Comparison of Probabilistic and Deterministic Splitting Heuristics for SAT Solving in Davis-Putnam Algorithm

Kai Liang (2656802) and Xiaoxiao Wen (2656130)
Faculty of Science, University of Amsterdam
{kai.liang, xiaoxiao.wen}@student.uva.nl

1 Introduction

Many problems in the real world, including NP-complete search problems which are widespread in various domains like artificial intelligence and computer science, can be formalised by propositional logic [1]. To determine the satisfiability of a propositional logic formula, in other words, to find if there is a truth assignment that makes the the formula evaluate to true, is the so-called Boolean satisfiability problem (SAT) [2]. As the first proven NP-complete problem [3], it is widely believed that all SAT problems have an exponential worst-case complexity [2]. However, lots of efforts have been and are still constantly contributed to this area to propose and implement different algorithms and heuristics to solve most SAT problems in a linear time [4, 5], among which the Davis-Putnam (DP) algorithm remains the most common prototype [4].

In this study, we implement a generic DP SAT solver and propose two probabilistic-embedded splitting heuristics which are modified on namely the Jeroslow-Wang heuristics [6] and Dynamic Largest Individual Sum heuristic (DLIS) [7]. The goal is to compare the efficiency of these heuristics in the Sudoku problem setting in terms of the number of splits and backtracking cost to find a solution. On the one hand, employing probability theories to the traditional splitting heuristics might improve the SAT solver performance since exploration is always thought to play an important role in domains such as reinforcement learning and machine learning [8]. On the other hand, exploitation, that is, adopting the most confident (i.e. rewarding) action is also essential for minimizing the overall loss, which in this case corresponds to the number of splits and backtracking. Although the exploration-exploitation trade-off mostly occurs in data-driven AI, it might also be a problem for model-driven AI in this particular case. Thus, our hypotheses are two-fold. Firstly, we expect that pure-explorative heuristics outperform pure-exploitative heuristics. Secondly, we expect that combining both exploration and exploitation in the heuristics will lead to a even better performance compared to heuristics with pure exploration.

2 Algorithm Description

In our SAT solver, a total of five different splitting heuristics are implemented, which are namely (1) the random heuristic (RAND), (2) the deterministic two-sided Jeroslow-Wang heuristic (Det-JW) [6], (3) the deterministic DLIS heuristic (Det-DLIS) [7], (4) the modified two-sided Jeroslow-Wang heuristic (Prob-JW) and (5) the modified DLIS heuristic (Prob-DLIS). Among the five implemented heuristics, (1) is pure-explorative, (2), (3) are pure-exploitative, while (4) and (5) combines the two by sampling from probability distributions based on their original evaluation functions.

2.1 General DP algorithm

The final implementation of the DP algorithm is done with Python 3, which consists of two main steps: *simplification* and *splitting*. The algorithm takes as input a file of clauses in Clause Normal Form (CNF) written in the DIMACS format, and begins with traversing through all the clauses to eliminate tautologies if any. Then, the algorithm uses recursion to iteratively assign truth values to variables until the overall evaluation becomes `True` or unsolvable.

For *simplification*, the algorithm first removes the clauses containing pure literals if any, which have a single form of existence (i.e. l or $\neg l$). Then, it eliminates unit clauses which have a length

of 1 if any and removes the negation form of those variables from remaining clauses. However, if after removal of the negation forms the lengths of some clauses become 0, it means that there is no solution under the current assignment, so the recursion algorithm will automatically backtrack.

After *simplification*, for *splitting*, the algorithm adopts one of the heuristics which will be introduced from subsection 2.2 to subsection 2.6 to assign the truth value to a variable in the remaining clauses, and eliminates clauses which are evaluated to `True`. Same as in the case of unit clauses, the algorithm also eliminates the negation form of the variable from other clauses. Again, if after removing the negation form the lengths of some clauses become 0, the algorithm will backtrack.

2.2 RAND heuristic

The random heuristic is the simplest splitting heuristic for the DP algorithm, and it splits on the variable through drawing a random sample from the collection of all available clauses. Then, the truth value is assigned to the variable based on the form of the chosen literal: if it is without the negation sign, then the variable is assigned to `True`, otherwise, to `False`.

2.3 Det-JW heuristic

The two-sided Jeroslow-Wang heuristic is a splitting heuristic for the DP algorithm which iteratively picks the most effective variable to split during the procedure of solving the SAT problems. It assigns every literal a weight of importance according to its occurrences in the clauses and the lengths of those clauses, and splits on the variable whose literal form is of the highest weight.

Specifically, the original two-sided JW heuristic computes the importance weight $J(l)$ for each literal l as:

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

where ω is a clause which contains the literal l and Ω is the set of all clauses. Then, it selects the variable whose $J(l) + J(\neg l)$ is the maximum and sets the truth value of the variable according to the scores of $J(l)$ and $J(\neg l)$: if $J(l) > J(\neg l)$, the value is set to `True`, otherwise, to `False`.

The aforementioned weight J for each literal is indicative of the impact the literal has on the clauses. Firstly, J is computed as a sum over all the clauses which contain literal l, indicating that increasing occurrences of l in the clauses is positively correlated to the importance of that literal. Secondly, for each clause ω containing l, J is negatively correlated to the length of the clause $|\omega|$, which signals that the shorter the clause the more informative the literal l is and the larger impact it has on the clauses.

2.4 Prob-JW heuristic

The probabilistic modification of JW has the same calculation formula of the importance weights J as in Equation 1. The distinctions lie in two aspects:

1. The selection of the variable according to $J(l) + J(\neg l)$.
2. The selection between l and $\neg l$ for assigning the truth value.

For the first selection, a probability distribution is created for every variable based on the sum of the weights of the literal and its negation $J(l) + J(\neg l)$. Then, the variable is picked by drawing a sample from the distribution. For the second selection, another probability distribution is similarly created for the selected literal l and its negation $\neg l$ using their corresponding weights, and the truth value is assigned by sampling from this distribution.

The intuitions behind the probabilistic modification of JW are as follows. Firstly, the original JW heuristic relies in a strong assumption that the most informative variable can be determined by the number of clauses it is in and the lengths of those clauses, which however is not necessarily

true. Hence, by adopting a probability distribution, we can weaken this assumption which might improve the performance of the overall algorithm. Secondly, the original JW heuristic might fail when two variables have the same $J(l) + J(\neg l)$ score: it might always select the less efficient one. Thus, this again signals the benefits of adopting the probabilistic adaption.

2.5 Det-DLIS heuristic

The Dynamic Largest Individual Sum heuristic is another splitting heuristic for the DP algorithm. Serving as the default splitting heuristic for GRASP [7], DLIS simply counts the number of occurrences for each literal in all clauses, and picks the literal with the largest number of occurrences.

Specifically, for each variable v, the deterministic DLIS first counts the number of occurrences of v in the positive form as $CP(v)$ and in the negative form as $CN(v)$. Then, the variable with the highest $CP(v)$ or $CN(v)$ is selected for splitting, and the truth value is assigned as in Det-JW. For DLIS, the number of occurrences of a variable in either its positive or negative form is considered to be a effective proxy of its importance.

2.6 Prob-DLIS heuristic

Similar to the deterministic DLIS, the probabilistic DLIS has the same procedure for assigning the weight for each variable. The distinction lies also in two aspects:

1. The selection of the two variables v_1 and v_2 according to $CP(v)$ or $CN(v)$.
2. The selection between v_1 and v_2 for assigning the truth value.

For the first selection, two probability distributions are created, one of which is based on the $CP(v)$ values of all variables, and the other of which is based on the $CN(v)$ values of all variables. Then, two variables v_1 and v_2 are drawn respectively from these two distributions. For the second selection, another probability distribution is created for the drawn variables v_1 and v_2 according to ones CP value and the other's CN value, from which one of them is sampled for assigning the truth value based on its form as in other heuristics.

In terms of the intuitions behind the probabilistic modification, they are similar to the ones as mentioned in subsection 2.4.

3 Experimental Setup

The dataset available in this study consists of 8 categories of Sudoku problems, which is in total approximately $22,000$ examples with different difficulty levels. However, due to the limited timeframe and computing resources, 30 samples are randomly drawn from each of the 8 categories, forming 240 test cases at the end that are assumed to be representative of the complete dataset [1].

For experiments, the DP algorithms with the five implemented heuristics are separately tested to solve the Sudoku problems in the sampled dataset, during which the statistics on the number of splits and backtracking are recorded for each test case. To statistically test the aforementioned hypotheses, the probabilistic heuristics (i.e. RAND, Prob-JW and Prob-DLIS) are each tested for 30 epochs. The deterministic heuristics (i.e. Det-JW and Det-DLIS) are however only tested for 1 epoch each because their results are guaranteed to be constant.

We use the the number of splits plus backtracking as our main metric, because it is an accurate signal on the performance of the splitting heuristics, while invariant in other irrelevant conditions (e.g. machine power).

[1] The sampling script with a fixed random seed is provided in the source code for full reproducibility.

4 Experimental Results

After the experiments, the final performance of each splitting heuristic is analyzed according to the collected statistics. In Table 1, the overall average and standard deviation over the number of splits and backtracking of each heuristic on all the test cases can be found. As could be inferred from the table, the best-performed splitting heuristic in terms of both the average and standard deviation is Prob-JW, which is closely followed by RAND and Prob-DLIS. Furthermore, the three heuristics are all around 50 times better than both Det-JW and Det-DLIS, which are the two pure-exploitative heuristics.

Table 1: *The final performance of the splitting heuristics w.r.t the number of splits and backtracking for all test cases in all epoch(es).*

	RAND	Det-JW	**Prob-JW**	Det-DLIS	Prob-DLIS
Overall Average	211.78	11113.93	**208.10**	14523.02	231.22
Overall Std.	465.56	33114.22	**451.44**	42893.91	531.19

In addition, as displayed in Table 2, we count for each heuristic the number of test cases in which it solves with the least splits plus backtracking times among the five splitting heuristics. Again, it can be inferred that Prob-JW wins the most of times, which is followed by RAND and Prob-DLIS. Although the differences between explorative heuristics (i.e. RAND, Prob-JW and Prob-DLIS) and pure-exploitative heuristics (i.e. Det-JW and Det-DLIS) are huge in both Table 1 and Table 2, we still cannot tell if heuristics with both exploration and exploitation (i.e. Prob-JW and Prob-DLIS) are significantly better than the pure-explorative one (i.e. RAND).

Table 2: *The number of wins of the splitting heuristics w.r.t the average number of splits and backtracking (excl. draws) for all test cases.*

	Number of Wins
RAND	76
Det-JW	0
Prob-JW	**91**
Det-DLIS	1
Prob-DLIS	41

Thus, a one-way ANOVA is conducted on all the heuristics with the average number of splits and backtracking they cost for each test case, and the resulting p-value is lower than 0.05, which suggests that one or more heuristics are significantly different. Then, we conduct the Tukey HSD test to identify which pairs of heuristics are significantly different from each other, and the results are shown in Table 3.

Table 3: *The results of Tukey HSD test for the heuristics on the average number of splits and backtracking for each test case (H1: RAND, H2: Det-JW, H3: Prob-JW, H4: Det-DLIS, H5: Prob-DLIS).*

pair	H1-H2	H1-H3	H1-H4	H1-H5	H2-H3
p-value	$p<0.01$	insignificant	$p<0.01$	insignificant	$p<0.01$
pair	H2-H4	H2-H5	H3-H4	H3-H5	H4-H5
p-value	insignificant	$p<0.01$	$p<0.01$	insignificant	$p<0.01$

As can be inferred from Table 3, there are indeed significant differences between explorative heuristics (i.e. RAND, Prob-JW and Prob-DLIS) and pure-exploitative heuristics (i.e. Det-JW and Det-DLIS), which matches our first hypothesis. However, the differences between heuristics with both exploration and exploitation (i.e. Prob-JW and Prob-DLIS) and the pure-explorative one (i.e. RAND) are not significant, which is partly against our second hypothesis.

5 Conclusions

In this study, a SAT solver implemented with five different heuristics based on the DP algorithm is implemented and tested on a set of Sudoku problems and the efficiency of these heuristics are compared in terms of the number of splits and backtracking. According to the results presented in section 4, the following conclusions can be drawn:

Firstly, the performance of RAND is significantly better than Det-JW and Det-DLIS. This testifies our hypothesis that pure-explorative heuristics outperform pure-exploitative heuristics.

Secondly, the probabilistic modifications upon the originally deterministic heuristics do show a significant improvement over pure-exploitative ones. However, although the overall average, overall standard deviation and number of wins of Prob-JW are all better than those of RAND, the differences between them are insignificant. Thus, the second hypothesis cannot be fully testified.

The above findings also indicate that the exploration-exploitation trade-off exists as a major problem not only in data-driven AI, but also in model-driven AI for some particular cases, which might provide an interesting inspiration for future efforts on combining symbolic and statistical methods in AI.

For further research, some directions are as follows. Firstly, as there are large variances in the current results, the dataset should be further scaled up also with the total number of experiments increased. Secondly, as the current dataset only contains Sudoku problems, it can be extended to cover a wider range of SAT problems to capture a more generic performance of the SAT solver.

References

[1] J. Rintanen, "Propositional logic and its applications in artificial intelligence," tech. rep., Aalto University, Helsinki, Finland, 2015.

[2] J. Marques-Silva, "Practical applications of boolean satisfiability," *2008 9th International Workshop on Discrete Event Systems*, pp. 74–80, 2008.

[3] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, (New York, NY, USA), pp. 151–158, ACM, 1971.

[4] J. Huang and A. Darwiche, "A structure-based variable ordering heuristic for sat," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, (San Francisco, CA, USA), pp. 1167–1172, Morgan Kaufmann Publishers Inc., 2003.

[5] F. van Harmelen, F. van Harmelen, V. Lifschitz, and B. Porter, *Handbook of Knowledge Representation*. San Diego, USA: Elsevier Science, 2007.

[6] R. G. Jeroslow and J. Wang, "Solving propositional satisfiability problems," *Annals of Mathematics and Artificial Intelligence*, vol. 1, pp. 167–187, sep 1990.

[7] J. Marques-Silva and K. Sakallah, "GRASP: a search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, pp. 506–521, may 1999.

[8] S. B. Thrun, "Efficient exploration in reinforcement learning," tech. rep., Carnegie Mellon University, Pittsburgh, PA, USA, 1992.