Instance-Based Selection of Policies for SAT Solvers[⋆]

Mladen Nikolić, Filip Marić, Predrag Janičić

Faculty of Mathematics, University of Belgrade,
Belgrade, Studentski Trg 16, Serbia
{nikolic, filip, janicic}@matf.bg.ac.rs

Abstract. Execution of most of the modern DPLL-based SAT solvers is guided by a number of heuristics. Decisions made during the search process are usually driven by some fixed heuristic policies. Despite the outstanding progress in SAT solving in recent years, there is still an appealing lack of techniques for selecting policies appropriate for solving specific input formulae. In this paper we present a methodology for instance-based selection of solver's policies that uses a data-mining classification technique. The methodology also relies on analysis of relationships between formulae, their families, and their suitable solving strategies. The evaluation results are very good, demonstrate practical usability of the methodology, and encourage further efforts in this direction.

1 Introduction

The propositional satisfiability problem (SAT) is one of the fundamental problems in computer science. It is the problem of deciding if there is a truth assignment under which a given propositional formula (in conjunctive normal form) evaluates to true. SAT was the first problem proved to be NP-complete [Coo71] and it still has a central position in the field of computational complexity. SAT problem is also very important in many practical domains such as electronic design automation, software and hardware verification, artificial intelligence, and operations research. Thanks to recent advances in propositional solving technology, *SAT solvers* (procedures that solve the SAT problem) are becoming a tool for attacking more and more practical problems.

A number of SAT solvers have been developed. The majority of state-of-the-art complete SAT solvers are based on the branch and backtrack algorithm called Davis-Putnam-Logemann-Loveland or the DPLL algorithm [DP60,DLL62]. Spectacular improvements in the performance of DPLL-based SAT solvers achieved in the last few years are due to (i) several conceptual enhancements on the original DPLL procedure, aimed at reducing the amount of explored search space (e.g., *backjumping*, *conflict-driven lemma learning*, *restarts*), (ii) better implementation techniques (e.g., *two-watched literals* scheme for unit propagation), and (iii) smart heuristic components (which we focus on in this work). These

⋆ This work was partially supported by Serbian Ministry of Science grant 144030.

advances make possible to decide the satisfiability of industrial SAT problems with tens of thousands of variables and millions of clauses.

Complex *policies*, heuristics that guide the search process, represent important parts of modern SAT solvers and are crucial for solver's efficiency. These include policies for literal selection, for determining the clause database size, for choosing restart points, etc. Specific policies are usually *parameterized* by a number of numerical and categorial parameters. Single policy with different parameter values can be treated as different policies. SAT solving process is completely determined (up to randomized choices) only when all its heuristic policies are set. Selected combinations of policies specify the *solving strategy* (or simply *strategy)*.

Typically, every SAT solver uses a predetermined, hard-coded strategy and applies it on all its input formulae. However, in recent times, SAT solvers tend to implement multiple policies and the question arises as to how to choose a strategy that would give good performance for a specific SAT instance. Addressing this question is of crucial importance because the solving time for the same input formula can vary for several orders of magnitude depending on the solving strategy used. The problem of adapting a SAT solver to the input formula has been addressed for the first time only recently. Our approach significantly differs from the only existing related approach we are aware of (as discussed in Sect. 5).

Propositional formulae can be clustered in families of formulae by their origin — industrial problems (e.g., FPGA routing), manually crafted problems (e.g., graph coloring, Hanoi towers), or random generated problems (e.g., k-SAT). It is interesting to explore the behaviour of different policies and solving strategies on families of formulae. The important question is whether one strategy shows the same or similar behaviour on similar formulae. If this is the case, and if one can automatically guess a family to which a given formula belongs, then this could be used for selecting an appropriate strategy for this particular formula. To implement this approach, one needs (i) a technique for classifying formulae based only on their syntax; (ii) information about behaviour of different policies on various families of formulae.

The main message of this work is that intelligent selecting of solving policies, based on the syntax of the input formula, can significantly improve efficiency of a SAT solver. The proposed methodology will not lead to optimal performance on each input formula, but the solving performance will be significantly improved in average on multiple input formulae. Here, by improving efficiency of a SAT solver we mean increasing the number of formulae solvable within some time limit and decreasing the solving time.

The proposed methodology relies on several hypotheses that will be investigated in the rest of the paper:

(H1) Formulae of the same family (i.e., of similar origin) share some syntactical properties that can be used for automated formula classification;

(H2) For each family of formulae there is only a small number of solving strategies that are appropriate — that show better performance on formulae belonging to that family then all other available strategies.

(H3) For formulae that are syntactically similar, the best strategies are also (in some sense) similar.

If the above hypotheses hold, then our methodology will be practically applicable. Namely, if the formula is correctly classified then it has a good chance to be solved by a solving strategy suitable for a family that the formula belongs to. However, even if the formula is misclassified, it will be solved using a strategy similar to the optimal one.

The rest of the paper is organized as follows: in Sect. 2, a brief background information on SAT problem, SAT solvers, and their heuristic components is given. In Sect. 3, the proposed methodology is described. The experimental results are given in Sect. 4. In Sect. 5 related work is discussed. In Sect. 6 final conclusions are drawn and some directions of possible further work are discussed.

2 Background

Most of today's state-of-the-art solvers are complex variations of the DPLL procedure. In the rest of the paper, we shall assume that the reader is familiar with the modern SAT solving techniques. More on these topics can be found, for example, in [NOT06,KG07,Mar08,GKSS07]. Although modern SAT solvers share common underlying algorithms and implementation techniques, their operation is guided by a number of heuristic policies that have to be selected in order to define solving strategies. The most important heuristic policies determine: (i) which literals to choose for branching, (ii) when to apply restarting, and (iii) when to forget some clauses that are learnt during the solving process. In the rest of this section, policies that were varied in our experiments will be described.

Literal selection policies. During the DPLL backtrack-search, literals used for branching should be somehow selected. This is the role of *literal selection policies*. Most literal selection policies separately select a variable v (by using a *variable selection policy*) and only then choose its polarity, i.e., choose if it should be negated or not (by using a *polarity selection policy*).

Some variable selection policies are the following[1]:

VS_{random} — This policy randomly chooses a variable among all variables of the initial formula that are not defined in the current valuation, i.e., assertion trail.

$\mathsf{VS}^{b,d,init}_{VSIDS}$ — The goal of this policy (introduced in the solver Chaff [MMZ+01]) is to select a variable that was active in recent conflicts. In order to implement this, an activity score is assigned to each variable. On every conflict, the scores of the variables that occur in the conflict clause are bumped, i.e., increased by a *bump factor* given by the parameter b. Also, during the conflict analysis process, on each resolution step all variables that occur in the explanation clause are bumped. To stimulate recent conflicts, on each conflict all the scores are decayed, i.e., decreased by a *decay factor* given by the parameter d.

[1] The policy names will be printed in subscripts and their parameters in superscripts.

An important aspect of the VSIDS variable selection policy is how to assign initial scores to variables. If the parameter *init* has the value ZERO, scores of all variables are set to zero, hence all variables have the same chance to be selected. If the parameter *init* has the value FREQ, the initial score of each variable is set to its number of occurrences in the initial formula F_0.

$\mathsf{VS}^{p}_{random} \circ \mathsf{VS}_x$ — This compound policy chooses a random variable with probability p and otherwise uses a given policy here denoted by VS_x.

Some polarity selection policies are the following:

$\mathsf{PS}_{positive}$ — Always selects a non-negated literal.

$\mathsf{PS}_{negative}$ — Always selects a negated literal.

PS^{p}_{random} — A random selection which chooses a non-negated literal with probability p.

$\mathsf{PS}^{init}_{polarity\_caching}$ — When using this policy (introduced in the solver RSAT [PD07] as *phase caching*), a preferred polarity is assigned to each variable and it is used for polarity selection. Whenever a literal is asserted to the current assertion trail (either as a decision or as a propagated literal), its polarity defines the future preferred polarity of its variable. When a literal is removed from the trail (during backjumping or restarting) its preferred polarity is not changed. If the parameter *init* has the value POS, then initial polarities of all variables are positive, it has the value NEG then they are set to negative, and if it has the value FREQ, then preferred polarity of each variable is set to the polarity which is more frequent of the two in the initial formula.

Restart policies. Restart policies determine when to apply restarting. Most restart policies are based on conflict counting. On each conflict, the counter is increased. Restarting is applied whenever the counter reaches a certain threshold value. When this happens, the counter is reset and a new threshold is selected according to some specific policy. Some possible restart policies are the following:

$\mathsf{R}_{no\_restart}$ — Restarting is not applied.

$\mathsf{R}^{c_0,q}_{minisat}$ — The initial threshold value is set to c_0 and the threshold values form a geometric sequence with a quotient q [ES04].

R^{m}_{luby} — The threshold values are elements of the Luby series [LSZ93] multiplied by a positive integer m, while the Luby series is:

$$t_i \; = \; \begin{cases} 2^{k-1} & \text{if } i = 2^k - 1 \\ t_{i-2^{k-1}+1} & \text{if } 2^{k-1} \leq i \leq 2^k - 1 \end{cases}$$

Its first few elements for $m = 1$ are $1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, ...$

$\mathsf{R}^{c_0,q}_{picosat}$ — In this policy (introduced by the solver PicoSAT [Bie08]), restarts are controlled by two geometric sequences of threshold values — inner and outer, both with an initial member c_0 and a quotient q. Restarting is applied when the number of the conflicts reaches the current inner threshold value, and then the inner threshold value advances to the next element of the sequence. When the inner threshold exceeds the current outer threshold value, it is reset to the initial value c_0, and the outer threshold advances.

3 Methodology

Our methodology, when applied to a given SAT solver, selects an appropriate solving strategy (i.e., combination of policies) for each input formula. The proposed methodology can be applied to any DPLL-based SAT solver, provided it supports multiple policies. In our experiments, the ArgoSAT solver[2] was used since it implements a large number of policies and since its modular architecture allows easy modification of existing and implementation of new policies [Mar09].

The overall methodology consists of two phases.

Training phase. This phase consists of systematic solving of all formulae from a representative corpus, by using all candidate solving strategies. This allows selecting the best solving strategy (the one that solves the most formulae) for each family of formulae from the corpus. *Profiles* of all formulae from the corpus (i.e., their representation suitable for classification) are also computed in this phase.

Exploitation phase. In this phase, the family of a given formula is guessed and the strategy that showed the best results on that family during the training phase is used for its solving.[3] After the training, the system implementing the methodology can be applied both to the formulae from the training corpus and to some other formulae.

Several issues still need to be addressed, as discussed below.

The choice of candidate solving strategies. In our case candidate strategies are defined as all ($60 = 3 \cdot 5 \cdot 4$) possible combinations of the given policies. They are listed in Table 1. Apart from the policies that are subject to automatic selection, some important policies (e.g., forget and conflict analysis) are fixed. [4] We do not claim that some other policies could not give better performance. Some of the considered policies are even expected to be inferior. However, we are proposing a general methodology that can be used with any input set of policies.

The choice of corpus of formulae for training and evaluation. In our experiments (both during the training and for testing), the corpus from the *SAT competition* in 2002 was used. It consists of a large number of families of formulae, with many families containing formulae of various difficulty. The total number of formulae in this corpus is 1964, and we clustered them into 39 families, mostly just by following the directory structure. Since this corpus was systematically solved in the training phase, it can be used both for

[2] `http://argo.matf.bg.ac.rs/software/`

[3] We also tested an alternative approach that does not use information on families. In that approach, k ($k \geq 1$) formulae that are most similar to the input formula are detected. Then, a solving strategy that occurs most frequently among l ($l \geq 1$) best strategies for each detected formula is chosen. This alternative approach will not be discussed in more details because it gave inferior results.

[4] A MiniSAT-style forget policy [ES04] and 1UIP technique [ZMMM01] for conflict analysis are used.

Variable selection	VS_{random}, $\mathsf{VS}_{VSIDS}^{1.0,\,1.0/0.95,\,\text{FREQ}}$, $\mathsf{VS}_{random}^{0.05} \circ \mathsf{VS}_{VSIDS}^{1.0,\,1.0/0.95,\,\text{FREQ}}$
Polarity selection	PS_{POS}, PS_{NEG}, $\mathsf{PS}_{random}^{0.5}$, $\mathsf{PS}_{polarity\_caching}^{\text{NEG}}$, $\mathsf{PS}_{polarity\_caching}^{\text{FREQ}}$
Restart policies	$\mathsf{R}_{no\_restart}$, $\mathsf{R}_{minisat}^{100,1.5}$, R_{luby}^{512}, $\mathsf{R}_{picosat}^{100,1.5}$

Table 1. Overview of policies used in our experiments

testing of hypotheses and for thorough analysis of the proposed methodology. For testing of a generalization power of our methodology on a different corpus, the *SAT competition* corpus from 2007 was used. Namely, out of 906 formulae in this corpus, only 12 of them also belong to the corpus from 2002. In addition, the two corpora include significantly different families (although overlapping exists). Since the SAT2007 corpus was used only for evaluation of our resulting solving system, we do not consider its partitioning into families.

The choice of relevant features of propositional formulae. In order to measure the syntactic similarity of propositional formulae (which is necessary for classification), the formulae were represented by using the first 33 features used in [XHHLB08]. These are features that can be calculated in short time. They include the number of clauses c and variables v in the input formula, their ratio $\frac{c}{v}$, fraction of binary, ternary, and Horn clauses, node degree statistics for variable nodes in variable-clause graph like mean, variation coefficient, minimum, maximum, and entropy, etc. The vectors of these features are called the *formula profiles* or simply *profiles.*

The choice of methods for classification of propositional formulae. For classification, the *k-nearest neighbour* algorithm was used. When given an unknown instance, this algorithm selects the class that contains the most of the k instances from the training corpus that are closest to the given one. A number of distance functions between the profiles were used [TJK06]. For instance:

$$d_1(\mathcal{P}', \mathcal{P}'') = \sqrt{\sum_i (\mathcal{P}'_i - \mathcal{P}''_i)^2} \quad d_2(\mathcal{P}', \mathcal{P}'') = \sum_i \left(\frac{\mathcal{P}'_i - \mathcal{P}''_i}{\sqrt{|\mathcal{P}'_i \mathcal{P}''_i)|} + 1} \right)^2$$

$$d_3(\mathcal{P}', \mathcal{P}'') = \sum_i \frac{|\mathcal{P}'_i - \mathcal{P}''_i|}{\sqrt{|\mathcal{P}'_i \mathcal{P}''_i|} + 1} \quad d_4(\mathcal{P}', \mathcal{P}'') = \sum_i \left(\frac{\mathcal{P}'_i - \mathcal{P}''_i}{\sqrt{|\mathcal{P}'_i \mathcal{P}''_i|} + 10} \right)^2$$

where $\mathcal{P}'$ and $\mathcal{P}''$ are instance profiles.

4 Experiments and Evaluation

Experiments described in this section test the hypotheses that our approach relies on (given in Sect. 1), and also demonstrate a good overall quality of our methodology.

Training Phase. During the training phase, the cutoff time for solving one formula by one strategy was set to 600s. It would be interesting to consider higher cutoff times as well, but this choice was made with regard to available computational resources. Since shuffling of clauses and variables of a formula can lead to big differences in its solving time (up to an order of magnitude), the solving times associated with the formulae were calculated in the following way. For each formula, the original and one shuffled variant were solved. If both variants of the formula were solved within the time limit, the arithmetic mean of their solving times was associated to the formula. If either variant was not solved within the cutoff time limit, the formula was considered to be unsolved. The SAT solver was used on all the formulae from the extended corpus, for all 60 strategies. The total number of calls to the SAT solver was 235680 ($= 1964 \cdot 60 \cdot 2$). The experiments were conducted on an IBM Cluster 1350 cluster computer with 32 processors. The total processor time used was around 1010 days.

Along with solving the formulae, their profiles were computed. The average profile computation time was 0.39s per formula.

4.1 Testing Hypotheses

Hypothesis (H1). The first hypothesis is that formulae from the same family share syntactical properties that can be used for automated formula classification. In the k-nearest neighbours method, values $1, 3, 5, 7$ were used for k. The best results were obtained for $k = 1$ with the distance function d_3[5]. The precision, a ratio between the number of correctly classified formulae and the total number of formulae classified, was 98.5%. The arithmetic mean of precisions for individual families[6], was 89.4%. To avoid evaluating on the same data that was used for training, both statistics were estimated using the *leave-one-out procedure.* This procedure consists of removing formula from the corpus, computing the relevant statistic on the rest of the corpus, and returning the formula into the corpus. This is done for all formulae. The obtained values of the statistic were averaged at the end to give the final estimate of the statistic on given data.

The results of the classification are outstanding (especially keeping in mind a rather large number of classes — 39) and show that the first hypothesis of the methodology is sound. Since the average profile computation time for a formula is 0.39s and the classification time of a known profile is less than 0.01s, this approach to classification is practically usable for our purposes and may have applications in other domains too.

Hypothesis (H2). The second hypothesis of our methodology is that there is a small number of strategies for each family of formulae that show better performance on formulae belonging to that family then all other available strategies.

[5] Therefore, in all experimental results in the rest of the paper, this distance function will be assumed.

[6] Precision alone is not reliable in cases when some families are much larger than the others (which is the case with the SAT2002 corpus), so a high precision on large classes can hide a low precision on small classes.

To check this hypothesis, for each strategy and for each family of formulae a percentage of formulae for which that strategy was better then any other strategy was calculated. The results are shown in a graphical form in Fig. 1. In the left part of the figure, darker shades correspond to higher values, and lighter to lower values. The highest value (i.e., percentage) in the table is 30, and the lowest value is 0. In the right part of the figure, normalized entropies for families are shown. For simplicity, the results are shown only for families with at least 10 formulae that were solved by at least one strategy. The figure shows that there is no family with a dominantly best strategy. However, the presented matrix is sparse and the average normalized entropy for all families is 0.39 — hence, for each family there is a rather small set of good strategies and therefore, the second hypothesis can be considered to be justified.

These results also reveal the quality of some strategies. For instance, 15 empty columns correspond to strategies with VS_{random} variable selection policy, which suggests a poor performance of this policy.

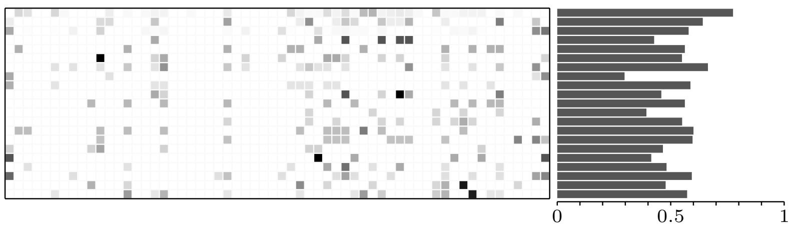

Fig. 1. The left part shows percentages of formulae from a family for which a strategy is better than any other (columns correspond to strategies and rows correspond to families). The right part shows normalized entropy values for families.

Hypothesis (H3). The third hypothesis of our methodology is that the best solving strategies for syntactically similar formulae are also similar. Syntactical similarity between formulae is already defined by the choice of profiles and the distance function (d_3) from Sect. 3. On the other hand, similarity between strategies can be defined using the *edit distance* over strategies:

$$d_c(s_1s_2s_3, t_1t_2t_3) = \sum_{i=1}^{3} c(s_i, t_i)$$

where $s_1s_2s_3$ and $t_1t_2t_3$ are triples of policies that determine strategies and $c(s_i, t_i)$ are non-negative numerical costs of switching from policy s_i to policy t_i.

To analyze the correlation between similarity of formulae and similarity of their corresponding best strategies, for each two formulae f_1 and f_2 from the corpus, with best strategies c_1 and c_2 respectively, values $\log d_3(f_1, f_2)$ and $d_c(c_1, c_2)$ were calculated. Then, the Pearson correlation coefficient between these sets of values was calculated. The costs in the function d_c were manually tuned to achieve a maximal correlation coefficient. This procedure was legitimate and it is closely related to the following question: for which group of policies (restart, variable selection, literal selection) optimal choices differ the least for syntactically similar formulae? Only formulae solved in more than 2s were included in the calculation. [7]

The calculated correlation coefficient was 0.51 with the p-value less then 0.001. This can be considered a moderate, but important correlation, keeping in mind the small number of policies that were used in the experiments and the inherent instability of the SAT solving process. Hence, we can consider the third hypothesis to be justified.

Magnitudes of costs in the function d_c, suggest that for syntactically similar formulae, the optimal restarting policy varies the least (thus being in some sense the strongest common characteristic of syntactically similar formulae) and the polarity selection policy varies the most.

4.2 Exploitation Phase

Evaluation of Strategy Selection Method on the SAT2002 Corpus. For evaluation purposes the proposed strategy selection method was compared to (i) the *"best-fixed"* strategy selection method which always uses the best fixed candidate strategy[8], and to (ii) the *"oracle"* (idealized) strategy selection method which uses the best candidate strategy for each specific formula. The first one is a fair choice for the lower bound of required performance and the second one represents the upper bound for performance because it gives the best possible performance over the set of candidate strategies on the SAT2002 corpus.

As the main measures of the overall quality of a strategy selection method the total number of formulae that it solved and its median solving time were used. There are strong reasons to base the evaluation on median instead on mean and total solving time. First, one cannot account for the censored data — solving times over the cutoff limit. If one chooses not to include these data in the calculation then the mean and total solving time show preference for solvers that solve less formulae because even if a hard formula is solved its solving time is typically near the cutoff limit, and thus raises the mean and the total solving time. On the other hand, if at least half of the formulae were solved, the median time can be calculated. Also, median time is known to be less sensitive to outliers.

To estimate the performance on the SAT2002 corpus, similarly as in the leave-one-out procedure, the solving time of each formula was computed for the strategy selected based on the results of the training phase, but with that formula excluded from the corpus. Table 2 shows the results for the two referent methods and for the one proposed. Important results were achieved, especially keeping in mind a rapid growth of the sorted solving times of formulae for all three methods, as shown in Fig. 2. The differences in the numbers of the solved formulae for all three methods seem small, but would be much higher if numerous easy formulae were not taken into account.

[7] Trivial formulae cannot discriminate between good and bad strategies. Also, variation of their solving time due to processor context switching is larger relative to their solving time.

[8] In our case the best strategy was $(\mathsf{VS}_{VSIDS}^{1.0,\,1.0/0.95,\,\text{FREQ}}, \mathsf{PS}_{polarity\_caching}^{\text{NEG}}, \mathsf{R}_{minisat}^{100,1.5})$.

Method	Number of solved formulae	Median solving time
"Best fixed"	1073	207.4s
Proposed	1135	92.6s
"Oracle"	1187	45.8s

Table 2. Comparison of strategy selection methods on SAT2002 corpus.

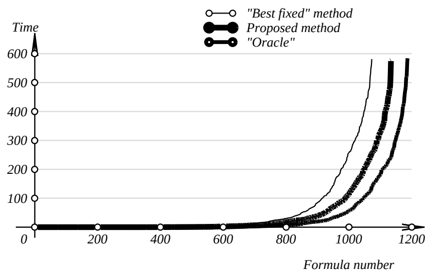

Fig. 2. Sorted solving times for different ways of choosing the strategy. Ordinal numbers in the sequence of the sorted solving times of the formulae are shown on the X axis. Solving times are shown on the Y axis.

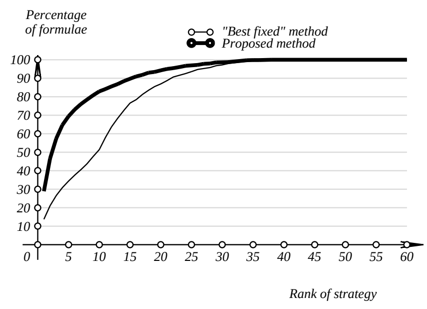

Fig. 3. Cumulative distribution function of ranks of chosen strategies for the "best fixed" method and the proposed method. Ranks of strategies are shown on the X axis, while percentage of formulae for which strategies of that or smaller rank are chosen is shown on the Y axis.

Given a strategy selection method, it is important to consider how often it chooses best, good or bad strategies (especially when compared to some other selection methods). For each formula, candidate strategies can be sorted in ascending order according to their corresponding solving times. Each strategy can be given a rank according to its position in such sequence. A strategy with the rank 1 is the most desirable for that specific formula. In Fig. 3 we present the cumulative distribution function of the ranks of chosen strategies for the proposed and for the "best fixed" method. The figure shows that the proposed method chooses good strategies much more often than the "best fixed" method.

The histogram in Fig. 4 shows the number of formulae solved by the proposed method in some percentage of a referent time, up to 200%. As a referent time we use the solving time obtained by the "best fixed" method. Only formulae that were solved either way were considered. 48 formulae are solved in more than 200% of the referent time. There are 74 formulae that were solved by the proposed method but not by the "best fixed" method, but only 12 that were solved by the "best fixed" method and not by the proposed method. It can be observed that much more formulae were solved faster than slower, compared to the "best fixed" method.

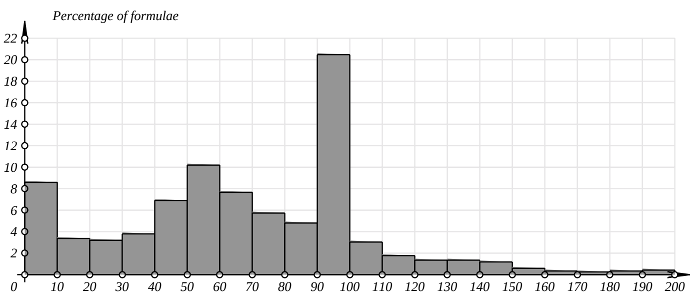

Fig. 4. Histogram of the number of formulae solved by the proposed method in some percentage of the referent time. Percentage of the referent time is shown on the X axes, while the number of formulae solved in that percentage of the referent time is shown on the Y axis.

These results show that the main thesis of this work — that intelligent choosing of the solver's strategy based on the syntax of the input formula, can significantly improve efficiency of a SAT solver — is true.

Evaluation of ArgoSmArT *system on SAT2007 corpus.* Based on the methodology described above, we implemented a SAT solving system ArgoSmArT (on top of the ArgoSAT solver).

For an additional evaluation of the proposed methodology, we used the SAT2007 corpus and showed that performance improvement achieved on one corpus is present on a different corpus too.

The formulae from the SAT2007 corpus are much harder then the ones from the SAT2002 corpus and the median time cannot be calculated, since in cutoff time of 600s less than a half of the formulae can be solved. Thus, we present 20-th percentile of the solving times[9]. The results of the comparison of ArgoSmArT to its base solver ArgoSAT are shown in Table 3 (ArgoSAT used the best fixed strategy detected in the experiments on the SAT2002 corpus).

Notice that the performance improvement achieved on the SAT2002 corpus in the number of solved formulae is also present on the SAT2007 corpus. The improvement in the solving time is also significant. As said above, these two corpora share only 12 formulae, and only one of them was solved by the ArgoSmArT within 600s, so the improvement cannot be attributed to overlapping of the training and the test set.

[9] 20-th percentile of the solving times is the value that splits the sorted solving times in two parts, the lower one having 20% of the total number of values.

System	No. of solved formulae	20-th percentile of solving time
ArgoSAT	219	311.6s
ArgoSmArT	239	249.5s

Table 3. Results of comparison between ArgoSAT and ArgoSmArT on the SAT2007 corpus.

5 Related Work

Hypotheses like H2 and H3 are already discussed as a basis for instance-based solving of the algorithm selection problem [SM08]. Algorithm selection for constraint satisfaction problems (CSP) based on performance prediction is described in [LL98]. A reinforcement learning based approach to choose variable selection policy for CSP, with only preliminary results is described in [XSS09]. In quantified boolean formulae (QBF) solving, multinomial logistic regression was successfully used for dynamic, online selection of variable selection policies[SM07]. Strategy selection for MiniSAT based on neural networks that gave limited results is described in [Kib07].

Features used for classification in this paper are first described in [NBH^{+}04] for prediction of a solver's running time and were later used in SATzilla system [XHHLB08]. SATzilla is the system that uses linear regression predictions of solver running times to select one of its component solvers for solving an input formula. As reported in [XHHLB08] for the corpus SAT2007, evaluation on the random category of SAT instances, demonstrated a significant improvement in running time. Average running time for SATzilla system was around 90s, while its best component solver average was 290s. On the crafted category, SATzilla average was around 150s, compared to its best component solver average of 280s. For the industrial category, this improvement was smaller, but still significant — 260s compared to 330s. In 27% of cases SATzilla chooses its best component solver [XHHLB07].

While both SATzilla and ArgoSmArT adjust the solving process to the input formula, there are important differences between these two approaches. First, SATzilla is the system that chooses among its component solvers (7 solvers were used at the *SAT competition* 2007), and these solvers are used as they are. On the other hand, our approach aims at boosting performance of just a single, arbitrary, base solver by selecting strategies appropriate for an input formula (in the current setup it chooses between 21 strategies that happen to be the best for some of the families from the training corpus). Therefore these two approaches can be considered complementary. The advantage of the SATzilla approach, compared to the ArgoSmArT approach, is that it offers an estimate of the running time. On the other hand, an important advantage of ArgoSmArT is that it can detect a family that the input formula belongs to.

Stochastic optimization of SAT solver parameters is described in [HBHH07]. It could be used for finding better strategies within our approach.

6 Conclusions and Future Work

We proposed a methodology for instance-based selection of solving strategies that can be applied to an arbitrary SAT solver which supports multiple solving strategies. We showed that the family a formula belongs to can be automatically recognized and that precision achieved was excellent. Also, we demonstrated that for each family of formulae, among many possible strategies, just a small number of strategies is appropriate for its solving. Along with significant correlation between syntactical similarities of formulae and similarities of strategies most appropriate for their solving, these conclusions form a firm basis for our strategy selection methodology.

The methodology was evaluated on two representative corpora. As for the overall performance, on the SAT2002 corpus a greater number of formulae was solved and the median time dropped more than 50%. The performance improvement was also demonstrated on a corpus different from the one the system was trained on. Overall, the results obtained are very good and show that the methodology is practically applicable and that further research in this, still new field, is feasible. We are planning to work on additional statistical analyzes of gathered data in order to gain a deeper insight into the nature of our best strategies and relationships between their component policies. We plan to further improve our system by using the stochastic parameter optimization, which would significantly decrease duration of the training phase. Also, we will try to combine our approach with the SATzilla approach by training SATzilla to choose between different strategies of a solver. While inspecting our data we came across the incompatibility of the MiniSAT forgetting strategy we used with the fast restarting strategies when forgetting quickly ceases. Impacts of these incompatibilities should be investigated and potentially better results achieved by changing the forget strategy. Also, a deeper analysis of behaviour of best strategies for k-SAT instances is planned.

Acknowledgements. We thank Mathematical Institute of Serbian Academy of Sciences and Arts for providing us access to their cluster computer.

References

[Bie08] Armin Biere. PicoSAT Essentials. *Journal on Satisfiability, Boolean Modeling, and Computation*, 2008.

[Coo71] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *STOC '71: Proceedings of the Third Annual ACM Symposium on Theory of Computing*, ACM, 1971.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A Machine Program for Theorem-Proving. *Commun. ACM*, 1962.

[DP60] Martin Davis and Hilary Putnam. A Computing Procedure for Quantification Theory. *J. ACM*, 1960.

[ES04] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, 2004.

[GKSS07] C P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. Satisfiability Solvers. In *Handbook of Knowledge Representation*. Elsevier, 2007.

[HBHH07] Frank Hutter, Domagoj Babic, Holger H. Hoos, and Alan J. Hu. Boosting Verification by Automatic Tuning of Decision Procedures. In *FMCAD '07: Proceedings of the Formal Methods in Computer Aided Design*, 2007. IEEE Computer Society.

[KG07] S. Krstić and A. Goel. Architecting Solvers for SAT Modulo Theories: Nelson-Oppen with DPLL. In *FroCoS '07: LNCS 4720*, 2007.

[Kib07] Raihan H. Kibria. Evolving a Neural Net-Based Decision and Search Heuristic for DPLL SAT Solvers. In *IJCNN*, 2007.

[LL98] Lionel Lobjois and Michel Lemaitre. Branch and Bound Algorithm Selection by Performance Prediction. In *In AAAI*, AAAI, 1998.

[LSZ93] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal Speedup of Las Vegas algorithms. *Information Processing Letters*, 1993.

[Mar08] F. Marić. Formalization and Implementation of SAT Solvers. *Journal of Automated Reasoning, submitted*, 2008.

[Mar09] F. Marić. Flexible Implementation of SAT solvers. In *SAT2009, submitted*, 2009.

[MMZ$^+$01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

[NBH$^+$04] Eugene Nudelman, Kevin L. Brown, Holger H. Hoos, Alex Devkar, and Yoav Shoham. Understanding Random SAT: Beyond the Clauses-to-Variables Ratio. In *Principles and Practice of Constraint Programming - CP 2004*, 2004.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *J. of the ACM*, 2006.

[PD07] K. Pipatsrisawat and A. Darwiche. A Lightweight Component Caching Scheme for Satisfiability Solvers. In *SAT '07: LNCS 4501*, 2007.

[SM07] Horst Samulowitz and Roland Memisevic. Learning to Solve QBF. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, 2007. AAAI Press.

[SM08] Kate Smith-Miles. Cross-Disciplinary Perspectives on Meta-Learning for Algorithm Selection. *ACM Comput. Surv.*, 2008.

[TJK06] Andrija Tomovic, Predrag Janicic, and Vlado Keselj. n-Gram-Based Classification and Unsupervised Hierarchical Clustering of Genome Sequences. *Computer Methods and Programs in Biomedicine*, 2006.

[XHHLB07] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. The Design and Analysis of an Algorithm Portfolio for SAT. *Principles and Practice of Constraint Programming*, 2007.

[XHHLB08] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based Algorithm Selection for SAT. *Journal of Artificial Intelligence Research*, 2008.

[XSS09] Yehua Xu, David Stern, and Horst Samulowitz. Learning Adaptation to Solve Constraint Satisfaction Problems. In *LION 3*, 2009.

[ZMMM01] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. In *International Conference on Computer Aided Design (ICCAD)*, 2001.