SUDOSAT: a SAT-based Sudoku Solver

Tim Smit
University of Amsterdam
2564567
tim.smit2@student.uva.nl

Janne Spijkervet
University of Amsterdam
2655256
janne.spijkervet@student.uva.nl

March 3, 2019

Abstract

This paper describes a *Sudoku* solver based on boolean satisfiability. In this paper we describe two different aspects of the performance of our solver, the solving performance of (1) different decision heuristics and (2) compared to the complexity of a *Sudoku*.

`SUDOSAT` utilizes the Davis-Putnam (DP) search algorithm and an efficient method of Boolean Constraint Propagation, along with four decision heuristics, `RAND`, `GRAB FIRST`, `DLIS` and `RDLIS`, to achieve a higher performance gain in solving *Sudoku* problems.

Keywords— Boolean satisfiability, Decision heuristics, DLIS, RDLIS

1 Introduction

The Boolean Satisfiability (SAT) problem attempts to determine if a satisfying variable assignment exist for a Boolean formula. If there exists a variable assignment that evaluates the formula to a `TRUE` state, we call the formula *satisfiable*. SAT was proven to be the first and is one of the key NP-complete problems today [1]. It is used extensively in domains including Electronic Design Automation, logistics and AI, resulting in many practical SAT applications being proposed and implemented (WalkSAT, MaxSAT, BerkMin). Most SAT-solver employ the Davis-Putnam (DP) backtrack search algorithm and one or a combination of decision heuristics.

In this paper, we focus on optimizing `SUDOSAT` for solving *Sudoku* puzzles of different levels of difficulty. A *Sudoku* has a 9×9 grid enclosing nine 3×3 sub-grids (regions). A *Sudoku* puzzle is satisfied when each row, column and region contains only one instance of each digit ranging from $1 - 9$. There are approximately 6.671×10^{21} valid *Sudoku* grids [2], which makes it infeasible to use a naive backtracking algorithm. Like many SAT-solvers, `SUDOSAT` also employs the Davis-Putnam backtrack search algorithm in combination with a decision heuristic.

In this paper, the performance of four decision heuristics for solving *Sudoku*'s are compared. We hypothesize a decision heuristic taking into account more dynamic information provided by the backtracking search algorithm will perform better than a random decision heuristic.

This paper also compares the solving performance with regard to the complexity of a *Sudoku* puzzle. We hypothesize a *Sudoku*'s complexity has a non-linear correlation with the performance of the SAT solver.

1.1 Basic Davis-Putnam Backtrack Search

The Davis-Putnam backtrack search algorithm [3] works according to the following set of operations:

Algorithm 1 Davis-Putnam-Logemann-Loveland Backtrack Search Algorithm

Input Set of clauses Φ
Output Truth assignment

1: **procedure** DPLL(Φ)
2: **if** $\Phi == []$ **then**
3: **return True**;
4: **if** $\{\}$ in Φ **then**
5: **return False**;
6: **while** l in Φ **do** ▷ unit clause in Φ
7: $\Phi \leftarrow$ `unit-propagate`(l, Φ)
8: **while** l pure in Φ **do** ▷ pure literal in Φ
9: $\Phi \leftarrow$ `pure-literal-rule`(l, Φ)
10: $l \leftarrow$ `variable-assignment`(Φ)
11: `split-count` = `split-count` + 1
12: **return** `DPLL`$(\Phi \wedge \{l\})$ or `DPLL`$(\Phi \wedge \{\neg l\})$

The `variable-assignment()` operation selects a not yet assigned variable and gives it a value according to the selected decision heuristic. When making a decision, its result is taken out of the unassigned variable `stack` and the `split-count` variable is incremented by one. When the DPLL algorithm does not return `TRUE`, i.e. the set of clauses Φ is not yet satisfiable, the `backtracking-count` variable is incremented by one. Clauses consisting of only literals with value `FALSE` and one unassigned literal are *unit clauses*, i.e. the assignment for the remaining literal must be `TRUE`. The Boolean Constraint Propagation (BCP) operation identifies *unit clauses* containing l, removes them from Φ and removes occurrences of $\neg l$ from all remaining clauses in Φ.

1.2 Decision Heuristics

A decision is a determination of which variable and its state should be selected, each time `variable-assignment` is run. In this paper, four common decision heuristics for SAT solvers are evaluated. The most elementary strategy performs a random decision among the unassigned variables, which we refer to as `RAND`. This decision heuristic is chosen as a baseline to compare the performance of the other heuristics to. Most branching heuristics take more information into account, e.g. the number of literals in unresolved clauses and the sizes of the unresolved clauses containing the literal. One of the most frequently used strategies is the dynamic largest individual sum (`DLIS`) heuristic, which was introduced in GRASP [4]. It selects the literal that appears in the largest number of unresolved clauses. `RDLIS` is a variation on this heuristic by performing a weighted random selection on the variables appearing most frequently in unresolved clauses. Both these heuristics take into account dynamic information provided by DP which could lead to a more informed decision based on the past branching decisions. Branching on literals satisfying mentioned constraint will maximize the effect of BCP and increases the likelihood of reaching an unsatisfiable position early on in the chosen branch.

Since *Sudoku*'s favor negative assignments of variables due to its problem nature, i.e. there are more negative than positive variable assignments in the solution of a *Sudoku*, `DLCS` and `RDLCS` are not evaluated in this study. In the datasets described in Section 2.1, there are approximately 70% less positively than negatively assigned occurrences of the same variable in all clauses. These decision heuristics rely on accumulating the number of positive and negative assignments of a variable, while in contrast *Sudoku*'s will clearly benefit from the separation of the two.

Lastly, we introduce a `GRAB FIRST` heuristic, which picks the first unassigned variable from the aforementioned `stack`. This heuristic is chosen to compare against a fully random heuristic (`RAND`) and a dynamically informed and partly random heuristic (`DLIS` and `RDLIS`).

2 Method

The representation of a *Sudoku* and its rules are transformed into conjunctive normal forms and are satisfiable if and only if the *Sudoku* has a solution. We make the assumption all given *Sudoku*'s are satisfiable.

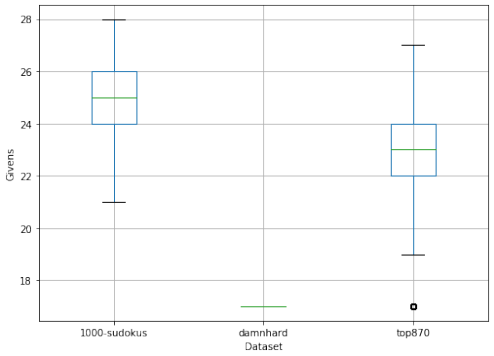

Figure 1: Number of givens in each dataset

2.1 Datasets

Three *Sudoku* datasets with varying degrees of complexity are used to compare the decision heuristics. The `top870` dataset contains 870, the `1000-sudokus` contains 1000 and the `Damnhard` dataset contains 35 *Sudoku*'s respectively.

In order to independently measure the performance of the decision heuristics, an assessment on the complexity of the given problem should be made. The number of different states in the search space of a *Sudoku* is strongly correlated to the number of non-given values [5]. While aforementioned study shows this measure alone is not sufficient to predict difficulty rating, an estimation can be made for a given *Sudoku* problem. Therefore, the amount of givens in a *Sudoku* is used in this study to measure its complexity. The `Damnhard` dataset is considered to yield the most difficult *Sudoku*'s, while it exclusively contains *Sudoku*'s with 17 givens. The `top870` dataset contains more difficult *Sudoku*'s than the `1000-sudokus` dataset, with median givens of 23 and 25 respectively.

2.2 Performance Measure

With the strategies mentioned in Section 1.2, it is important to understand how to evaluate them. In most comparative studies on branching heuristics, the number of decisions performed for a given *Sudoku* is used. The reasoning behind this measure, is that a fewer amount of decisions should imply smarter decisions were made. For this study, the number of decisions performed for a given *Sudoku* problem ($d_{\texttt{STRATEGY}}$) is therefore naturally chosen as a measure to measure the performance of the decision heuristic. The performance of each decision heuristic is compared to the `RAND` strategy, as shown in equation 1.

It should be noted that it is argued [6] not all decisions yield an equal number of BCP operations, i.e. a shorter sequence of decisions may actually lead to more BCP operations than a longer sequence of decisions. This challenges the reliability of this measure. Another challenge to consider, is that not all decision heuristics have an equal amount of computational overhead. The best decision heuristic may be the slowest if its overhead is significant. In the end, we want to know which strategy has the fastest performance. This could be taken into account for further research.

$$P_{\texttt{STRATEGY}} = \frac{d_{\texttt{STRATEGY}}}{d_{\texttt{RAND}}} \tag{1}$$

2.3 SAT Design Decisions

Similarly to many other solvers, `SUDOSAT` employs the deletion of conflicting clauses to avoid the use of a lot of memory. It also utilizes the Davis-Putnam search algorithm and an efficient method of Boolean Constraint Propagation. Each literal is given its own memory segment, i.e. a pointer to a `literal class`. A new set of clauses is made by replacing the literals with a pointer referring to the literal. Each clause in the set of clauses Φ also gets its own memory segment, i.e. a pointer to a `clause class`. During this process, an inverted index is constructed which links each literal pointer to the pointers of the clauses it occurs in. During BCP, getting the memory addresses of the clauses containing the given literal is faster than performing full iterations on the total set of clauses.

A downside to this method is the bookkeeping of the inverted index, since a full copy of all memory addresses must be made each time DPLL backtracks. This increases the execution time. A solution to this problem is left for further research.

	top870	1000-sudokus	Damnhard
RAND	66.34 (61.98)	6.74 (9.21)	99.45 (105.78)
GRAB FIRST	69.68 (91.23)	7.07 (11.20)	115.27 (148.91)
DLIS	298.05 (232.18)	58.95 (124.20)	292.25 (278.01)
RDLIS	75.24 (99.51)	7.47 (12.06)	192.94 (199.29)

Table 1: Mean and (standard deviation) of the number of decisions in different heuristics and datasets

2.4 Experimental Design

Since both RAND and RDLIS are non-deterministic decision heuristics, the experiment is run 10 times on every *Sudoku*. This will allow the comparison of their performance measure and the deterministic decision heuristics (GRAB FIRST and DLIS).

To compare the performance of each decision heuristic with the *Sudoku*'s complexity, every *Sudoku* is assigned to a *bin* corresponding to its number of givens.

To determine a correlation between the complexity of the *Sudoku*'s and the performance of the SAT solver, all *Sudoku*'s are again binned according to the number of givens. The average number of decisions from all datasets and strategies are accumulated to obtain one measure, from which the shape of the line can be obtained.

3 Experimental Results

We have gathered the mean number of decisions and standard deviation per decision heuristic for each dataset in Table 1. The RAND heuristic outperforms all other heuristics in every dataset. RDLIS performs better than DLIS. We argue that because of this, introducing a probability ϵ of choosing an unassigned literal or employing a series of weights increases the performance of the SAT solver. This stands in contrast to our belief that decision heuristics taking into account more dynamic information perform better than random selection.

When looking at Figure 2, we can see a downward trend in the number of decisions for every strategy except DLIS. RDLIS has the best performance on *Sudoku*'s with the highest amount of givens, while GRAB FIRST has the best performance at 19 givens. RAND performs steadily across all givens.

Figure 3 shows the performance of all decisions heuristics compared to the RAND baseline. DLIS performs unexpectedly bad compared to its variant RDLIS. RAND is only outperformed by GRAB FIRST at 19 givens and by RDLIS and 28 givens.

Figure 4 describes the accumulated average number of decisions of all datasets and strategies. A strong downward slope is observed from 17 to 20 givens, suggesting a decrease of the search space significantly increases the performance of the SAT solver. An increase of 250% is observed from $17 - 20$, while from 20 givens onward, the slope is less steep, but still yields an increase in average performance of 200% from $20 - 24$. This suggests a non-linear correlation between the *Sudoku*'s complexity and the SAT solver performance.

Considering the large variance in performance for *Sudoku*'s, there is little room for statistical significance in the results.

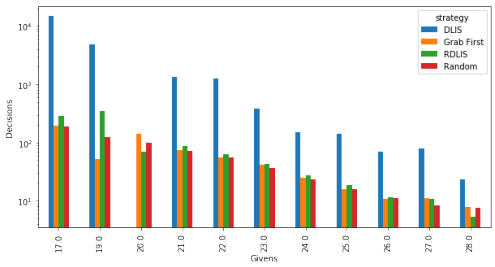

Figure 2: Number of decisions in four decision heuristics, grouped by number of givens

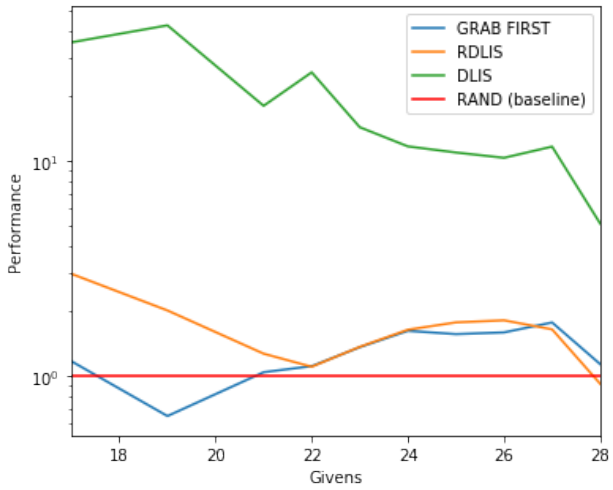

Figure 3: Performance compared to the `RAND` baseline for all number of givens

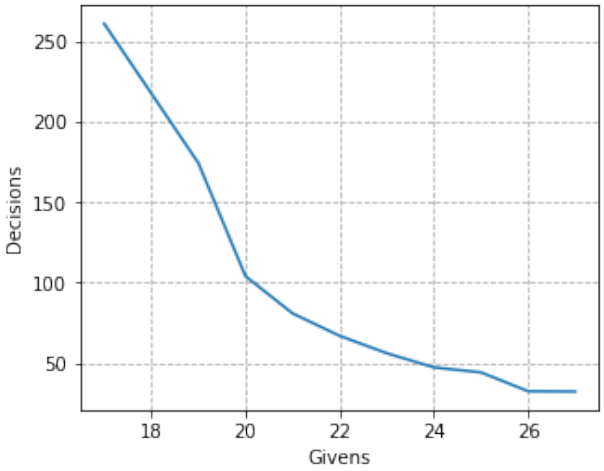

Figure 4: Accumulated average number of decisions of all datasets and strategies, with the number of givens

4 Conclusion

This paper described a *Sudoku* solver based on boolean satisfiability. Two different aspects of the performance of the solver were studied, the solving performance of (1) different decision heuristics and (2) compared to the complexity of a *Sudoku*. We demonstrated worse performances among decision heuristics taking into account more dynamic information than a random decision heuristic for *Sudoku* problems. We have also shown the complexity of a *Sudoku* has a non-linear correlation with the performance of the SAT solver. An increase of 250% is observed from $17 - 20$ givens, and 200% from $20 - 24$ givens.

Further research could investigate the addition of a heuristic designed with *Sudoku*-solving-strategies in mind. In addition to this, a better duplication algorithm for memory addresses in a backtracking search algorithm is needed to improve the execution time of the SAT solver.

References

[1] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[2] Bertram Felgenhauer and Frazer Jarvis. Enumerating possible sudoku grids. 07 2005.

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

[4] João P. Marques Silva and Karem A. Sakallah. Grasp—a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '96, pages 220–227, Washington, DC, USA, 1996. IEEE Computer Society.

[5] Sian Jones, Paul Roach, and Stephanie Perkins. Sudoku puzzle complexity. pages 19–24, 03 2011.

[6] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff:

Engineering an efficient sat solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, 2001. ACM.