a higher amount of solutions than a Sudoku with a medium or low difficulty level. This makes the high level Sudokus easier to satisfy. Therefore, the number of splits is significantly lower in comparison to the other Sudokus. This hypothesis is something that can be investigated in future work.

References

1. Ejim, S.: Implementation of greedy algorithm in travel salesman problem (09 2016). https://doi.org/10.13140/RG.2.2.23921.48485
2. Freeman, J.: Improvements to propositional satisfiability search algorithms. Ph.D. thesis, Citeseer (1995)
3. Jeroslow, R., Wang, J.: Solving propositional satisfiability problems. Annals of Mathematics and Artificial Intelligence **1**(1–4), 167–187 (Sep 1990). https://doi.org/10.1007/BF01531077, https://doi.org/10.1007/BF01531077
4. McMillan, K.L.: Sat-based bounded model checking (2015), https://slideplayer.com/slide/4805174/
5. Montgomery, D.: Applied Statistics and Probability for Engineers, 6th Edition. John Wiley and Sons, Incorporated (2013), https://books.google.nl/books?id=eHpbAgAAQBAJ
6. Oh, C.: Between sat and unsat: The fundamental difference in cdcl sat. In: Heule, M., Weaver, S. (eds.) Theory and Applications of Satisfiability Testing – SAT 2015. pp. 307–323. Springer International Publishing, Cham (2015)
7. Pisanov, V.: Novel value ordering heuristics using non-linear optimization in boolean satisfiability (2012), http://hdl.handle.net/10012/6941
8. Pretolani, D.: Efficiency and stability of hypergraph sat algorithms. DIMACS Series in Discrete Mathematics and Theoretical Computer Science **26**, 479–498 (1996)
9. Zhang, L., Malik, S.: The quest for efficient boolean satisfiability solvers. In: Proceedings of the 14th International Conference on Computer Aided Verification. p. 17–36. CAV '02, Springer-Verlag, Berlin, Heidelberg (2002)