Generating Diverse Solutions in SAT

Alexander Nadel

Intel Corporation, P.O. Box 1659, Haifa 31015 Israel
alexander.nadel@intel.com

Abstract. This paper considers the Diverse$k$Set problem in SAT, that is, the problem of efficiently generating a number of diverse solutions (satisfying assignments) given a propositional formula. We provide an extensive analysis of existing algorithms for this problem in a newly developed framework and propose new algorithms. While existing algorithms adapt modern SAT solvers to solve Diverse$k$Set by changing their polarity selection heuristic, our new algorithms adapt the variable ordering strategy as well. Our experimental results demonstrate that the proposed algorithms improve the diversification quality of the solutions on large industrial instances of Diverse$k$Set arising in SAT-based semiformal verification of hardware.

1 Introduction

SAT solving is a core reasoning engine in a variety of applications [1]. The basic functionality of a SAT solver consists of solving the following decision problem: given a propositional formula in Conjunctive Normal Form (CNF), decide whether it has a satisfying assignment (also called a *model* or *solution*). However, major industrial applications require additional abilities from the solver. This paper considers the Diverse$k$Set problem in SAT: given a satisfiable propositional formula in CNF, return a user-given number of solutions that are as diverse as possible.

In [2] we proposed a number of algorithms for solving Diverse$k$Set in SAT in the context of the SAT-based semiformal hardware verification flow, where the Diverse$k$Set solver is the core reasoning engine. The flow has practical importance, since it is able to find bugs in complex industrial designs that are missed by both Bounded Model Checking (BMC) and simulation [2]. The main idea of [2] is that, given a complex property that cannot be verified by BMC, since BMC cannot reach a sufficient bound, one can advance towards the property along multiple paths via user-given waypoints. The paths must be as diverse as possible in order not to miss bugs. A Diverse$k$Set solver is used to extract such paths. Diversification quality is defined in [2] as the normalized sum of the Hamming distances between each pair of solutions. The Diverse$k$Set algorithms proposed in [2] adapt a modern conflict-driven clause-learning (CDCL) SAT solver, invoking it only once to generate all the models. The algorithms are *polarity-based* in the sense that diversification is achieved solely by changing the polarity selection heuristic, where the polarity is the value assigned to each new decision variable. We proposed *randomized* and *guided* polarity-based approaches. Randomized approaches select the polarity randomly on all or some of the occasions, while guided approaches select the polarity so as to explicitly guide the solver away from previous solutions.

The first contribution of this paper is the development of a convenient framework for analyzing DIVERSEkSET algorithms and the analysis of existing algorithms using this framework. (The analysis in [2] is very brief, since [2] is mostly dedicated to a particular application of DIVERSEkSET.) In particular, our framework allows one to measure diversification quality online (i.e. while the solver is running, as opposed to offline, after the solver has finished) and to estimate the contribution of each variable to diversification quality.

We analyze the empirical behavior of DIVERSEkSET algorithms on 66 CNF instances generated by Intel's semiformal verification flow for generating 2 to 100 models. This is in contrast to [2], which reports about experiments with 10 models only, since this number was used by the semiformal application. Our experimental setup provides us the valuable ability to analyze the behavior of the algorithms as a function of the number of models. Our analysis can also be empirically helpful for semiformal verification, since the number of required models is expected to grow as more computational resources become available. In order to improve readability, we present and discuss relevant experiments immediately after describing a certain family of algorithms instead of concentrating all the experimental results in one section. The instances we used have 213,047 variables and 738,862 clauses on average, while the largest instance has 910,868 variables and 3,251,382 clauses. All the instances are available from the author. All the algorithms were implemented in Intel's CDCL SAT solver Eureka and run on Intel® Xeon® machines with 4Ghz CPU frequency and 32Gb of memory.

The second contribution of this paper is the introduction and analysis of new algorithms for DIVERSEkSET. Our new algorithms are *variable-based*; that is, they change the variable ordering in addition to the polarity selection heuristic. We propose guided and randomized variable-based methods, which can be local or global with respect to the default decision heuristic. Our algorithms improve diversification quality. We observe a trade-off between diversification quality and run-time. Moderate improvement of diversification quality can be achieved with negligible run-time cost, while more significant improvement in quality requires additional run-time. We show how one can control the trade-off between quality and run-time.

The rest of the paper is organized as follows. Section 2 reviews some related work and provides the necessary background. Definitions are provided in Section 3. Existing algorithms are analyzed in Section 4. Section 5 is dedicated to the new variable-based algorithms. The conclusion and directions for future work appear in Section 6.

2 Related Work and Background

As far as we know, the only work that considers the DIVERSEkSET problem in SAT is our previous work [2]. However, the problem of finding a user-given number of diverse solutions has been studied in the Constraint Satisfaction Problem (CSP) domain (e.g., [3, 4]). A guided value-based method for solving DIVERSEkSET for CSP is proposed in [3]. A randomized value-based method is also known in the CSP community [4] (we did not find a paper introducing it). A number of efficient value-based methods are proposed in [4] in the context of automatic generation of architectural tests.

A number of works (e.g., [5–7]) are dedicated to the related problem of generating a (nearly) uniformly distributed sampling of the solution space in various domains, including SAT [7]. We denote by kSAMPLING the problem of generating k out of N solutions, where each solution should be selected with the probability as close as possible to $1/N$. It is important to realize the difference between DIVERSEkSET and kSAMPLING (explained in the context of CSP in [4]). Consider the problem of finding two diverse/uniformly distributed solutions given a tautological formula. Consider an algorithm which returns the following two models: (1) All the variables are assigned 1; (2) All the variables are assigned 0. This algorithm returns the optimal solution for DIVERSEkSET for a tautological formula. However, it is unsatisfactory for the sake of kSAMPLING, since the solutions are predefined. Still, since a set of solutions for kSAMPLING can be used as an approximation for a set of solutions for DIVERSEkSET, one can evaluate the performance of existing algorithms for kSAMPLING on DIVERSEkSET instances.

The DIVERSEkSET algorithms presented in this paper are built on top of a modern CDCL SAT solver. Modern SAT solvers are extremely efficient on huge industrial instances. Among the key features that enable the solvers to be so efficient, despite the apparent difficulty of solving huge instances of NP-complete problems, are *dynamic behavior* and *search locality*, that is, the ability to maintain the set of assigned variables and recorded clauses relevant to the currently explored space. This effect is achieved through various techniques, such as 1UIP conflict clause learning [8], non-chronological backtracking [9], rapid restarts [10], variable decision heuristics (also known as variable ordering heuristics) and polarity selection heuristics.

The variable decision heuristics of a modern SAT solver are dynamic [8]. Their goal is to improve the locality of the search by picking variables that participated in recent conflict analysis. One can distinguish between variable-based decision heuristics and clause-based decision heuristics (mixed variable- and clause-based heuristics are also in use). Variable-based heuristics are based on VSIDS [8]. VSIDS maintains a score for each literal. The score is increased for a variable that participates in conflict analysis. Once in a while the scores are decreased. Consider now the clause-based heuristic CBH [11]. CBH maintains a clause list containing both the initial and the conflict clauses. Whenever a new conflict clause is derived, CBH moves the clauses that participated in conflict analysis, along with the new conflict clause, to the top of the list. The next decision variable is picked from the topmost non-satisfied clause using the variable with the greatest VSIDS score. CBH tends to pick interrelated variables, a fact which makes the search more local.

Most modern SAT solvers (including Eureka, which we use for our experiments) employ the phase-saving heuristic [12–14] as their polarity selection heuristic. The phase-saving heuristic for variable v always chooses the last value v was assigned. This strategy tries to refocus the search on subspaces that the solver has knowledge about.

3 Definitions

We start with defining some auxiliary notions. Given two boolean values σ_1 and σ_2, a pair $\{\sigma_1, \sigma_2\}$ is *different* if $\sigma_1 \neq \sigma_2$. Assume we are given a propositional formula in CNF F with q variables V and r satisfying assignments $M = \{\mu_1, \ldots, \mu_r\}$ (also known as *models* or *solutions*) for F. We define $\mu_m^u \in \{0, 1\}$, where $u \in V$ and $1 \leq m \leq r$, to be a value assigned to the variable u in μ_m.

The Hamming distance between two models μ_i and μ_j is defined to be the number of different pairs amongst $\left\{\mu_i^u, \mu_j^u\right\}$ for $u \in V$. Diversification quality is defined in [2] as the sum of the Hamming distances between each pair of models, normalized to the range $[0 \ldots 1]$ by dividing by $q\binom{r}{2}$.

We use the same measure for diversification quality but calculate it differently, keeping in mind two goals. First, we want to be able to estimate the contribution of each variable to quality. Second, we want to be able to measure quality online as well as offline. An offline version of our definitions is presented next. Afterwards we show how to generalize our definitions so that they can be used online as well.

Let the *variable (diversification) quality* S_m^u, given a variable u and m models, be the number of different pairs amongst the pairs of values assigned to u (namely, $\left\{\mu_i^u, \mu_j^u\right\}$, where $1 \leq i, j \leq r$ and $i < j$). Note that the variable quality S_{m+1}^u for $m \geq 1$ models is the variable quality for m models plus the number of different pairs amongst $\left\{\mu_{m+1}^u, \mu_i^u\right\}$ for $1 \leq i \leq m$. Let p_m^u and n_m^u be the number of times u was assigned 1 and 0, respectively, in m models. We have the following recursive definition for variable quality for $m \geq 1$:

$$S_1^u = 0; S_{m+1}^u = \begin{cases} S_m^u + n_m^u \text{ if } \mu_{m+1}^u = 1; \\ S_m^u + p_m^u \text{ if } \mu_{m+1}^u = 0. \end{cases}$$

We provide an alternative definition for variable quality, which is sometimes more useful. It is not hard to see that $p_m^u \times n_m^u$ is exactly the number of different pairs amongst $\left\{\mu_i^u, \mu_j^u\right\}$. Hence, we have:

$$S_m^u = p_m^u \times n_m^u$$

Now we can define the *(diversification) quality* Q_m for $1 \leq m \leq r$ models as the sum of all the variable qualities, normalized to the range $[0 \ldots 1]$:

$$Q_m = \frac{\sum_{u \in V} S_m^u}{\binom{m}{2} q}$$

We provide another useful notion of a potential of a variable. Given a variable $u \in V$ and $m \geq 1$ models, $\Pi^u_m = p^u_m - n^u_m$ is the *potential* of u. The potential of a variable is the difference between the number of 1's and 0's assigned to u in all the models. The *absolute potential* of u is the absolute value of the potential $|\Pi^u_m|$. We will see later that the potential and the quality of a variable are closely connected. Fig. 1 provides a simple example of applying our definitions.

	μ_1	μ_2	μ_3
v	0	0	0
u	1	1	0

Fig. 1: An example of applying our definitions, given two variables and three models. We have: $p^v_3 = 0$; $n^v_3 = 3$; $p^u_3 = 2$; $n^u_3 = 1$. The variable qualities are: $S^v_3 = p^v_3 \times n^v_3 = 0$; $S^u_3 = p^u_3 \times n^u_3 = 2$. The quality is: $Q_3 = (0 + 2)/(3 \times 2) = 1/3$. The potentials are: $\Pi^u_3 = 1$; $\Pi^v_3 = -3$.

Now we show how to modify our definitions so as to allow using them both online and offline. The algorithms presented in this paper invoke a CDCL SAT solver once to generate all the models and restart the search immediately after a new model is discovered. We call the algorithms/solvers which follow the above-mentioned scheme *compact.* Suppose that a compact Diverse$k$Set solver has found $m > 0$ models and is searching for a new model. Such a solver maintains the current partial assignment. We modify the notions of p^u_m/n^u_m, the variable quality S^u_m, and the variable potential Π^u_m simply by considering the current partial assignment as another model when counting the number of 1's and 0's assigned to a variable (the modification is required for assigned variables only). To generalize the notion of overall quality, one needs to make an adjustment, since the number of satisfying assignments is now different for assigned and unassigned variables. Let q_1 be the number of unassigned variables and q_2 be the number of assigned variables. Then we have: $Q_m = (\sum_{u \in V} S^u_m)/(\binom{m}{2} q_1 + \binom{m+1}{2} q_2)$. Note that the online versions of our definitions are a strict generalization of our notions (namely, those of p^u_m, n^u_m, S^u_m, Π^u_m, and Q_m) in the sense that they converge with the offline versions after the solver has completed its run. When mentioning these notions in the paper, we are referring to their online version when a Diverse$k$Set solver invocation is analyzed online and either to their online or offline version if the solver has finished.

4 Analizing existing algorithms

First, we refer to two non-compact algorithms, dpll-based sampling (mentioned in [6] without a reference) and xor-sample [7], that were designed for the kSampling problem, but which can also be applied to Diverse$k$Set. Both dpll-based sampling and xor-sample invoke a SAT solver once to generate each model. Diversification is achieved by randomizing the first value assigned to each variable for dpll-based sampling and by adding random XOR constraints for xor-sample. dpll-based sampling and xor-sample are shown in [2, 15] to be inferior to compact Diverse$k$Set algorithms in terms of both quality and run-time. In addition, [15] analyzes a compact DIVERSEkSET algorithm AllSAT-Sampling [2], which yields a low quality but is very fast. The present work concentrates on compact algorithms that yield a relatively high quality.

PRAND [2] (Rand-k-SAT in [2]) is a compact randomized polarity-based algorithm. It operates by overriding the traditional polarity selection heuristic to select the polarity randomly on all occasions. PRAND can be thought of as a generalization of DPLL-BASED SAMPLING.

PGUIDE [2] (Guide-k-SAT in [2]) is a compact guided polarity-based algorithm. It is designed to greedily improve quality. PGUIDE does not change the default behavior of the SAT solver before the first model is encountered. Assume PGUIDE is about to decide on the polarity of a newly assigned variable u when $m > 0$ models have already been found. If $\Pi_m^u > 0$, u is assigned 0; if $\Pi_m^u < 0$, u is assigned 1; if $\Pi_m^u = 0$, u is assigned a random value. Prop. 1 shows that this simple algorithm improves quality whenever the variable decision heuristic picks an unassigned variable with a non-zero potential. Note that this useful property does not hold for PRAND, hence PGUIDE is expected to result in better quality.

Proposition 1. *Assume that a compact* DIVERSEkSET *solver employing* PGUIDE *is running and that it has encountered $m > 0$ models. Let u be an unassigned variable picked by the variable decision heuristic. Let Q_m^1 and Q_m^0 be the qualities if the current partial assignment is extended by assigning a value 1 and 0, respectively, to u. Then, if $\Pi_m^u > 0$ then $Q_m^0 > Q_m^1$; if $\Pi_m^u < 0$ then $Q_m^1 > Q_m^0$; if $\Pi_m^u = 0$ then $Q_m^1 = Q_m^0$.*

Proof. Assume $\Pi_m^u > 0$. The recursive definition of variable quality implies that assigning u the value 1 or 0 will change its variable quality by n_m^u or p_m^u, respectively. The assumption $\Pi_m^u > 0$ implies that $p_m^u > n_m^u$. Hence, the change in the variable quality of u will be greater if u is assigned 0 than if it is assigned 1. Note that the change in the overall quality is proportional to the change in the variable quality of u, since the variable quality of u is the only addendum that changes in the dividend of the definition of quality, while the change in the divisor is independent of the value assigned to u. Hence we have $Q_m^0 > Q_m^1$. The proof for the other two cases is similar. □

PGUIDE is designed to correct the potential of one variable at a time, where by correcting the potential we mean bringing the absolute potential closer to 0. This operation improves the quality of one specific variable. Such a strategy yields the optimal overall quality given a tautological formula. However, it does not take into account dependencies between variables, which appear in real-world well-structured formulas.

PBCPGUIDE [2] (BCP-aware Guide-k-SAT in [2]) is a refinement of PGUIDE which takes into consideration dependencies between variables by taking into account the impact of Boolean Constraint Propagation (BCP) on quality. It performs BCP for both polarities of a new decision variable u alternatively and measures the new quality for each polarity. It then continues with the polarity that yielded the better quality. PBCPGUIDE's flow is detailed in [2]. Note that unlike PGUIDE, PBCPGUIDE is designed to take into account dependencies between variables. Applying PBCPGUIDE might have a significant negative impact on performance, since it has to perform BCP two or three times per decision. To be able to control the trade-off between quality and run-time, one can limit PBCPGUIDE usage as follows. PBCPGUIDE will be used until a user-given number of conflicts T is encountered by the solver. Afterwards, the algorithm will switch to plain PGUIDE until the next model is encountered. Then PBCPGUIDE is reinvoked until T conflicts are encountered again.

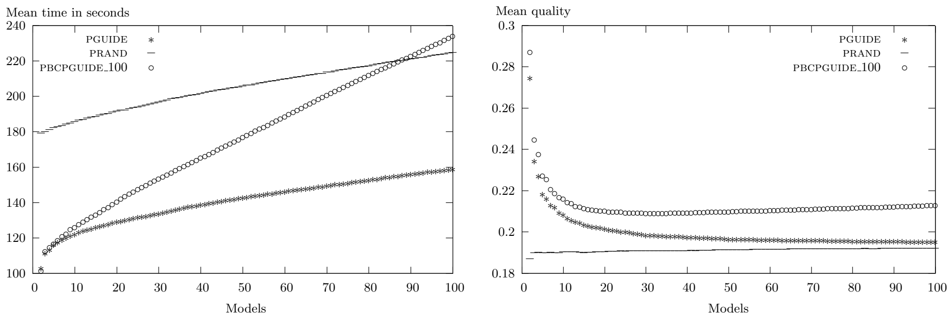

Fig. 2: Comparing polarity-based algorithms.

Compare the empirical behavior of PGUIDE and PRAND in Fig. 2 and Table 1. Both the mean run-time and the mean quality of PGUIDE is consistently better than that of PRAND for any number of models. The difference in quality is especially significant for the first models, but it goes down quickly as the number of models increases and seems to approach an asymptote.

Compare the behavior of PBCPGUIDE to that of PGUIDE in Table 1. Four versions of PBCPGUIDE with different threshold values $T \in \{10, 100, 1000, 10000\}$ (called PBCPGUIDE_T for each T) were tested. Predictably, PBCPGUIDE has a positive impact on quality, but deteriorates performance, where the effect is more significant for larger threshold values. Interestingly, while PBCPGUIDE_1000 and PBCPGUIDE_10000 yield almost the same quality, the run-time of PBCPGUIDE_10000 is significantly inferior. Hence it is not worth using a threshold greater than 1000. The balance between quality and run-time achieved by PBCPGUIDE_100 is attractive. For example, for generating 100 models, PBCPGUIDE_100 yields a better quality than PGUIDE by 9.1% and is slower by only 47.2%. Compare the behavior of PBCPGUIDE_100 as a function of the number of models with that of PGUIDE in Fig. 2. The run-time function of PBCPGUIDE_100 goes up much more quickly than that of PGUIDE; however, the gap for 100 models is still reasonable. The quality function of PBCPGUIDE_100 is always higher than that of PGUIDE. Interestingly, the gap increases with the number of models. Moreover, while PGUIDE's quality function is monotonically decreasing, PBCPGUIDE_100's quality function's tail is increasing. This is related to the fact that, unlike PGUIDE, PBCPGUIDE takes into account dependencies between variables. Further explanation regarding the behavior of the quality functions of pguide and pbcpguide_100 and the difference between them is provided in [15].

Table 1: Mean quality and mean run-time for Diverse*k*Set algorithms, given 100, 50, and 10 models on 66 benchmarks from the semiformal verification of hardware. All the numbers except the last row are relative to the behavior of pguide. For example, pbcpguide_100 yields better quality than pguide by 9.1%, but is slower by 47.2% for generating 100 models. The algorithms are sorted by the quality obtained when generating 100 models. The absolute mean quality and mean run-time in seconds of pguide are provided in the last row.

	100		**50**		**10**	
Algorithm	**Quality**	**Time**	**Quality**	**Time**	**Quality**	**Time**
pbcpguide_100-vrandglob_30	1.123	4.649	1.119	4.536	1.087	4.435
pbcpguide_100-vrandglob_20	1.121	3.829	1.114	3.653	1.081	3.469
pbcpguide_100-vrandglob_10	1.117	2.925	1.109	2.694	1.074	2.474
pbcpguide_10000	1.111	7.628	1.111	6.731	1.1	4.559
pbcpguide_1000	1.11	3.465	1.11	2.677	1.082	1.505
pbcpguide_100-vguideglob_200	1.107	1.707	1.09	1.509	1.063	1.259
pbcpguide_100-vguideglob_100	1.107	1.665	1.09	1.472	1.058	1.224
pbcpguide_100-vrandglob_2	1.106	2.007	1.093	1.766	1.05	1.52
pbcpguide_100-vguideglob_10	1.105	1.656	1.087	1.44	1.055	1.171
pbcpguide_100-vrandloc	1.102	1.742	1.083	1.537	1.036	1.348
pguide-vrandglob_30	1.099	4.058	1.095	4.181	1.062	4.243
pbcpguide_100-vguideloc	1.097	1.537	1.078	1.317	1.047	1.121
pguide-vrandglob_20	1.091	3.331	1.086	3.407	1.055	3.422
pbcpguide_100	1.091	1.472	1.068	1.239	1.036	1.033
pbcpguide_100-vrandloc-naïve	1.085	1.733	1.07	1.53	1.041	1.375
pguide-vrandglob_10	1.077	2.4	1.072	2.426	1.041	2.42
pbcpguide_100-vguideloc-naïve	1.076	1.632	1.06	1.367	1.045	1.118
pguide-vguideglob_200	1.054	1.288	1.057	1.282	1.04	1.203
pguide-vguideglob_100	1.053	1.262	1.051	1.253	1.033	1.199
pbcpguide_10	1.042	1.071	1.039	1.03	1.03	1.016
pguide-vrandglob_2	1.042	1.484	1.042	1.483	1.021	1.485
pguide-vguideglob_10	1.041	1.202	1.039	1.202	1.031	1.132
pguide-vrandloc	1.036	1.263	1.033	1.274	1.01	1.296
pguide-vguideloc	1.014	1.004	1.016	1.008	1.021	1.013
pguide	1	1	1	1	1	1
prand	0.9839	1.413	0.9731	1.439	0.9123	1.519
The absolute numbers for pguide	0.1954	159	0.1968	143	0.2086	122

5 Variable-based Methods

This section introduces a number of *variable-based* compact algorithms for Diverse*k*Set. We show how diversification quality can be improved by changing the variable ordering. We propose local and global variable-based methods. Local methods select the next decision variable from a subset of variables considered as relevant by the variable decision heuristic, while global methods consider a wider set of variables. Local variable-based methods are expected to result in a moderate quality improvement, but be run-time-efficient. Global variable-based methods are expected to be more costly in terms of performance, but yield better diversification quality. We propose both guided and randomized variable-based algorithms. Guided variable-based methods select variables with the largest absolute potential. Randomized variable-based methods add a certain degree of randomness to the variable ordering. We take pguide and pbcpguide_100 as the baseline algorithms in the sense that we integrate our variable-based methods into a solver that already uses pguide or pbcpguide_100.

Section 5.1 proposes local variable-based algorithms intended to be integrated with PGUIDE. Section 5.2 shows how to modify these algorithms of Section 5.1 in order to combine them with PBCPGUIDE_100. Section 5.3 presents the global variable-based algorithms. Our algorithms are integrated within the CBH decision heuristic.

5.1 Local variable-based methods for PGUIDE

VGUIDELOC is a *guided local variable-based* algorithm. VGUIDELOC changes the variable decision heuristic after $m > 0$ models have already been found as follows. It picks an unassigned variable with the maximal absolute potential from the topmost non-satisfied clause in the clause list. If more than one variable have the same absolute potential, a variable with the greatest VSIDS score is picked as in the original CBH. VGUIDELOC is designed to increase PGUIDE's positive impact on quality, since, according to Prop. 2, picking a variable with a larger absolute potential improves quality by a larger margin. PGUIDE-VGUIDELOC (that is, the combination of PGUIDE and VGUIDELOC) is not expected to significantly deteriorate the performance of the solver, since this strategy makes only minimal changes to the default CBH heuristic. Prop. 3 yields that VGUIDELOC picks variables with the worst variable quality in the clause and corrects its potential. However, since PGUIDE is unaware of dependencies between variables, PGUIDE-VGUIDELOC might deteriorate the quality of other variables assigned by BCP. This fact might hurt the ability of PGUIDE-VGUIDELOC to improve overall quality.

VRANDLOC is a *randomized local variable-based* algorithm. VRANDLOC picks a random variable from the topmost non-satisfied clause. PGUIDE-VRANDLOC (that is, the combination of PGUIDE and VRANDLOC) is fairer than both plain PGUIDE and PGUIDE-VGUIDELOC with respect to variable ordering, since PGUIDE-VRANDLOC may choose variables that would rarely or never be chosen by the other two methods due to their low VSIDS score or low absolute potential. Randomized variable-based method will work better than the guided method when there are many hidden dependencies between variables. PGUIDE-VRANDLOC is expected to have a negative impact on the performance of the solver, since it violates the locality principle, yet this impact should not be too significant, because the variables are still picked from the same clause.

Below, after proving a useful lemma, we provide two propositions that are essential for understanding the ideas behind VGUIDELOC.

Lemma 1. *Assume that a compact* DIVERSEkSET *solver employing* PGUIDE *is running and that it has encountered $m > 0$ models. Let v and u be two unassigned variables, such that $|\Pi^u_m| > |\Pi^v_m|$. Then $max(p^u_m, n^u_m) > max(p^v_m, n^v_m)$.*

Proof. The definition of the potential implies that for every variable t it holds that $|\Pi^t_m| = max(p^t_m, n^t_m) - min(p^t_m, n^t_m)$. Since $p^t_m + n^t_m = m$, we have $|\Pi^t_m| = max(p^t_m, n^t_m) - (m - max(p^t_m, n^t_m)) = 2 \times max(p^t_m, n^t_m) - m$. The latter equation implies that if $|\Pi^u_m| > |\Pi^v_m|$ then $max(p^u_m, n^u_m) > max(p^v_m, n^v_m)$. □

Proposition 2. *Assume that a compact* DIVERSEkSET *solver employing* PGUIDE *is running and that it has encountered $m > 0$ models. Let v and u be two unassigned variables, such that $|\Pi^u_m| > |\Pi^v_m|$. Assume that the solver is about to assign either u or v. Let the quality* after *u or v is assigned be Q^u_m or Q^v_m, respectively. Then, $Q^u_m > Q^v_m$.*

Proof. Recall from the proof of Prop.1 that the change in overall quality is proportional to the change in the quality of the variable picked by the variable decision heuristic. PGUIDE's flow implies that if it holds that $p^t_m > n^t_m$ or $n^t_m > p^t_m$ for an unassigned variable t picked by the decision heuristic, then PGUIDE will pick the value 0 or 1, respectively, for t. This latter fact and the recursive definition of variable quality imply that the change in variable quality following an assignment of t by PGUIDE is $max(p^t_m, n^t_m)$. By Lemma 1, $max(p^u_m, n^u_m) > max(p^v_m, n^v_m)$. □

Proposition 3. *Assume that a compact* DIVERSEkSET *solver employing* PGUIDE *is running and that it has encountered $m > 0$ models. Let u and v be two unassigned variables. If $|\Pi^u_m| > |\Pi^v_m|$, then $S^u_m < S^v_m$.*

Proof. Assume to the contrary that $S^u_m \geq S^v_m$. We denote $max(p^u_m, n^u_m)$ and $max(p^v_m, n^v_m)$ by x and y, respectively. The definition $S^t_m = p^t_m \times n^t_m$ and our assumption imply that $x(m-x) \geq y(m-y)$. By Lemma 1, if $|\Pi^u_m| > |\Pi^v_m|$ then $x > y$, hence one can express x as $x = y + \delta$, where $\delta > 0$. Substituting the latter equality into $x(m-x) \geq y(m-y)$ gives us: $x(m-x) \geq (x-\delta)(m-x+\delta)$. Opening parenthesis and simplifying gives us the following inequality: $\delta(2x - m - \delta) \leq 0$. Since $\delta > 0$, we have $2x - m - \delta \leq 0$. Substituting $\delta = x - y$ into the latter inequality gives us the following one: $m \geq x + y$. Definitions imply that $max(p^t_m, n^t_m) \geq m/2$ for any unassigned t, hence $x, y \geq m/2$. Since $x > y$, it must hold that $x > m/2$. Since $y \geq m/2$ and $x > m/2$, it cannot hold that $m \geq x + y$. Contradiction. □

Compare the empirical behavior of PGUIDE-VGUIDELOC and PGUIDE-VRANDLOC to that of plain PGUIDE in Table 1 and Fig. 3. Both PGUIDE-VGUIDELOC and PGUIDE-VRANDLOC yield a consistent improvement in quality over PGUIDE. PGUIDE-VGUIDELOC's run-time penalty over PGUIDE is negligible, while PGUIDE-VRANDLOC is 26% slower than PGUIDE for 100 models. PGUIDE-VRANDLOC yields better quality than PGUIDE-VGUIDELOC when the number of models exceeds 14. As the number of models increases, PGUIDE-VRANDLOC's advantage in quality becomes more significant. Hence, in the long run, being fairer with respect to variable ordering contributes to PGUIDE's impact on quality more than correcting the potential of one variable at a time, even though the selected variable has the largest absolute potential in the topmost clause. The guided variable-based approach is more efficient in terms of run-time than the randomized approach, since the guided method is closer to the efficient default variable decision heuristic: while the randomized method chooses variables, independently of their VSIDS score, the guided method prefers variables with the highest VSIDS score out of all the variables with the same potential. Interestingly, both PGUIDE-VGUIDELOC and PGUIDE-VRANDLOC are inferior to PBCPGUIDE_10 in terms of quality. In the next section we show how one can combine the variable-based algorithms with PBCPGUIDE.

5.2 Local variable-based methods for PBCPGUIDE

We considered a number of ways of combining local variable-based methods with PBCPGUIDE. First, one could integrate VGUIDELOC and VRANDLOC as is into PBCPGUIDE. However, it is unclear whether the resulting algorithms (named PBCPGUIDE_100-VGUIDELOC-NAÏVE and PBCPGUIDE_100-VRANDLOC-NAÏVE, respectively) would yield better quality than plain PBCPGUIDE, since, unlike PGUIDE, PBCPGUIDE tries to improve quality by taking into account dependencies between variables per se. It might turn out that additional variable-based considerations are not required. Second, one could apply VGUIDELOC and VRANDLOC only after the threshold on the number of conflicts for PBCPGUIDE is reached. We dub the resulting strategies PBCPGUIDE_100-VGUIDELOC and PBCPGUIDE_100-VRANDLOC, respectively. They utilize the power of both PBCPGUIDE and the combination of VGUIDELOC and VRANDLOC with PGUIDE to improve quality by taking into account dependencies between variables, yet they should not be as costly as applying PBCPGUIDE with a larger threshold.

Consider the behavior of PBCPGUIDE_100-VGUIDELOC and PBCPGUIDE_100-VRANDLOC in Table 1 and Fig. 3. These methods improve the quality of plain PBCPGUIDE_100 at the expense of run-time. Note that while the guided method is faster than the randomized method, the randomized method is preferable to the guided method in terms of quality. Recall that we observed (and explained) a similar behavior when we combined the variable-based methods with PGUIDE. Compare the empirical behavior of PBCPGUIDE_100-VGUIDELOC-NAÏVE and PBCPGUIDE_100-VRANDLOC-NAÏVE with plain PBCPGUIDE_100 in Table 1. Not only do these strategies not improve the quality of PBCPGUIDE_100, but they deteriorate both the quality and the run-time for 100 models as well. Hence the variable-based methods that try to take into account dependencies between variables interfere with PBCPGUIDE.

It would be interesting to try a BCP-aware guided variable-based method, which would apply BCP for both polarities for more than one variable and pick the variable and polarity that yield the best quality. We did not implement such an algorithm, since a straightforward implementation would be extremely costly in terms of run-time. Applying BCP in parallel and borrowing techniques from look-ahead SAT solvers [16], designed to consider a wider set of variables at each decision point, are appealing directions for future work to improve quality. The present work proposes another way to improve quality: global variable-based methods which apply the same principles as the local methods but consider a wider set of variables at each decision point. These global variable-based methods are discussed in the next section.

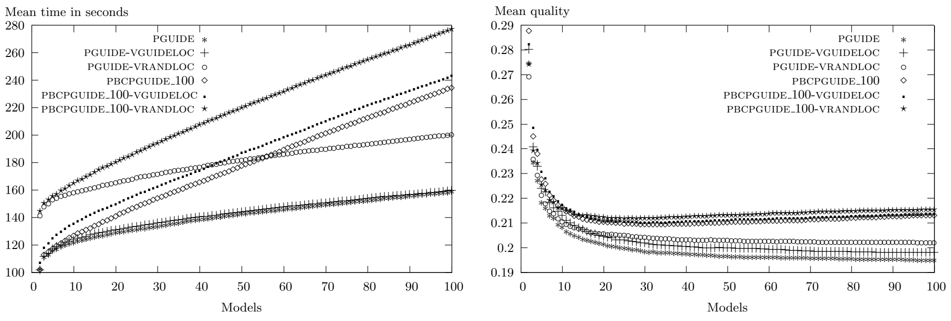

Fig. 3: Combining local variable-based algorithms with PGUIDE and PBCPGUIDE_100.

5.3 Global variable-based methods

VGUIDEGLOB is a *global guided variable-based* algorithm. It picks a variable v with the largest absolute potential out of a wider selection of clauses than its local counterpart. VGUIDEGLOB picks a variable with the greatest potential from the N topmost clauses, including satisfied clauses, unless the N topmost clauses are satisfied, in which case the algorithm considers all the clauses in the list up to and including the topmost non-satisfied clause. The primary criteria for breaking ties is to prefer a variable from a clause that is as close as possible in the list to the topmost non-satisfied clause. The secondary criteria for breaking ties is to prefer variables with better VSIDS scores. The idea behind the tie-breaking strategies is to make the heuristic as efficient as possible by making it as close as possible to the original CBH. For our experiments, we used $N \in \{10, 100, 200\}$. We refer to the combination of PGUIDE and PBCPGUIDE_100 with VGUIDEGLOB with a parameter value N as PGUIDE-VGUIDEGLOB_N and PBCPGUIDE_100-VGUIDEGLOB_N, respectively.

VRANDGLOB is a *global randomized variable-based* algorithm. It picks an unassigned variable at random in T% of the cases out of all the decisions, where T is a parameter. The default decision heuristic is used in the rest of the cases. Note that this strategy is independent of the decision heuristic. We tried $T \in \{2, 10, 20, 30\}$ for our experiments. We refer to the combination of PGUIDE and PBCPGUIDE_100 with VRANDGLOB with a parameter value T as PGUIDE-VRANDGLOB_T and PBCPGUIDE_100-VRANDGLOB_T, respectively.

We combined global variable-based methods with PBCPGUIDE by applying them only after the threshold on the number of conflicts for PBCPGUIDE was reached, since this was found in Section 5.2 to be the optimal strategy for integrating local variable-based methods with PBCPGUIDE.

Consider the empirical behavior of the combination of the global variable-based methods with PGUIDE and PBCPGUIDE_100 in Table 1. The following conclusions equally hold for combining global variable-based methods with either PGUIDE or PBCPGUIDE_100. As expected, the global methods yield better quality than the local methods (combined with the respective polarity-based algorithm), but do so at the expense of run-time. This observation holds for both randomized and guided methods. The parameter N or T for randomized or guided methods, respectively, can be used to control the trade-off between quality and run-time. The larger the corresponding parameter, the better the quality and the worse the run-time. However, increasing the parameter too much does not yield added benefit, since the improvement in quality becomes marginal, while the run-time continues to increase. Compare now the global guided methods vs. the global randomized methods. The randomized methods are costly in terms of run-time, but yield better quality (when the threshold N is higher than 2). Recall that we observed the same behavior while analyzing local methods.

Interestingly, PBCPGUIDE_100-VRANDGLOB_T, when T is sufficiently large, outperforms plain PBCPGUIDE with threshold 10000 in terms of both quality and run-time for 100 and 50 models. This result shows that, in the long run, it pays first to use PBCPGUIDE_100 and then to switch to PGUIDE-VRANDGLOB, rather than to use plain PBCPGUIDE with a larger threshold. Understanding the reasons for this phenomenon is left for future research.

6 Conclusions and Future Work

This work is the first full-blown paper dedicated to the Diverse$k$Set problem in SAT, that is, the problem of efficiently generating a number of diverse solutions (satisfying assignments) given a propositional formula. We proposed a framework for analyzing Diverse$k$Set algorithms in SAT and carried out an extensive empirical evaluation of existing and new algorithms on large industrial instances of Diverse$k$Set arising in SAT-based semiformal verification of hardware [2].

Our analysis showed that adapting the SAT solver's polarity selection heuristic in a guided way, that is, explicitly avoiding previous solutions, is preferable to randomizing the polarity. Considering the dependencies between variables by taking into account the effect of BCP improves diversification quality, but deteriorates run-time.

We introduced a number of variable-based algorithms that improve diversification quality by adapting the variable decision heuristic in addition to the polarity selection heuristic. We distinguished between randomized and guided algorithms as well as between local and global algorithms. Randomized and global algorithms are more costly in terms of run-time but yield better quality than guided and local algorithms. Overall, while a moderate improvement in quality over purely polarity-based methods can be achieved at a negligible run-time cost, obtaining a more significant improvement in quality requires additional run-time. We showed how one can control the trade-off between quality and run-time to achieve an attractive balance. The eventual choice of algorithms should depend on the needs of each specific application.

The following directions for future research seem attractive. Parallelizing Diverse$k$Set algorithms should be helpful in improving both quality and run-time. It would be interesting to investigate ways to adapt various components of the SAT solver (such as conflict analysis schemes or restart strategies) to Diverse$k$Set. Borrowing techniques from look-ahead SAT solvers and developing Diverse$k$Set algorithms on top of such solvers is another interesting direction.

Acknowledgments

The author would like to thank Amit Palti for supporting this work, Paul Inbar for editing the paper, and Jim Grundy, Vadim Ryvchin, and Yevgeny Schreiber for providing useful suggestions that helped to improve this work.

References

1. Biere, A., Heule, M., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. Volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press (2009)
2. Agbaria, S., Carmi, D., Cohen, O., Korchemny, D., Lifshits, M., Nadel, A.: SAT-based semiformal verification of hardware. In Bloem, R., Sharygina, N., eds.: Proceedings of the 10th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2010). (October 2010) 25–32
3. Hebrard, E., Hnich, B., O'Sullivan, B., Walsh, T.: Finding diverse and similar solutions in constraint programming. In Veloso, M.M., Kambhampati, S., eds.: AAAI, AAAI Press / The MIT Press (2005) 372–377
4. Schreiber, Y.: Value-ordering heuristics: Search performance vs. solution diversity. In Cohen, D., ed.: CP. Volume 6308 of Lecture Notes in Computer Science., Springer (2010) 429–444
5. Dechter, R., Kask, K., Bin, E., Emek, R.: Generating random solutions for constraint satisfaction problems. In: AAAI/IAAI. (2002) 15–21
6. Kitchen, N., Kuehlmann, A.: Stimulus generation for constrained random simulation. In: ICCAD. (2007) 258–265
7. Gomes, C.P., Sabharwal, A., Selman, B.: Near-uniform sampling of combinatorial spaces using XOR constraints. In Schölkopf, B., Platt, J.C., Hoffman, T., eds.: NIPS. (2006) 481–488
8. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: DAC, ACM (2001) 530–535
9. Silva, J.P.M., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers **48** (1999) 506–521
10. Huang, J.: The effect of restarts on the efficiency of clause learning. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence. (2007) 2318–2323
11. Dershowitz, N., Hanna, Z., Nadel, A.: A clause-based heuristic for SAT solvers. In Bacchus, F., Walsh, T., eds.: SAT. Volume 3569 of Lecture Notes in Computer Science., Springer (2005) 46–60
12. Frost, D., Dechter, R.: In search of the best constraint satisfaction search. In: AAAI. (1994) 301–306
13. Strichman, O.: Tuning SAT checkers for bounded model checking. In Emerson, E.A., Sistla, A.P., eds.: CAV. Volume 1855 of Lecture Notes in Computer Science., Springer (2000) 480–494
14. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In: SAT. (2007) 294–299
15. Nadel, A.: Generating diverse solutions in SAT: Paper addendum. `http://www.cs.tau.ac.il/research/alexander.nadel/multiple_cex_addendum.pdf` (2011)
16. Heule, M., van Maaren, H.: Look-ahead based SAT solvers. [1] 155–184