Analysis of Probabilisic Heuristics for Solving Sodukus Using DPLL

Karan Malhotra[1,2], Arvid Lindström[1,2]
[1]*Faculty of Science, Vrije Universiteit Amsterdam*
[2]*Faculty of Science, Universiteit van Amsterdam*
{*2655849, 2556466*}*@vu.student.nl,* {*karan.malhotra, arvid.lindstrom*}*@student.uva.nl*

Abstract: The DPLL algorithm for solving SAT-problems has become the basis for many industry standard algorithms. In particular a large amount of heuristics have been developed to further increase its performance. In this paper we modify two of the most popular heuristics, namely Jeroslow-Wang and DLCS, with a probabilistic approach which introduces non-greedy moves. We show that a Jeroslow-Wang Epsilon-Greedy heuristic with a 5% probability of choosing literals from the top 10% ranked literals outperforms the baseline DPLL on the task of classical 9x9 sodukus expressed in conjunctive normal form. The probabilistic modification to DLCS assigns truth values to literals depending on the literals frequency count and clause length. This method fails to outperform the baseline-DPLL by a large marigin due to the nature of how sodukus are expressed in CNF.

1 Introduction

Since its inception in 1962 by G. Logemann and D.W. Loveland, the DPLL-algorithm has become the basis for modern SAT-solving (Davis et al., 1962). Also known as the DP-algorithm, in reference to the original work by (Davis and Putnam, 1960), the algorithm is a complete Boolean satisfiability solver. We define a *split* as one step in the DPLL algorithm in which a literal is chosen and assigned a truth value. Using a technique known as backtracking, DPLL revisits previous states of a search tree of literals from a set of propositional logic clauses in order to exhaustively explore every possible combination of literals (SS and Bhaya, 2004).

In this paper we propose two heuristics used to improve the performance of the DPLL algorithm on the task of solving sodukus expressed as SAT-problems in conjunctive normal form. These heuristics attempt to, respectively, select more informative literals using a variation of the Jeroslow-Wang heuristic (G. Jeroslow and Wang, 1990) and assign a truth value $\rho \in \{True, False\}$ using a probabilistic framework (see Sec 4.2). By informative literals we mean literals which traverses the algorithm through the search space quicker during the split-step. Using the amount of splits and backtracks performed by the algorithm as a metric of efficiency, we investigate how many operations DPLL requires to fully satisfy a sudoku puzzle using our heuristics. For our purposes we define *operation* as a split or backtrack action performed by the algorithm.

1.1 Research questions and hypotheses

In particular, we strive to answer the following questions:

1. In the context of $9x9$ sudoku puzzles, does an epsilon-greedy Jeroslow-Wang heuristic reduce the required amount of operations by introducing random moves into the standard Jeroslow-Wang heuristic?
2. Similarly, does our second heuristic, which can be seen as a stochastic modification of DLCS (Marques-Silva, 1999), reduce operations by introducing a probability over choosing truth-assignments?

Our evaluation dataset consists of 1011 sudokus expressed as clauses in propositional logic. For a given sudoku, most clauses are binary clauses consisting of negated literals. A few longer clauses consist of non-negated literals. Based on this observation we state the following hypotheses:

- Because almost all clauses are binary, our modified Jeroslow-Wang heuristic should produce JW-scores of similar ranges for most literals and will therefore not perform differently from our baseline DPLL algorithm (which selects literals randomly in the split-step).
- For our second heuristic, the probability of assigning either truth value depends on the literal

count and clause-length. Compared to our baseline DPLL which always sets a chosen literal to first true and then false (backtrack), our second heuristic should perform better since it will initially have a larger chance of setting the negated literals to false, thus reducing the initial search space more efficiently.

2 Background

A number of heuristics have been proposed in the last decades with the intent of reducing the amount of operations required by DPLL to find a satisfiable solution to a SAT-problem. For our purposes we give a brief introduction to two types of heuristics used in our experiments. *Branching heuristics* are a type of heuristic which incorporates dynamic information from the backtracking algorithm to make informed choices as to which unassigned literal should be chosen next (Marques-Silva, 1999). In this paper, our first heuristic is an extension of the One-Sided Jeroslow-Wang heuristic which greedily chooses an unassigned literal based on (1) (G. Jeroslow and Wang, 1990).

$$J(l) = \sum_{l \in \omega \wedge \omega \in \rho} 2^{-|\omega|} \tag{1}$$

In JW, for each literal l which appear in our unsatisfied clauses $\omega \in \rho$, where ρ is the set of all remaining clauses, the score $J(l)$ is calculated as the sum of 2 raised to the negative length of l's clauses. In the One-sided case, the literal l is chosen which has the largest $J(l)$ score. The truth value assigned to l corresponds to the sign at which l received the largest $J(l)$. This has the effect of prioritizing literals which appear in clauses of smaller length, which has a larger chance of constraining other variables, thus reducing the search space.

Another family of heuristics is the *Literal Count Heuristics*, of which DLCS is a member (Sang et al., 2005). DLCS decides upon a literal based on the number of occurrences of that literal (of both signs) in our set of remaining clauses according to (2),

$$\arg\max_{V} = c(V_p) + c(V_n) \tag{2}$$

where $c(.)$ is the count function. V is a literal where V_p and V_n denotes its positive and negated occurrences. Literal V is set to true if $c(V_p) > c(V_n)$ and false otherwise.

2.1 Introducing random behavior

On both the aforementioned heuristics, we extend their functionality by adding stochastic behavior, further explained in Section 4. The idea of using random behavior in SAT-solvers is well explored in the literature (Marques-Silva, 1999) and can benefit solvers operating in a space where solutions can be found without exhaustively exploring every possible variable assignment. Hence, random actions reduces greediness. Because the problem domain of this research concerns sudoku solving, we believe that by allowing random actions, the heuristic algorithms will avoid searching in difficult areas for too long, thus reducing the amount of operations.

3 Baseline Algorithmic Design

As the DPLL algorithm has been thoroughly explored in the literature, we shall refrain from giving a detailed description of the inner workings of the algorithm. For a detailed description we refer to the original work by (Davis and Putnam, 1960). The most important design choices for our implementation reside in the data-structures chosen to represent the state of the algorithm. These are outlined as follows:

1. Literal to Clause Map : A dictionary mapping of literals (and their negated form, if any) to the ID of clauses that they occur in.
2. Clause to Literal Map : The mirror dictionary to the aforementioned dictionary. Maps the literals contained in each clause.
3. Current count of literals : This data structure maintains the current count of the occurrences of literals and their negated form. The counts maintained are clause independent i.e they depict the total occurrence of literals across all clauses .
4. Assignments : It contains the current status of each literal in the data set. Status codes :- '0' for False , '1' for True and '-1' for unassigned.

These design choices help us carry out operations efficiently. When setting a literal to true or false, all the data structures reflect the changes accordingly. For instance, on setting a literal to 'True', the clauses containing the literal and it's negated form are determined from the 1st data structure through a dictionary lookup in constant time. Clauses in which a literal has been set to true will be removed, which takes place in the 2nd data structure. A SAT-problem has been satisfied when the length of dictionary 2) has reached 0. Subsequently, as clauses are removed, the counts maintained in data structure 3 will be decremented. This allows us to easily check for pure-literals, a commonly used simplification technique, by defining literals $l \in \rho$ in which $c(l_p) = 0$ or $c(l_n) = 0$ as pure-literals. Once again $c(.)$ is the count function and ρ is the set of remaining clauses.

4 Proposed Heuristics

4.1 Epsilon-Greedy Jeroslow-Wang

Our first heuristic builds upon the Jeroslow-Wang method. After calculating the $J(l)$ score for each literal, we choose with a probability $\varepsilon \in [0,1]$ to pick a random variable in the top k percent of the literals with highest $J(l)$ scores. With a probability of $1-\varepsilon$ we perform the standard One-sided Jeroslow-Wang and select the literal with highest $J(l)$. This has the effect of introducing search areas which may have been explored much later.

4.2 Probabilistic DLCS

Our second heuristic extends the DLCS method by assigning literal $V = true$ based on (3). A literal V is still selected based on (2).

$$\begin{aligned} M_p &= \frac{\sqrt{c(V_p)}}{\sum_{V_p \in \omega} 2^{|\omega|}} \\ M_n &= \frac{\sqrt{c(V_n)}}{\sum_{V_n \in \omega} 2^{|\omega|}} \\ p(V = true) &= \frac{M_p}{M_p + M_n} \\ p(V = false) &= 1 - p(V = true) \end{aligned} \tag{3}$$

We refer to M_p and M_n as the frequency weights of a literal V. The nominator uses the square root of the count of $V_{p/n}$ to introduce a non-linearity which ensures that different frequency counts always lead to different values of $M_{p/n}$. For example, without the $\sqrt{\cdot}$, a ratio like $1/2$ and $10/20$ would yield the same $M_{p/n}$. The denominator $\sum_{V_n \in \omega} 2^{|\omega|}$ discounts the influence of frequency count by the sum of the length of all clauses in which $V_{p/n}$ occurs. This means that the frequency weight $M_{p/n}$ will be higher for literals which occurs frequently, while placing an increased priority on literals which occur in short clauses. Finally we select the truth assignment based on a normalized probability distribution defined as the ratio of the positive frequency weight to the total weights for positive and negative occurrences.

5 Experimental Setup

We use *Number of Operations* as a metric to compare the performance of the algorithms. We define number of operations as the total number of *True / False* decisions made during the split steps as well as backtracks. For instance, if a literal is initially set to *True* and then *False* (backtrack), the operation count for this literal is 2. A lower count corresponds to a better algorithm. Intuitively this implies that split choices made by the algorithm were wise. A wise split choice is one which directs the algorithm in the solution region of the search space. A lower operation count essentially means less exploration. Wise decisions direct the algorithm in favour of simplification, further leading to fewer operations required. Note that we do not consider the simplification steps of the algorithms as part of our metric as the heuristics defined do not explicitly modify the simplification process.

In order to test our hypotheses, we run each of the three algorithms on a test set of 1011 sudokus and measure the operation counts on each sudoku. As there is an element of stochasticity involved, we run the algorithms on three different seeds to reduce noise and for fair comparison. We perform a statistical significance test to verify that the superior performance of an algorithm is not due to random chance. We choose Wilcoxon two-sided signed-rank test (Wilcoxon, 1945) with $\alpha = 0.05$ to test our hypothesis that our Jeroslow-Wang-Epsilon-Greedy heuristic and the baseline DPLL require the same amount of operations. To test our second hypothesis we perform a one-sided Wilcoxon signed-rank test to see if the amount of operations required for the baseline DPLL is greater than our probabilistic DLCS heuristic. Finally, we evaluate the remaining number of unsatisfied clauses over the amount of operations for a given soduku to inspect the search behavior of our baseline DPLL and two heuristics.

In all of the following results, hyper-parameters ε and k for heuristic 1 (see section 4.1) were set to 0.05 and 0.1 respectively through experimentation. These values were shown to strike a good balance between incorporating the standard Jeroslow-Wang heuristic and non-greedy behavior.

6 Results

	Baseline	Heuristic 1	Heuristic 2
Min	11	9	8
Median	41	26	119
Max	488	317	12587

Table 1: Aggregated operation counts for the three algorithms

	Operation count cases		
DPLL vs. Heuristic 1	$B = H_1$	$B > H_1$	$B < H_1$
	7	783	221
DPLL vs. Heuristic 2	$B = H_2$	$B > H_2$	$B < H_2$
	2	242	767

Table 2: Comparison between Baseline DPLL and proposed heuristics operation-counts across 1011 sudokus

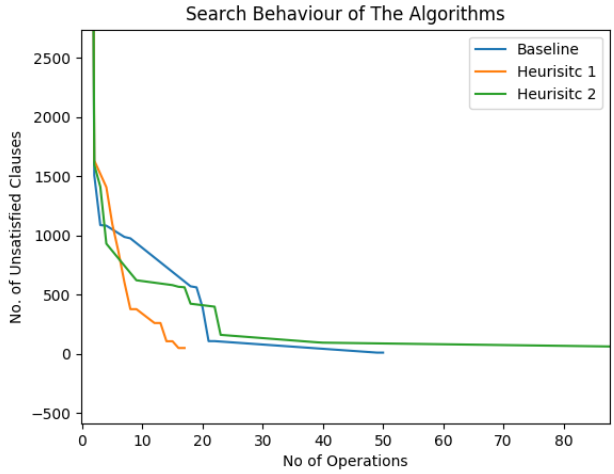

Figure 1: This figure shows the behaviour of the three algorithms on soduku number 923.

6.1 Statistical Analysis

From the two-sided Wilcoxon signed-rank test, we report a p-value of 4.56×10^{-90}. This indicates a very low probability that the difference in operation-counts from our baseline algorithm and Jeroslow-Wang-Espsilon-Greedy heuristic center around 0. As of such, we reject our initial hypothesis that their performance should be equal.

For our one-sided Wilcoxon signed-rank test, the reported p-value is 2.32×10^{-111}, showing a very low probability that the operation count difference: $count(H_2) - count(\text{Baseline})$ is negatively skewed. We therefore also reject our second hypothesis stated in sec 1.1.

6.2 Interpretation of results

6.2.1 Jeroslow-Wang Epsilon Greedy

From Table 1 and 2, it can be inferred that the performance of heuristic 1 differs from that of the baseline algorithm. In column 2 of table 2, it can be seen that the cases in which the baseline required more operations than heuristic 1 ($B > H_1$) amounts to 783 out of 1011 sudokus. In addition, the maximum amount of operations required for heuristic 1 was 317, compared to 488 for our baseline DPLL. We note that these could have originated from different sudokus of different difficulty, possibly favouring one method over the other. As such our emphasis lies on the statistical evidence from section 6.1. Based on the Wilcoxon signed-rank test, the answer to our first research question is that Jeroslow-Wang Epsilon-Greedy does indeed reduce the amount of operations required to solve 9x9 sodukus.

To further draw insight into the search behavior of our heuristics and baseline-DPLL, we refer to Figure 1. It can be seen that for a given soduku, both the baseline and heuristic 1 traverse the search space equally within the the first few amount of operations. This is caused by the nature of our problem-statement in which almost every clause is in binary form, consisting of negated literals. The initial dip (until around 1500 unsatisfied clauses) is therefore dominated by a reduction of binary clauses. Consequently, the real advantage of heuristic 1 is only exercised after the search space has already been reduced. We define a 'plateau' region as an area in the search space where an algorithm performs many split steps and backtracks in pursuit of reducing the number of unsatisfied clauses with very limited progress. For the particular soduku in Fig. 1, heuristic 1 has chosen literals which still appear primarily in small clauses, thus reducing wide plateau regions.

6.2.2 Probabilistic DLCS

Looking at Table 1, it can be seen that the median required number of splits for sodukus solved by heuristic 2 is confidently larger in comparison to the baseline-DPLL. The maximum required amount of operations for heuristic 2 is drastically larger than that of the baseline algorithm. The decision to reject our second hypothesis is primarily supported by the Wilcoxon test shown in section 6.1. As such we must re-asses our position on our second heuristic as it clearly performs much worse than the baseline algorithm.

A keen insight on what has caused this severe decrease in performance can be gained from Figure 1. Around 20 operations and onward, a large plateau region can be observed for both the baseline algorithm - choosing random literals, and heuristic 2 - choosing literals based on count frequency (see equation 2). In this stage of the search space, most literals will occur in negated form. This creates a peculiar problem for our second heuristic, strictly related to sodukus expressed in this format. Due to the calculation of the probability distribution over truth-assignments presented in (3), heuristic 2 will have a very small

probability of ever assigning the value *true* to a literal, causing it to rely on unit-clause elimination to remove longer clauses. This becomes problematic with longer clauses consisting of positive literals as setting a literal to false will have a negligible effect for these clauses. In the event that a unit-clause elimination leads to a contradiction, heuristic 2 will perform a needlessly large amount of backtracking which could have been avoided by assigning literals to *true* earlier in the search. This is a problem our baseline algorithm does not face as it always assigns the value *true* until a contradiction is found.

7 Conclusions

In this paper we have explored the potential of extending two commonly used heuristics used in the DPLL algorithm (Davis et al., 1962) with stochastic behavior to reduce the required amount of operations (splits and backtracks) when solving classical 9x9 sodukus. Namely, we have extended the Jeroslow-Wang (G. Jeroslow and Wang, 1990) and DLCS (Sang et al., 2005) heuristics by allowing for non-greedy selection of literals and truth assignments. On a dataset of 1011 sodukus, it is shown that a Jeroslow-Wang heuristic with a ε chance of selecting from the top k percent highest ranked candidate literals can solve sodukus in much fewer operations than the baseline-DPLL implementation. In contrast, a probabilistic extension on DLCS which selects truth assignments based on a probability distribution governed by frequency count and clause length drastically slows down the algorithm with few remaining unsatisfied clauses.

A natural extension to our work is to control for the search behavior of probabilistic DLCS by monitoring the amount of recently assigned truth values. If all literals within a time-window have been assigned a particular value, it becomes reasonable to explore other options with a larger probability. The probability of assigning a truth value should therefore be discounted by the current average truth assignment. This could prevent the algorithm from setting values to *true* if it has already set most of the recently selected literals to *true*, and vice versa.

References

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397.

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *J. ACM*, 7(3):201–215.

G. Jeroslow, R. and Wang, J. (1990). Solving propositional satisfiability problems. *Ann. Math. Artif. Intell.*, 1:167–187.

Marques-Silva, J. (1999). The impact of branching heuristics in propositional satisfiability algorithms. In *EPIA*.

Sang, T., Beame, P., and Kautz, H. (2005). Heuristics for fast exact model counting. In Bacchus, F. and Walsh, T., editors, *Theory and Applications of Satisfiability Testing*, pages 226–240, Berlin, Heidelberg. Springer Berlin Heidelberg.

SS, R. K. and Bhaya, G. R. (2004). A comparative study of algorithms for propositional satisfiability.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83.