Sudoku your way into SAT problems

Groot Kormelink, Joseph - 11784903 (UvA)
josephgk@hotmail.nl

Rostov, Maxim - 11808470 (UvA)
maxim.rostan@gmail.com

March 3, 2019

1 Introduction

In this paper we provide an overview of the experiments conducted with a set of boolean satisfieability (SAT) solvers. These tools work on the problems which are NP-complete and utilize the first-order logic rules to search through a space of possible solutions. SAT solving is an active area of research with applications in logistics, chess, planning and (product) verification. Though, boolean satisfiability solvers have been actively developed since the beginning of the 20th century, the challenges of complexity in NP-problems hang around even the most modern and optimized solvers.

In this paper we aim to provide a comparative study of the performance of various SAT solving heuristics and their behaviour under problems of different complexity. The SAT solvers discussed in the paper are adaptations of the Davis-Putnam (DP) algorithm [1; 2] which provides a basis for many modern SAT solvers [3]. Heuristics discussed include CDCL [5] and MOMs, and our own alternative to CDCL called Chronological CDCL. The conducted experiments inspect how different heuristics compare to each other in the context of Sudoku solving. Sudoku problems and rules are used in DIMACS format and adjusted to represent different levels of complexity of SAT problems. Although we are solving Sudokus, our SAT solvers are fully sound and complete, and the results should generalize well to other domains.

Based on on the previous literature we expect the combination of CDCL and MOMS to provide the best result [4]. CDCL uses conflict learning to better predict what clauses lead to unsatisfied formulae and MOMS heuristic solves the "which-variable-to-split-on" problem. In an attempt to further improve efficiency of SAT solving we adjust the standard CDCL model to account for chronological backtracks when encountered a conflict. This might not minimize the total amount of splits, but does seem to minimize the total complexity of the algorithm.

In order to see whether our logic regarding the SAT solving of Sudoku problems was correct we introduce a hypothesis.

- We hypothesize that chronological CDCL outperforms the other algorithms if we look at computational efficiency.

To measure complexity we rely on a human expert judgment, therefore, we use already labeled by difficulty Sudoku dataset. The assumption here is that if a Sudoku is hard to solve for a human it will also require more efficient algorithm to solve it in a reasonable amount of time. Efficiency of an algorithm is measure by the means of two proxies: number of splits performed by the model and 'total number of satisfied clauses'. Although in the literature splits are often used as a proxy for the efficiency, we argue this measure by itself is deficient. Therefore we propose a second measure,

the total number of satisfied clauses, which counts all the satisfied clauses which were flipped to $True$ during the run-time of the algorithm. In order to understand why we propose this metric, consider an example: let assume we have two algorithms, algorithm A B. Algorithm A performs a split, yielding 1000 satisfied clauses. Now after these 1000 clauses, it finds a conflict and has to backtrack, unsatisfying all 1000 clauses. Now it makes another split, yielding 1500 satisfied clauses. So algorithm A solved the problem after satisfying 2500 clauses. Now algorithm B performs 5 splits, satisfying 500 clauses. It backtracks and performs another split, yielding 1500 satisfied clauses and solving the problem. The total number of satisfied clauses for Algorithm B is 2000. Now, based on the total amount of splits Algorithm A would be considered the better algorithm, while we argue that algorithm B is the better algorithm. This is because it is not hard to set a random variable to a truth value, the hard thing is satisfying all those clauses afterwards. Thus we argue that total number of satisfied clauses should also be taken into account when evaluating an algorithm.

In the section 2 the reader will be first introduced to the heuristics examined in the paper. The following section 3 will explain in more detail the design choices which were made when conducting the experiments. Finally, results and conclusion are presented in the consequent last two section.

2 Heuristics

We have implemented DP, CDCL and chronological CDCL [1; 5]. For all these methods we have implemented the MOM's heuristic for finding variables most likely to satisfy clauses, the reason we chose this methods and the underlying principles of these methods will be shortly described here.

2.1 DP

Davis Putnam is the most basic SAT solver algorithm. it's a combination of unit propagation and random backtracking. Unit propagation happens when A clause is evaluated as false and only has one literal variable left, once this is the case the literal should be set so that the clause will be evaluated true. Because by assumption all clauses are correct, we can apply this unit propagation. As DP is a very simple algorithm, performance is not optimal, it explores many branches it has already explored before, but it is guaranteed to find a solution if it exists.

2.2 CDCL

Conflict Driven Clause Learning is an adaptation of the DP algorithm. Once A conflicting variable assignment is found the conflict is evaluated. Evaluation of the conflict leads to a new clause that will added to the already existing list of clauses. This new clause will allow the algorithm to make less splits after backtracking. less splits means less guesses, and thus most likely a better performance.

2.3 Chronological CDCL

Because CDCL backtracks multiple steps, it might in total satisfy more clauses than is needed (cumulative), chronological CDCL is an alternative that might decrease the total number of clauses satisfied. It does this by backtracking chronological like DP, but it also learns from clauses like CDCL. Because of this combination of properties we hypothesize that chronological CDCL will outperform the other algorithms.

2.4 MOMS

Maximum Occurrence of clauses of Minimum size is an heuristic for picking the next variable to split. The intuition behind this algorithm is that we should pick a balanced variable that occurs

in as many tiny clauses as possible. If we have a variable that occurs often, this increases the total amount of possible clauses that will be satisfied given this variable. These clauses should also be small, as small clauses easier become unit clauses, and thus have an higher change of propagation unit variables. The formula for this heuristic is:

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

Where $f^*(a)$ is the function that counts the amount of clauses that contain term a.

3 SAT design decision

As for every project, results are partially based on implementation details. Performance of our SAT solver is dependent on how we choose our split variables, what we set those variables to e.g. $True$ or $False$, and how we backtrack. The first design decision we took is to store our clauses and variables in two separate python dictionaries, where the clauses have indexes of the variables it contains, and the variables have indexes of the clauses they appear in. This structure allows for fast look ups ($O(n)$), and thus for solving SAT problems in a fast manner. Furthermore, the variables we will pick to split are in selected order of their number, so variable 111 will be earlier spitted than variable 333 (this does not hold for MOMS heuristic). We feel this method is better than random selection, as it ensures that our system is a deterministic one, and leaves less uncertainty about the outcome. Of-course there is no guarantee that this works better or worse than randomly selecting which variable to split, as we assume that our SAT problems have no inherent order in which variables should be assigned. Furthermore, on assignment, every variable is first set to $True$. Based on preliminary research, the data has shown that this produces better results than first setting the value to False. Bear in mind that this setting is arbitrary, and our evidence is based on our selected domain of Sudokus, in problems outside of this scope there might be cases where setting the values to $False$ might work better. Regardless of which variable is set first, the algorithms produce the correct results.

4 Experimental Design

We have gathered multiple Sudokus, which have been consequently translated to DIMACS format. Every Sudoku is solved once for every algorithm, this to ensure that our measures are not biased, and all algorithms will solve Sudokus of the same difficulty. Three levels of difficulty are chosen to conduct the experiments. The easy level consist of 1060 4x4 Sudokus, the medium level is 1021 9x9 Sudokus and there are 950 9x9 hard sudokus. There is one hyper-parameter in the moms algorithm, the k that is used as power term. This k has been set to 3 after preliminary research has deemed this an appropriate value. As all algorithms are sound and complete, all Sudokus are solved and we can validly compare the results of both measures. We compare based on two measures, the amount of splits, and the total amount of clauses solved. A split happens once there is no unit propagation possible anymore, and a variable is selected, at random or through some heuristic, to be set to a boolean value.

5 Results

In order to assess the hypothesis, we perform the experiments to compare our results on the sets of different complexity. In the table 1 we see the average amount of total satisfied clauses per Sudoku and its standard deviation. Additionally, the ‘Splits‘ row indicates the amount of splits performed by an algorithm on average per Sudoku. It can be seen that for the benchmark dataset (i.e. hard sudokus with the size 9x9 cells) the proposed heuristics with chronological backtracking show a

Table 1: Summary statistics of the results from experiments. All heuristics are presented with the mean and standard deviation values for total number of clauses satisfied metric. Also, number of splits is presented to depicted to provide an idea about the average number of splits performed per Sudoku. All the results are also sub-dived according to their respective dataset (i.e. hard, medium and easy).

		DP	DP_MOMS	CDCL	CDCL_MOMS	CDCL_Chron	CDCL_Chron_MOMS
Hard	Mean	20957.76	21640.77	24440.34	26653.17	17706.08	19002.50
	STD	15984.13	14704.50	14938.42	15593.74	6227.08	6891.17
	Splits	40.95	43.12	31.44	35.56	23.65	27.93
Medium	Mean	13298.43	13157.28	13952.79	14063.51	13168.26	13216.92
	STD	2421.00	2249.20	3641.32	4539.43	1998.59	2383.51
	Splits	7.35	6.68	5.89	5.95	6.17	6.41
Easy	Mean	139.07	147.31	127.18	128.14	121.88	122.49
	STD	157.32	176.47	118.00	119.78	121.88	98.80
	Splits	0.7	0.79	0.45	0.41	0.64	0.56

lower number of total satisfied clauses. This means that those models were possibly more efficient with choosing the split variables and performing backtracking. These results are also supported by the t-test for two independent means. For the t-test the results were first normalized and α of 5% was chosen to threshold the type I error. The corresponding p-values were calculated for Chronological CDCL and MOMS Chronological CDCL versus the best not chronological technique i.e. CDCL, with a p¡0.001 we can conclude that Chronological CDCL models satisfies significantly less clauses before solving a problem than the other algorithms (validated on the Hard sudokus).Also, Chronological CDCL outperforms Chronological CDCL MOMS.

While the total number of satisfied clauses metric showed superiority of chronological backtracking models. The average number of splits does not demonstrate a significant decrease for chronological CDCL. This is indicated by statistical testing and can be visualized with a box plot, see Figure 1

In order to combine the two metrics we visualize the average amount of satisfied clauses per Sudoku for each split. As we are only interested in relative performance of the algorithms, we apply logarithmic scaling and uniform smoothing for better readability of the plot. Figure 2 clearly shows that on average chronological backtracking satisfies clauses faster while performing less splits, meaning that it chooses variables smarter when it backtracks. This leads to less clauses being reverted back to inconsistent state after each backtrack.

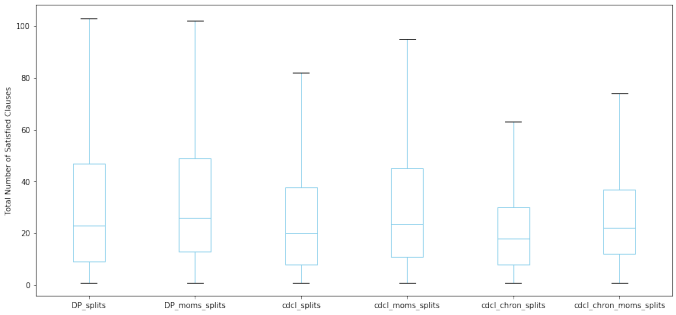

Figure 1: Box plot

6 Conclusion

With this report we introduce a newer approach to solve SAT problems of different difficulty. We compare existing implementations of DP algorithms with the new method which is enhanced with chronological backtracking. It appears that the new method outperforms its vis-a-vis's on the Sudokus which were judged difficult to solve by humans. For easier problems the results do not suggest any significant performance difference but hint on better marginal computational efficiency. For further research it might be interesting to see how these results generalize to other problem domains. Also it might be interesting to see how well chronological CDCL performs when better more elaborate variable selection heuristics are applied, as MOMS is relatively simple.

7 Graphs

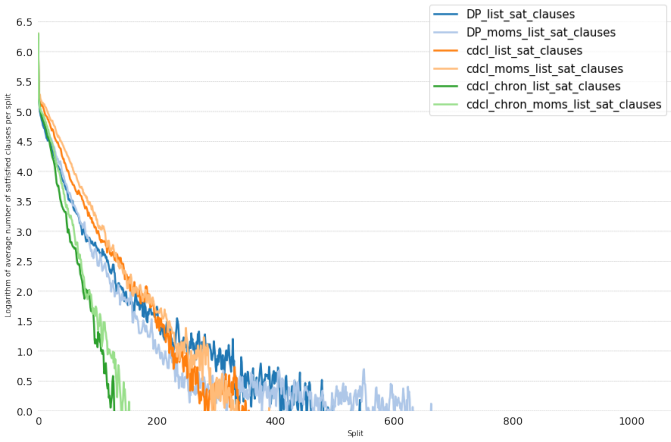

Figure 2: Average number of satisfied clauses per Sudoku per split. Difficulty hard.

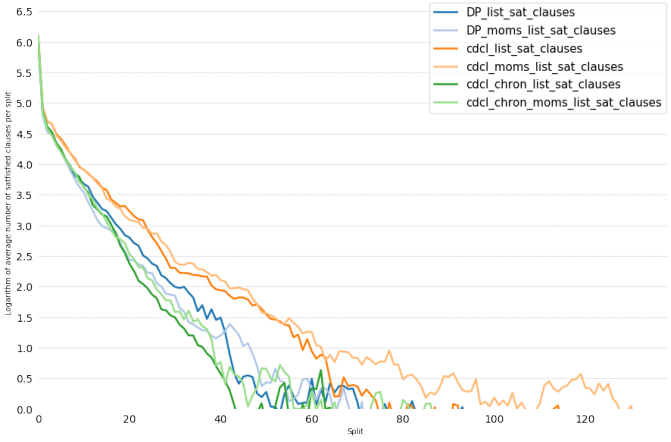

Figure 3: Average number of satisfied clauses per Sudoku per split. Difficulty medium.

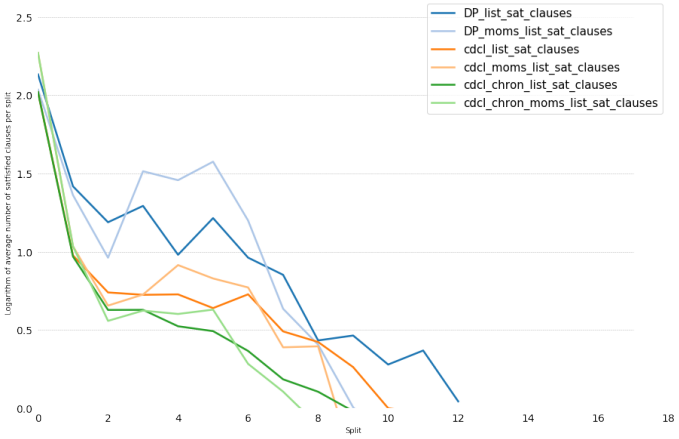

Figure 4: Average number of satisfied clauses per Sudoku per split. Difficulty easy.

References

[1] Davis, Martin; Putnam, Hilary *A Computing Procedure for Quantification Theory* Journal of the ACM. 7 (3): 201–215. doi:10.1145/321033.321034, 1960

[2] Davis, Martin; Logemann, George; Loveland, Donald *A Machine Program for Theorem Proving* Communications of the ACM. 5 (7): 394–397. doi:10.1145/368273.368557, 1962

[3] John Harrison *Handbook of practical logic and automated reasoning* Cambridge University Press. pp. 79–90. ISBN 978-0-521-89957-4, 2009

[4] Chu Min Li and Anbulagan *Heuristics Based on Unit Propagation for Satisability Problems* IJCAI, 1997

[5] P. Marques Silva, João and Lynce, Inês and Malik, Sharad *Conflict-Driven Clause Learning SAT Solver* Frontiers in Artificial Intelligence and Applications, 2009