The effect of making SAT solvers flexible on their ability to solve Sudoku's

Rohan Zonneveld[1], Emma Cornelia Wilhelmina van Lipzig[1], and Doruk Soypaçacı[1]

Vrije Universiteit, Amsterdam 1081 HV, Amsterdam, The Netherlands

Abstract. Cognitive flexibility and Grit-Perseverance are both important human qualities that help with problem solving. Specifically, they help with solving Sudoku's. Computers already possess the Grit-Perseverance characteristic. The current study aims to improve the SAT solving algorithm DPLL for solving Sudoku's with a heuristic inspired by human Sudoku solving tactics, and a heuristic that makes the algorithm more flexible. Analysis shows these heuristics all significantly improve the DPLL algorithm's performance, but that making the algorithm more flexible does not provide additional benefit.

Keywords: SAT Solving · Human and Artificial Intelligence Problem Solving · Sudoku.

1 Introduction

1.1 The Sudoku in Science

The popular game of Sudoku has occupied the minds of many people since the 1980's. Not all of them simply solve them for fun. Mathematicians and computer scientists have worked on algorithms to solve Sudoku's for them [3] and cognitive scientists are trying to understand how other people solve them [2, 4, 6]. The game requires the player to fill in a 9 by 9 grid, subdivided into 9 squares, with the numbers 1 to 9. The grid always already contains some numbers. The challenge of the puzzle lies in the following ruleset. Each number can only appear once in every row, column and square. Of course the scientists working with Sudoku's in their research are not simply interested in solving as many Sudoku's as possible. There are two ways in which the Sudoku stands for bigger questions. Firstly, Sudoku's can be adapted to become more challenging. They can, for example, have different sizes. Although the traditional Sudoku is played with a 9 by 9 grid, it can also be played on a 16 by 16 grid. The bigger the Sudoku, the harder it is to solve. In fact, as [3] explains, they become so complex that they can no longer be solved in any realistic amount of time. This is why Sudoku belongs to the NP-complete class of problems. Solving one NP-complete problem would help solve all problems of this complexity, which is why it is useful to work on improving Sudoku solving algorithms. Such algorithms can be relevant for all NP-complete problems. Secondly, Sudoku's are very useful for studying human problem solving [2, 4]. As they can be made more and less challenging, not just by changing their size, but also varying the amount of digits which are already filled in [2, 4], they can provide insight into various cognitive processes [2, 4, 6] and the role of motivation and personality traits [4] in problem solving.

1.2 SAT Solvers

The current study uses lessons from this cognitive research on Sudoku problem solving to improve existing Sudoku solving algorithms. Typically, Sudoku's are solved using SAT solving algorithms. These first require that the problem at hand is turned into a Boolean Satisfiability (SAT) problem. This means that the problem is encoded into a set of propositional logic formulas, and that these formulas are represented in clause normal form (CNF). A SAT solving algorithm can then be used to find truth value assignments for all atoms in the propositional logic formulas that either result in no inconsistencies in the set of formulas, or finding that there is no such assignment possible. For some problems, although not for Sudoku, a SAT solving algorithm can also be geared towards finding optimal truth assignments for the atoms that result in the smallest amount of inconsistency. The same atom can appear in multiple formulas, and in the SAT solving literature, they are referred to as 'literals'. The Davis Putnam Logemann Loveland (DPLL) algorithm is a commonly used SAT solving algorithm. It searches through the entire search space of possible truth assignments by assigning true or false (depending on how it is implemented) to one literal, and continues to assign truth values to more literals, until either all literals have a truth assignment, which means the combination of truth assignments satisfies the problem, or until a formula becomes false. When this happens, the algorithm backtracks to a previous search point and switches the truth assignment for that literal [9]. Notice that the DPLL algorithm does not specify in which order the literal receive truth assignments. Without specification, this could be in a numerical or alphabetical order for example, which can result in a very inefficient way of going through the search space, depending on the problem at hand. This is why adding heuristics to specify which literals are assigned truth values first is very important.

1.3 Humans Solving Sudoku's

Inspiration for heuristics that can improve the original DPLL can be drawn from knowledge about how humans solve Sudoku's. Several studies have looked at how humans make single Sudoku decisions like what to put in one cell of the puzzle [2, 6] or where to place a single digit [2]. Filling in an entire Sudoku requires combinations of these decisions, which are strategies. Although the human process of filling in an entire Sudoku remains an open question to this day [2], these strategies, which sometimes have exciting names like 'swordfish', are widely shared amongst Sudoku enthusiasts [3]. [2] describes two tactics for making single Sudoku decisions. When the cell-based tactic is used, people consider a cell and look at relevant connected units, like the square of which the cell is a part or the rows in which the cell is placed to decide which digit could go in the cell, e.g.: 'Which number goes in the center of this square?'. For the number-based tactic, people consider a unit, like a row or a square, and try to fill in a specific number somewhere in that unit, e.g.: 'Where in this row can I place an 8?'.

1.4 Cognitive Flexibility and Grit

In [2]'s study people tend to use one tactic until it is clear it does not lead to a solution before switching to the other. With which tactic they start is potentially influenced by personal bias and experience level, but is mostly influenced by how they are asked to fill in the Sudoku. When people are required to fill in a cell they tend to start with a cell-based tactic, but when they are asked to fill in a specific digit they tend to start with a number-based tactic. When given the number-based instruction for a Sudoku decision which can only be definitively made using a cell-based tactic, twenty percent of participants particularly struggled to find an accurate solution in the given time frame. However, when a cell-based instruction, and a number-based tactic is required to find a solution, participants are generally able to switch to this tactic. Both situations show that, to find a solution within reasonable time, it is important to be able to switch tactics when a Sudoku calls for it. Therefore, cognitive flexibility, which is defined as the ease with which people can switch to a different approach or consider a different perspective [1], is important for solving Sudoku's.

Another helpful trait for solving Sudoku's is Grit. Particularly the Grit subtype Perseverance, which reflects commitment and effort towards one's enduring goal [4]. [4] have shown that individuals who score are more persevering take more attempts at Sudoku's, and this leads them to outperform individuals who score lower on the Grit-Perseverance trait on more difficult Sudoku problems. Computers do not get tired, and can, if provided with enough resources, infinitely attempt to solve a problem. Therefore they have more Grit-Perseverence than any human. Although this means that any Sudoku solving algorithm can try at a Sudoku for a very long time, this is not a perfect advantage. This is because individuals high in Grit-Perseverence are generally less cognitively flexible [4, 7]. This leads very persevering individuals to sometimes pursue the same strategies over and over, which costs time and may not lead to a solution [4, 7]. This dichotomy between flexibility and perseverance is also seen in studies of creativity, where there are two opposing paths shown to lead to creative ideas. As is described by the dual pathway to creativity model (DPCM) [8, 1] people either systematically, incrementally probe ideas within the same categories (cognitive perseverance), or flexibly go from concept to concept (cognitive flexibility). Again using the best of both worlds might help find good ideas more quickly. Therefore this study asks whether adding a cognitive flexibility heuristic to the original DPLL algorithm could improve its performance.

1.5 Research Question and Hypothesis

Specifically, the question this study tries to answer is as follows: Does adding a heuristic that makes the original DPLL switch between a number based tactic and a cell based tactic improve its performance in terms of the number of backtracks and calls to the function it needs to make to solve 9 by 9 Sudoku's? To answer this question the original DPLL algorithm is compared to a DPLL algorithm with the flexibility heuristic. Two separate algorithms, one DPLL with only a number based tactic heuristic, and the other with only a cell based tactic heuristic, are also used for the experiment, to be able to dissociate between the effects of the separate tactics and the flexibility.

As switching tactics is important to humans when solving Sudoku's efficiently, it is expected that an algorithm with smart heuristics, that is able to switch between those tactics, will be able to solve Sudoku's more efficiently than one without these tactics or the ability to switch. Therefore the hypothesis is that a DPLL algorithm with a flexibility heuristic requires less backtracks and calls to the function to solve 9 by 9 Sudoku's than a DPLL algorithm without heuristics or a DPLL with only a number or cell based heuristic.

2 Method

2.1 Materials

As described in the introduction, SAT solvers require the problem at hand to be encoded as propositional logic, in the clause normal form (CNF). For Sudoku, this means the following. Each cell is represented as the row number, column number and number inside the cell. This means that the number 9 filled in in the upper left corner would be represented as '119'. These cell representations are the atoms in the logic formulas. With these atoms, rules for a complete Sudoku can then be written in propositional logic. A complete 9 by 9 Sudoku needs to have a number from 1 to 9 in each cell for example. This rule, for the first cell, could be represented as 1. This experiment uses an encoding of all the rules for a 9 by 9 Sudoku as such, provided by [10].

$$111 \vee 112 \vee 113 \vee 114 \vee 115 \vee 116 \vee 117 \vee 118 \vee 119 \tag{1}$$

A set of 2365 9 by 9 Sudoku's, also provided by [10], is used to test the algorithms. Due to time constraints one type of Sudoku dimension is chosen. As all literature concerning human Sudoku solving that was consulted for this research used 9 by 9 Sudoku's, the same dimensions are used for the current study.

2.2 DPLL Algorithm and Heuristics

Five DPLL algorithms are implemented. Each is used to solve the same 2365 9 by 9 Sudoku's provided by [10]. The first DPLL algorithm, without heuristics, is used as a baseline to compare the other algorithms with heuristics to. The second DPLL algorithm has a heuristic that is based on the number based tactic described by [2]. The third has a heuristic based on the cell based tactic [2]. The fourth algorithm uses both the number and cell based tactic heuristic. It starts with the cell based heuristic and has a specific probability for when to switch to the other. The fifth algorithm is essentially the same, but starts with the number based tactic heuristic.

Original DPLL The basic DPLL can be broken down into several steps:

1. **Unit propagation:** clauses that contain only one literal must be satisfied to solve the problem. Thus the literal in the clause must be true. By result of this all other clauses in which this literal appears are true. Additionally, all clauses in which the negated version of the literal appears can be shortened to not contain the literal as the false literal will no longer have any influence on the satisfaction of the clause. Unit propagation of one literal affects all other clauses, which means new unit clauses can arise. This process is repeated until there are no more unit clauses.
2. **Split:** if no obvious unit clause exists a literal must be selected to assign a truth value. Many heuristics exist on how to choose this literal in a smart way to improve performance of the algorithm. In the original DPLL though, the next literal is selected at random and is assigned as true. If this assignment leads to an unsatisfiable clause the literal is assumed to be false.
3. **Backtrack:** If the assignment of a literal leads to a false CNF formula the algorithm performs a backtrack to the last assigned literal and sets its truth value to false. If both true and false lead to an unsatisfiable clause an extra backtrack is performed.
4. **(Un)satisfiable:** This step tests if the CNF formula is (un)satisfied. If there are no more clauses left to satisfy the algorithm stops and the satisfiable assignment of all literals is returned (the solution to the Sudoku). If there is a clause with no literals in the remaining set of clauses a backtrack is triggered, assigning the opposite truth value to the previously chosen literal.

Implementation Specific Choices The algorithms were implemented in python using function recursion to recursively search through the search space. The input to the program was either given in DIMACS or .sdk format. If necessary the input was transformed to be a concatenation of the Sudoku rules and the given digits as unit clauses in CNF. The set of clauses was searched to clean out the clauses if the clause contained both a literal and its' negated form (which would make the formula true in any case). Then a DPLL algorithm was initiated with the corresponding heuristic(s).

Number Based Tactic Heuristic DPLL The goal of this algorithm is to mimic the human strategy for solving Sudoku's of trying to place a certain digit in a unit (row, column or cell). It is hypothesized that humans tend to choose the row or column which already contains a lot of digits (but is not yet filled) to limit the number of options to be scanned. The heuristic chooses this row or column as the unit and tries to place the most occurring digit in the current solution into it. It is also hypothesized that humans tend to complete a set of digits as fast as possible, so the heuristic will try to place the most occurring digit in the row (if it is already in the unit it will try the next). Once the heuristic has run through all these steps it will return a set of possible literals. Different from the original DPLL, the algorithm will iterate through these possibilities. If a backtrack is triggered the algorithm will try the next possibility instead of trying the opposite truth value for the literal.

Cell Based Tactic Heuristic DPLL Equivalently, the cell based heuristic attempts to mimic the human Sudoku strategy of scanning all possibilities for a certain cell. The heuristic selects the first empty cell in the current solution. Next it scans the corresponding row, column and box of the cell. Finally, it will output the possible literals which correspond to the possible options for the cell. The remainder of the algorithm is identical to the number base heuristic DPLL.

Flexibility Heuristic DPLL Lastly, the flexible heuristic algorithm switches between the two human strategy based heuristic DPLL algorithms to mimic the effect of cognitive flexibility of Sudoku players. A probability (p) is set and is used to update the current heuristic at every new call to the function. Note that a new heuristic can only be chosen after a new call to the function and can not result in possibilities to be skipped from either the number or cell based heuristic. As humans tend to stick to one tactic for a while, and only switch when necessary, a relatively low value for p, namely 0.2, is used in this experiment.

3 Analysis and Results

During the experiment, both the number of backtracks each algorithm needed to solve each Sudoku, as well as the number of calls to the function are recorded. Backtracks occur when the algorithm reaches a conflict (a truth value assignment that is not possible), calls to the function occur every time a new truth assignment takes place. Both are run time independent measures of the algorithms' performances.

3.1 Calls to Function

Upon visual inspection, the data does not seem normally distributed (see figure 3 of the appendix), which is confirmed by a Shapiro-Wilk test (see table 1). To test whether any of the algorithms performed significantly different from the others a Kruskal-Wallis is chosen. As the data is not normally distributed, this is more suitable than a one-way ANOVA. A first Kruskal-Wallis test on the calls to function data shows that indeed there is at least one algorithm that performs differently than the others ($H \approx 1661.14$, $p = 0.0$). Figure 1 shows that algorithm 1, which is the DPLL without heuristics, performs more calls to function than the other algorithms. A second Kruskal-Wallis test without this algorithm shows that the other algorithms do not differ from each other in calls to function performance ($H \approx 0.17$, $p \approx 0.98$).

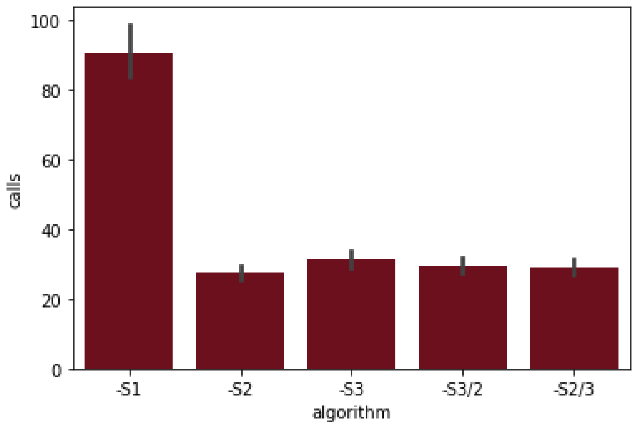

Fig. 1: Boxplot of the number of calls to the function per algorithm.The labels correspond to the algorithms as follows: S1: Original DPLL, S2: Number based heuristic DPLL, S3: Cell based heuristic DPLL, S3/2: Flexibility heuristic DPLL with a cell based start, S2/3: Flexibility heuristic DPLL with a number based start.

3.2 Backtracks

Upon visual inspection, the data does not seem normally distributed (see figure 4, which is confirmed by a Shapiro-Wilk test (see table 1).

A Kruskal-Wallis test on the backtracks data shows that none of the algorithms significantly differ in performance ($H \approx 1025.42$, $p \approx 1.11\text{e}{-220}$). Although, as figure 2 shows, algorithm 1, which is the DPLL without heuristics, performs more backtracks than the other algorithms, this difference is not significant.

4 Conclusion

The analysis shows that all the studied DPLL algorithms with heuristics outperform the DPLL algorithm without heuristics in terms of calls to function, but not in terms of backtracks. Therefore it cannot be definitively concluded that the heuristics make the algorithm more efficient.

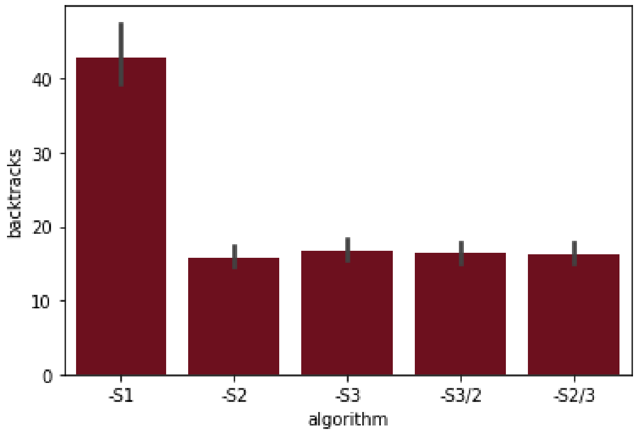

Fig. 2: Boxplot of the number of backtracks per algorithm.The labels correspond to the algorithms as follows: S1: Original DPLL, S2: Number based heuristic DPLL, S3: Cell based heuristic DPLL, S3/2: Flexibility heuristic DPLL with a cell based start, S2/3: Flexibility heuristic DPLL with a number based start.

Table 1: P values of the Shapiro-Wilk test performed on the calls to function and backtrack data of each algorithm.

Algorithm	Calls to Function	Backtracks
S1	0.304620	0.304473
S2	0.340706	0.329924
S3	0.346100	0.342527
S4	0.338832	0.323098
S5	0.333387	0.326080

Contrary to expectation, the algorithm with the flexibility heuristic does not outperform the algorithms with only the number based tactic or cell based tactic heuristic for either performance measure. Therefore we can conclude that adding the number based tactic heuristic, cell based tactic heuristic and flexibility heuristic to the DPLL algorithm could be beneficial to its performance, but that flexibility in particular does not provide extra benefit over the others.

5 Discussion

It is interesting to see that adding human reasoning inspired heuristics to the DPLL algorithm could improve its ability to go through the search space efficiently. Recent literature suggests that inherently, on average humans do not have a strong preference for a number or cell based tactic [2]. Perhaps neither tactic necessarily works better than the other, just like heuristics in this experiment. Because they perform so similarly, switching between them at random moments does not have a strong effect. Humans only switch tactics when the problem they are solving calls for it. Implementing specific situations relating to the difficulty the algorithm has with solving the Sudoku that trigger a switch might make a flexibility heuristic more effective. The switch probability used in the experiment was also relatively low, and the Sudoku's did not take many steps to solve. A higher switch probability, or Sudoku's that need more steps to solve would also exploit the flexibility heuristic more. Whether that would lead to a positive or negative difference in performance in comparison to the single tactic heuristics lies outside the scope of this paper and is open for future research.

What is clear is that these single tactic heuristics have potential. It would be useful to compare them to other commonly used heuristics like MOMS [5]. If they are effective in that context, these heuristics could prove that incorporating human strategies is useful, possibly even for other SAT problems.

References

1. Baas, M., Roskes, M., Sligte, D., Nijstad, B.A., De Dreu, C.K.W.: Personality and Creativity: The Dual Pathway to Creativity Model and a Research Agenda. Social and Personality Psychology Compass **7**(10), 732–748 (2013). https://doi.org/10.1111/spc3.12062, http://onlinelibrary.wiley.com/doi/abs/10.1111/spc3.12062, _eprint: https://compass.onlinelibrary.wiley.com/doi/pdf/10.1111/spc3.12062
2. Behrens, T., Räuker, M., Kalbfleisch, M., Jäkel, F.: Flexible use of tactics in Sudoku. Thinking & Reasoning pp. 1–43 (Jun 2022). https://doi.org/10.1080/13546783.2022.2091040, https://www.tandfonline.com/doi/full/10.1080/13546783.2022.2091040
3. Delahaye, J.P.: The Science behind SUDOKU. Scientific American **294**(6), 80–87 (2006), http://www.jstor.org/stable/26061494, publisher: Scientific American, a division of Nature America, Inc.

4. Kalia, V., Fuesting, M., Cody, M.: Perseverance in solving Sudoku: role of grit and cognitive flexibility in problem solving. Journal of Cognitive Psychology **31**(3), 370–378 (Apr 2019). https://doi.org/10.1080/20445911.2019.1604527, https://www.tandfonline.com/doi/full/10.1080/20445911.2019.1604527
5. Lagoudakis, M.G., Littman, M.L.: Learning to Select Branching Rules in the DPLL Procedure for Satisfiability. Electronic Notes in Discrete Mathematics **9**, 344–359 (Jun 2001). https://doi.org/10.1016/S1571-0653(04)00332-4, https://www.sciencedirect.com/science/article/pii/S1571065304003324
6. Louis Lee, N.Y., Goodwin, G.P., Johnson-Laird, P.N.: The psychological puzzle of Sudoku. Thinking & Reasoning **14**(4), 342–364 (Nov 2008). https://doi.org/10.1080/13546780802236308, http://www.tandfonline.com/doi/abs/10.1080/13546780802236308
7. Lucas, G.M., Gratch, J., Cheng, L., Marsella, S.: When the going gets tough: Grit predicts costly perseverance. Journal of Research in Personality **59**, 15–22 (Dec 2015). https://doi.org/10.1016/j.jrp.2015.08.004, https://www.sciencedirect.com/science/article/pii/S009265661530012X
8. Nijstad, B., Dreu, C.D.D., Rietzschel, E., Baas, M.: The dual pathway to creativity model: Creative ideation as a function of flexibility and persistence (2010). https://doi.org/10.1080/10463281003765323
9. Robinson, J.A.: Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. Communications of the ACM, vol. 5 (1962), pp. 394–397. The Journal of Symbolic Logic **32**(1), 118–118 (Jun 1967). https://doi.org/10.2307/2271269, http://www.cambridge.org/core/journals/journal-of-symbolic-logic/article/abs/martin-davis-george-logemann-and-donald-loveland-a-machine-program-for-theoremproving-communications-of-the-acm-vol-5-1962-pp-394397/EF09D5642AC20F9C7AE5C118B11285DA, publisher: Cambridge University Press
10. Schlobach, K.: Project 1 (SAT) Resources

A Appendix

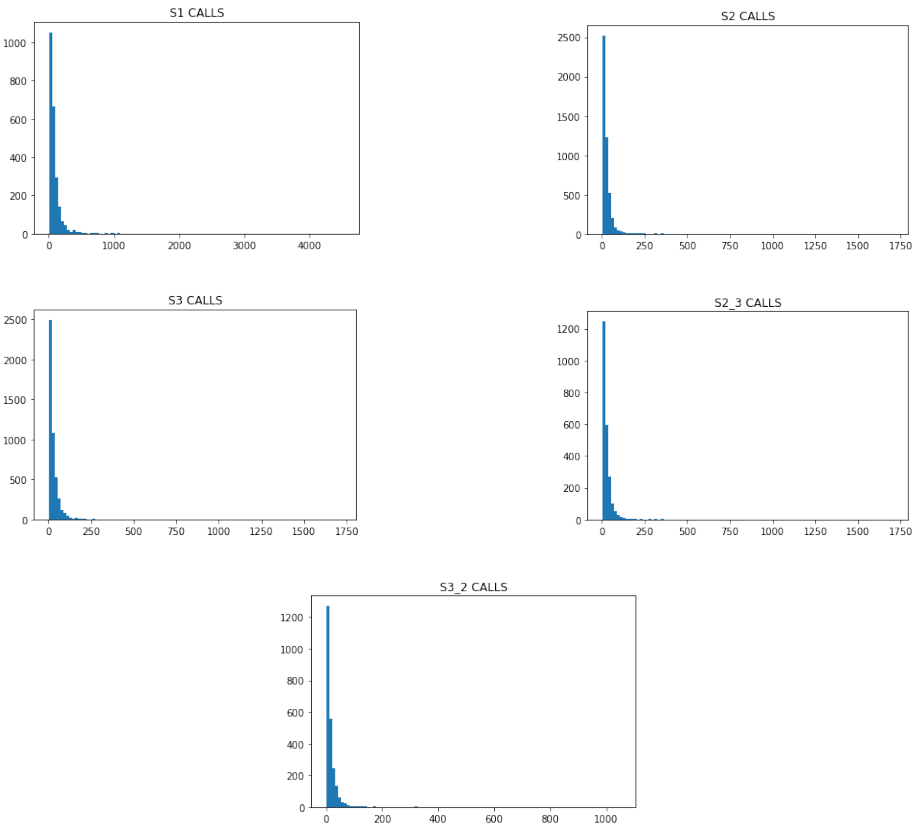

Fig. 3: Histograms showing the distribution of the calls to function data of each algorithm. The labels correspond to the algorithms as follows: S1: Original DPLL, S2: Number based heuristic DPLL, S3: Cell based heuristic DPLL, S3/2: Flexibility heuristic DPLL with a cell based start, S2/3: Flexibility heuristic DPLL with a number based start.

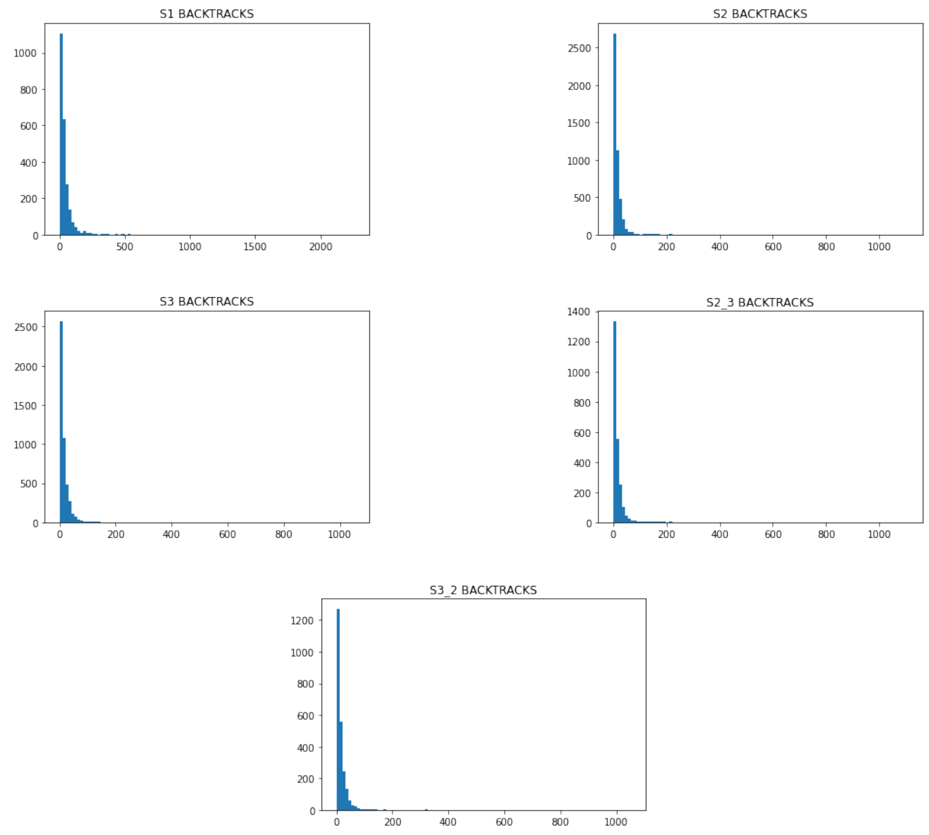

Fig. 4: Histograms showing the distribution of the backtracks data of each algorithm.The labels correspond to the algorithms as follows: S1: Original DPLL, S2: Number based heuristic DPLL, S3: Cell based heuristic DPLL, S3/2: Flexibility heuristic DPLL with a cell based start, S2/3: Flexibility heuristic DPLL with a number based start.