Exploring the effect of an intelligent heuristic on a SAT-based Sudoku solver

Victor Retamal Guiberteau[1] and Lars Woudstra[1]

MSc Artificial Intelligence, Vrije Universiteit, Amsterdam, The Netherlands
{v.retamalguiberteau,l2.woudstra}@student.vu.nl

Abstract. SAT solvers are widely used to solve NP-complete problems. The SAT solver in this paper solved a problem of a CNF propositional formula implementing the Davis-Putnam-Logemann-Loveland algorithm, a backtracking algorithm. A version utilizing Minimum Residual Values heuristic is implemented to use intelligent selections when picking the right literals. This research compares this intelligent heuristic with a simple heuristic and random literal picking in a DPLL algorithm taking Sudoku puzzles expressed as SAT problems as input. The results show a significant difference between the intelligent heuristic and both simple heuristic and random literal picking in the number of backtracks and time. This means that an intelligent heuristic improves the efficiency of a SAT solver when solving Sudoku puzzles.

Keywords: AI · Heuristics · SAT · DPLL · Sudoku · SAT · Jeroslow-Wang · MRV · Propositional logic

1 Introduction

The Boolean satisfiability problem (SAT) consists of determining if exists an interpretation that satisfies a given Boolean formula. This means it checks the feasibility of the variables of a given Boolean formula to be replaced by values True or False in a way that the formula evaluates to True. When a formula evaluates as True, we can say that it is satisfied. SAT problem is the first problem to be proven to be NP-complete [3]. That means that all problems catalogued as NP-complete are as hard to solve as SAT. Heuristic algorithms denominated SAT solvers make an approximation to solve SAT by solving some problem instances.

SAT solvers have found many practical applications over the last few years, like AI planning, circuit testing and model checking [6]. Currently, SAT solvers also find application in more practical algorithms [6]. Most of them rely on the principle of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm. Mainly when in the mid-'90s, new techniques were found to improve the efficiency of the algorithm, like memory management and phase selection [2], which allows the algorithm to solve even larger problems. These changes in heuristics still cause improvements in nowadays SAT solvers [7]. The principle of changing heuristics relies on choosing the best variable in the Boolean formula to make it either true or false so that the solution is found most efficiently.

A SAT solver can thus be used for every problem that has elements that can either be true or false, such that it makes the complete propositional formula true or false. Sudoku puzzles are widely used as input for SAT solvers since the rules can easily be transformed into Boolean variables [8]. It is a placement puzzle of an originally 9x9 grid, made up of 3x3 sub-grids that hold numbers between 1 and 9 so that every sub-grid, every row, and every column has unique numbers.

This research explores the impact of a specific heuristic based on Minimum Remaining Values(MRV), comparing it with two classical, not intelligent heuristics as Random Naive selection and Jeroslow-Wong. We implemented three variations of the DPLL algorithm to generate three SAT solvers to solve Sudoku puzzles. These three variations use different heuristics (Random Naive, JW and MRV). In order to study this implication, we defined a research question as follows:

Research Question *Can an intelligent heuristic increase the efficiency of SAT solver when solving Sudoku puzzles expressed as SAT problems?*

We defined intelligent heuristics as the ones utilizing information about the search space to conduct a refined search. In the Sudoku puzzles case, the intelligent heuristic will make use of the information provided by the clues to take an optimal decision on which literal choose to split next.

The hypothesis to this research question is that the intelligent heuristic will increase the efficiency of SAT compared to both other picking strategies.

2 Methodology

2.1 Sudoku as SAT problem

Sudoku puzzles are interesting inputs to test the SAT solver algorithm due to the possibility of transcribing the rules to complete the puzzles into logical expressions in the form of Clause Normal Forms (CNF). In order to build a propositional formula Φ we need to introduce the variable $x_{r,c,v}$, for each row $r = 1, \ldots, n$, column $c = 1, \ldots, n$ and value $v = \ldots, n$ in a way that, if $x_{r,c,v}$ is True, the row r, column c has the value v. We need to encode the clauses of the Sudoku puzzle to allow the SAT to arrive to a proper solution. We introduce sets of clauses so, $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6$ where:

1. Each entry has at least one value: $C_1 = \bigwedge_{1 \leq r \leq n, 1 \leq c \leq n} (x_{r,c,1} \vee \ldots \vee x_{r,c,n})$
2. Each entry has at most one value: $C_2 = \bigwedge_{1 \leq r \leq n, 1 \leq c \leq n, 1 \leq v < v' \leq n} (\neg x_{r,c,v} \vee \ldots \vee \neg x_{r,c,v'})$
3. Each row has all the numbers: $C_3 = \bigwedge_{1 \leq r \leq n, 1 \leq v \leq n} (x_{r,1,v} \vee \ldots \vee x_{r,n,v})$
4. Each column has all the numbers: $C_4 = \bigwedge_{1 \leq c \leq n, 1 \leq v \leq n} (x_{1,c,v} \vee \ldots \vee x_{n,c,v})$
5. Each block has all the numbers:
   $C_5 = \bigwedge_{1 \leq r' \leq \sqrt{n}, 1 \leq c' \leq \sqrt{n}, 1 \leq v \leq n} (\bigvee_{(r,c) \in B_n(r',c')} x_{r,c,v})$ where
   $B_n(r', c') = \{(r'\sqrt{n} + i, c'\sqrt{n} + j) | 0 \leq j < \sqrt{n}\}$
6. The solution respects the given clues: $H : C_6 = \bigwedge_{(r,c,v) \in H} (x_{r,c,v})$

2.2 DPLL algorithm

To solve 9x9 Sudoku puzzles, the algorithm DPLL was implemented. The DPLL algorithm solves the problem by filtering out literals in the whole set of CNF clauses that belong to this problem. Some parts of the algorithm we want to consider:

Pure literal elimination : whenever a pure literal is encounter in Φ, this literal can be set to True, maintaining the satisfiability of Φ

Unit propagation : this states that if there is a unit clause, that literal can be set to True (or false in case of the negation), and all the clauses containing this particular literal should be removed.

Every time a literal has been picked and removed, the algorithm should check whether the problem is still satisfiable. If it is not, the algorithm branches back to the previous set of rules and tries the negation of the picked literal, until it finds a literal that causes a simplification of the rules that is still satisfiable. If there is no such literal, the problem is not satisfiable [9].

2.3 Heuristics

Because picking random literals is often not the most efficient way to solve a problem, methods are made to solve the SAT problem most efficiently [4]. These methods are called heuristics. In this research, two heuristics were implemented for two DPLL version and one version ran with random selection.

Jeroslow-Wang heuristic It scores all the unique literals by looking at the length of the clauses that the literal occurs in with the following formula:

$$J(I) = \sum_{I \in \varpi, \varpi \in \Phi} 2^{-|\varpi|} \tag{1}$$

Where ϖ represents the length of the clause. The main idea behind this heuristic is to remove the literal that occurs in the longest clauses, so that shorter clauses that consist of less literals remain and the problem will be solved in less steps [5].

Minimum Remaining Values Minimum Remaining Values (MRV) strategy iterates through every unassigned literal in a SAT problem and compares their domain sizes before selecting the literal with the minimum remaining values as the following literal to be solved. If two or more literals end with the same remaining value, the decision for this experiment is to select the literals in static order of appearance. MRV is a heuristic strategy that helps confirm if an assignment is doomed to fail in the early stages of the search. This heuristic achieves

this by applying forward checking (FC) to preserve the valid candidates for the unassigned variables as the solving process progresses. While backtracking still occurs with MRV, it is considerably reduced by FC. As a result, the MRV strategy appears to accelerate the problem-solving process by a factor of more than a thousand (1000) times compared to static or random variable selection [1]

Algorithm 1 MRV pseudocode

```
1: SelectNextState()
2: LessMRV ← int.MaximumValue
3: for Cell in SudokuBoard do
4:     if Cell.Value = NotAssignedCell and Cell.Candidates.Count < LessMRV
   then
5:         potential.Cell ← Cell
6:         LessMRV ← Cell.Candidates.Count
7:     end if
8: end for
9:
10: if potentialCell == Null then
11:     Current partial solution inconsistent, Backtracking
12:     Return Null
13: else
14:     Return PotentialCell
15: end if
```

2.4 Experiment Design

To carry out this research, one experiment involving a set of Sudoku puzzles was designed. The set of Sudoku puzzles used had two types of puzzles with different complexity between them. The easy Sudoku puzzles contained on average 21 clues, while the hard Sudoku puzzles included on average 16 clues. These two types of Sudoku puzzles were included in a Dataset containing 1035 different Sudoku puzzles, with 1000 easy puzzles and 35 hard ones. We tested each version of DPLL (random, Jw and MRV) in the entire Dataset and recorded the time expended in solving a Sudoku puzzle and the number of backtracks the algorithm needs before arriving at an optimal solution.

Statistical analysis was carried out, collecting descriptive statistics on the performance of each version of DPLL.

3 Results

During the two experiments that compared the performance of the different versions of DPLL implemented for this research, we obtained the results shown in Table 1. and Table 2.

Focusing on the descriptive statistics of the results, we discover a slightly superior performance in easy Puzzles by the DPLL version implemented with MRV having an average number of backtracks of $28.1 \pm 24.5(std)$ while the versions with JW and Random selection got $534.2 \pm 1115.1(std)$ and $87.2 \pm 127.7(std)$ respectively.

Table 1. Easy Sudoku puzzles Experiments Results

	Mean Backtracks	**Mean time**
MRV	28.1 ± 24.5 (std)	2.3 s
JW	534.2 ± 1115.1 (std)	2.9s
Random	87.2 ± 127.7 (std)	2.2s

Looking at the hard puzzles, we see a great superiority in time and complexity to find a solution for the Sudoku puzzle. The DPLL version with MRV heuristic completed the puzzles with $191.8 \pm 221.8(std)$ backtracks and $5.2 \pm 2.5s$ while the JW and Random versions obtained $71328.2 \pm 88395.1(std)$ and $1661.6 \pm 2792.6(std)$ for backtracking respectively; $419.3 \pm 868.2s$ and $13.1 \pm 19.1s$ where the results obtained for time.

Table 2. Hard Sudoku Puzzles Experiments Results

	Mean Backtracks	**Mean time**
MRV	191.8 ± 221.8 (std)	5.2 ± 2.5s
JW	71328.2 ± 88395.1 (std)	419.3 ± 868.2s
Random	1661.6 ± 2792.6 (std)	13.1 ± 19.1s

To confirm this, significance tests need to be performed. Firstly, it needs to be determined whether the data is normally distributed. Therefore, a Shapiro-Wilk test for normality has been performed. The distributions for the three versions of DPLL are non-normal; hence a Wilcoxon test for unpaired data was performed. The results are given in Table 3 and Table 4.

Table 3. Results Shapiro-Wilk Test for Normality and Significance Test for easy puzzles

To compare	p-value Shapiro	Test	p-value
Time	DPLL MRV - 0.71 — DPLL - 0.83	Mann–Whitney U	<0.001
Time	DPLL MRV - 0.71 — DPLL JW - 0.43	Mann-Whitney U	<0.001
Backtracks	DPLL MRV - 0.74 — DPLL - 0.54	Mann–Whitney U	<0.001
Backtracks	DPLL MRV - 0.74 — DPLL JW - 0.46	Mann-Whitney U	<0.001

Table 4. Results Shapiro-Wilk Test for Normality and Significance Test for Hard puzzles

To compare	p-value Shapiro	Test	p-value
Time	DPLL MRV - 0.81 — DPLL - 0.56	Mann–Whitney U	<0.001
Time	DPLL MRV - 0.81 — DPLL JW - 0.46	Mann-Whitney U	<0.001
Backtracks	DPLL MRV - 0.76 — DPLL - 0.56	Mann–Whitney U	<0.001
Backtracks	DPLL MRV - 0.76 — DPLL JW - 0.76	Mann-Whitney U	<0.001

4 Discussion

As stated in the results section above, there is a significant difference in the number of backtracks between the DPLL and DPLL MRV heuristic for solving both easy and hard Sudoku puzzles ($p < 0.001$). The DPLL MRV is thus significantly faster than the DPLL algorithm, which implies that the algorithm benefits from using the MRV heuristic in solving Sudoku puzzles. This was also hypothesized and embedded in previous literature [1].

Is also interesting to look at time differences between easy and hard Sudoku puzzles. For easy Sudoku puzzles, all three versions of DPLL are tied in 2-3 s, but for the hard Sudoku puzzles the time differences increase greatly. One cause could be the duration of picking a literal can cause this: random picking is faster than scoring all literals and then picking the best literal since the algorithm has to look over every element of the remaining problem. The algorithm thus only benefits from picking a score literally when the problem gets harder. Another cause for the difference in MRV, could be the Forward looking strategy.

When looking at the number of backtracks between the DPLL and the DPLL MRV algorithm, the DPLL MRV solved the problem with significantly fewer backtracks compared to the DPLL algorithm for both easy and hard Sudoku puzzles ($p < 0.001$). Both results show that the intelligent heuristic researched in this paper increases the efficiency of the SAT solver when solving Sudoku puzzles.

The opposite is true for the difference in both time and number of backtracks between the DPLL and DPLL JW algorithm ($p < 0.001$). The DPLL JW needed more backtracks and more time than the standard DPLL algorithm in solving hard Sudoku puzzles and more backtracks for the easy ones. Therefore, the JW heuristic does not increase the efficiency of the SAT solver compared to a standard DPLL using random literal picking.

Also, a significant difference is found in both time and number of backtracks between the DPLL MRV and DPLL JW algorithm in hard Sudoku puzzles and backtracks difference in easy ones, both ($p < 0.001$), which showed that the DPLL MRV was faster and needed fewer backtracks than the DPLL JW algorithm. This all brings together that an intelligent heuristic can increase the efficiency of a SAT solver when solving Sudoku puzzles expressed as SAT problems.

The performance of the DPLL algorithm given a specific heuristic is significantly impacted when facing more complex problems. In this research, we could

see how the time performance with the easy puzzles in the Dataset is almost similar for the three versions. However, when the complexity of the puzzles increase, naive approaches or simple heuristics require more computational power to arrive at the same solution as the intelligent heuristic tested in this experiment. One of the main reasons for this is the characteristic Forward Checking done by MRV, preventing unprofitable literal selection from being solved next. Thanks to FC, as expected, the DPLL with MRV can utilize information within the problem to make intelligent decisions. However, the MRV heuristic could still be improved. When calculating the minimum remaining values, the heuristic has some flaws and untapped potential. The selection is made by static order of the clauses, but more intelligent selections could be made to improve this particularity, and more information from the problem space could be included in the process to find an optimal literal for the next split to be solved.

5 Conclusion

The intelligent heuristic increases the performance of the DPLL algorithm for SAT solving significantly compared with vanilla DPLL or the version with JW. Future research could be done in intelligence selection of literals to split next using the information on the problem space.

References

1. Abuluaih, S., Mohamed, A., Annamalai, M., Lida, H.: Fog of search resolver for minimum remaining values strategic colouring of graph. International Conference on Soft Computing in Data Science pp. 201–215 (2018)
2. Bayardo Jr, R.J., Schrag, R.: Using csp look-back techniques to solve real-world sat instances. Aaai/iaai pp. 203–208 (1997)
3. Cook, S.: The complexity of theorem-providing procedures. Proceedings of the third annual ACM symposium on Theory of computing pp. 151–158 (1971)
4. Dershowitz, N., Hanna, Z., Nadel, A.: A clause-based heuristic for sat solvers. International conference on theory and applications of satisfiability testing pp. 46–60 (2005)
5. Jeroslow, R.G., Wang, J.: Solving propositional satisfiability problems. Annals of mathematics and Artificial Intelligence pp. 167–187 (1990)
6. Mahajan, Y.S., Fu, Z., Malik, S.: Zchaff2004: An efficient sat solver. International Conference on Theory and Applications of Satisfiability Testing pp. 360–375 (2004)
7. Shaw, A., Meel, K.S.: Designing new phase selection heuristics. International Conference on Theory and Applications of Satisfiability Testing pp. 72–88 (2020)
8. Weber, T.: A sat-based sudoku solver. LPAR pp. 11–15 (2005)
9. Wetzler, N., Heule, M.J., Hunt, W.A.: Mechanical verification of sat refutations with extended resolution. International Conference of Interactive Theorem Proving pp. 229–244 (2013)