*Diver*SAT: A Novel and Effective Local Search Algorithm for Diverse SAT Problem

Jiaxin Liang[1], **Junping Zhou**[1*], **Minghao Yin**[1, 2*]

[1]School of Information Science and Technology, Northeast Normal University, China
[2]Key Laboratory of Applied Statistics of MOE, Northeast Normal University, China
{liangjx348,zhoujp877,ymh}@nenu.edu.cn

Abstract

For many real-world problems, users are often interested not only in finding a single solution but in obtaining a sufficiently diverse collection of solutions. In this work, we consider the Diverse SAT problem, aiming to find a set of diverse satisfying assignments for a given propositional formula. We propose a novel and effective local search algorithm, *Diver*SAT, to solve the problem. To cope with diversity, we introduce three heuristics and a perturbation strategy based on some relevant information. We conduct extensive experiments on a large number of public benchmarks, collected from semiformal hardware verification, logistics planning, and other domains. The results show that *Diver*SAT outperforms the existing algorithms on most of these benchmarks.

Datasets — https://github.com/LyreRabbit/DiverseSAT

Introduction

The Boolean Satisfiability Problem, often abbreviated as SAT, is a fundamental problem in computer science (Biere et al. 2021). SAT has broad applications across various fields, such as intelligent planning, software testing, formal verification, and electronic design automation. It focuses on determining a solution (also known as a model) to a given logical propositional formula. However, many logical propositional formulae only approximate complex real-world problems and cannot capture all details and uncertainties in these problems. Moreover, in numerous real-world application scenarios, relying solely on a single solution to the original problems falls short of meeting their coverage demands. One potential approach to addressing this issue involves finding diverse solutions to the logical formula and allowing users to choose their preferences. Unfortunately, although researchers have developed various techniques to design advanced SAT solvers, the current solvers still return the same or similar solutions when running repeatedly, which motivates the research on the Diverse SAT problem, first introduced by Nadel (Nadel 2011).

The Diverse SAT is a problem of computing a set of k diverse solutions of a given logical propositional formula, such that the sum of the pairwise distances is maximized. It contributes to many valuable applications in many fields. For example, in intelligent planning, the agents benefit from multiple diverse solutions because changes are pervasive in our dynamic and uncertain world (Ghasemi et al. 2021). In software testing, automated software testing requires diverse solutions for maximizing fault detection rate or structural coverage criteria to ensure the reliability of a software product line (Devroey et al. 2016). In semiformal verification, verification engineers attempt to generate different witnesses (solutions) with the aim of achieving sufficient coverage and detecting corner-case bugs (Agbaria et al. 2010).

In recent years, diversity of the solutions has been discussed and explored in various fields, particularly in combinatiorial optimization problems (Hanaka et al. 2023; Petit and Trapp 2015). In SAT and constraint satisfaction problems, there are several studies on diversity between two solutions. For example, Distance-SAT problem determines whether there exists a solution that differs from a given configuration by at most d variable assignments (Bailleux and Marquis 1999), while Max Differ SAT problem focuses on finding two solutions with a minimum distance threshold of d (Misra, Mittal, and Rai 2024). Some researchers have also studied algorithms for a series of problems from the perspective of complexity, aiming at finding a pair of maximally far-apart solutions, known as Max Hamming Distance (Crescenzi and Rossi 2002), such as Max Hamming XSAT (Dahllöf 2005; Hoi and Stephan 2019). Additionally, a similar paradigm in the SAT literature is uniform solution sampling, where the objective is to find multiple solutions without directly considering diversity (Chakraborty et al. 2015; Dutra et al. 2018). However, research on algorithms specifically designed for Diverse SAT is limited. To the best of our knowledge, the currently available algorithms for Diverse SAT (referred to as Diverse$k$Set in the literature) are proposed by Nadel (Nadel 2011). These algorithms enumerate k diverse solutions by adjusting the variable ordering in the CDCL-based solver. There remain many aspects for improvement, such as the solution quality.

In this work, we develop an efficient local search algorithm for the Diverse SAT problem, called *Diver*SAT. *Diver*SAT performs local search in the space of models by starting with a solution S containing k diverse models and then repeatedly replacing a model in S with a new one to increase the diversity quality. In order to generate diverse models, we

*Corresponding author

design a local search sub-algorithm *GSMD* integrating three new heuristics: a variable decision heuristic, a polarity selection heuristic, and a Strengthening Configuration Checking with Aspiration (SCCA) heuristic. Unlike classical local search for SAT, where the initial candidate models are randomly generated, *GSMD* tries to construct diverse initial candidate models by employing the variable decision heuristic and polarity selection heuristic based on the relevant statistical information from the current solution S, with the aim of enhancing diversity. Subsequently, a greedy heuristic SCCA is applied to ensure this diversity. In the optimization process, we propose a scoring function to reflect the diversity of each model over the solution S, which abandons the objective and measures the uniqueness of a model with respect to the rest of the individuals in S. Furthermore, we introduce a new adaptive perturbation mechanism. With this mechanism, the solution updates by incorporating not only the traditional greedy heuristic in the global local search, but also the adaptive perturbation in the focused local search. To evaluate *Diver*SAT, we conduct extensive experiments on a large number of publicly available SAT instances. Our experimental results indicate *Diver*SAT outperforms the existing algorithms on most instances. Further, the results also confirm the effectiveness of core techniques in *Diver*SAT.

The remainder of our paper is structured as follows. In Section 2, we describe some preliminary definitions. Section 3 provides the algorithm framework and several vital techniques in detail. In Section 4, we analyze experimental results on CNF benchmarks to present the efficiency of *Diver*SAT. Section 5 concludes this paper.

Preliminaries

Let $X = \{x_1, x_2, \dots\}$ be the set of Boolean *variables*. The set of *literals* $L = \{l_1, l_2, \dots\}$ consists of variables and their negations. A *clause* is a set of literals connected by $\vee$. If a clause contains only one literal l, then l is referred to as an *unit clause*. A formula in Conjunctive Normal Form (CNF) is a conjunction of clauses. Given a formula ϕ over X, an *assignment* α is a mapping from each variable $x \in X$ to $\{1, 0\}$, where 1 and 0 represent *true* and *false*, respectively. The number of assigned variables in α is denoted as $|\alpha|$, and the value of x under α is denoted as $\alpha(x)$. A literal is satisfied if it is assigned to 1. A clause is satisfied if at least one of its literals is satisfied. A CNF formula is satisfied if all of its clauses are satisfied. Given a formula ϕ, the *Propositional Satisfiability Problem* (SAT) is to determine whether there is an assignment on X satisfying ϕ. Such an assignment is called a *satisfying assignment* (or *model*). If the value of a variable is invariant in all models of a given formula, the variable is said to be a *backbone* variable. If a unit clause l is existed in a formula ϕ, then l is assigned to 1, and the clauses containing l and the literal $\neg l$ are removed from ϕ and the remaining clauses respectively. The above operation is executed iteratively until there is no unit clause in ϕ. This process is called *unit propagation*. During the process, SAT solvers usually maintain a *trail* which is the sequence of literals that be currently assigned true.

Definition 1 (**Hamming Distance**). *Let ϕ be a CNF formula and $SA(\phi) = \{\alpha_1, \alpha_2, \dots\}$ be a set containing all models of ϕ. The Hamming distance between two models $\alpha_i, \alpha_j \in SA(\phi)$, denoted by $dH(\alpha_i, \alpha_j)$, is defined as the number of different values for the corresponding variables between the two models.*

Definition 2 (**Diverse SAT Problem**). *Given a formula ϕ and a positive integer k, the Diverse SAT Problem is to find a subset $S \subseteq SA(\phi)$ with k diverse models, such that the sum of Hamming distances between each pair of models in S is maximized.*

It is clear that the objective of the Diverse SAT Problem is to maximize the following equation:

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} dH(\alpha_i, \alpha_j), \alpha_i, \alpha_j \in S \tag{1}$$

In the rest of the paper, we express Equation (1) as $DQ(S)$ to measure the diversity quality of the solution S of a Diverse SAT instance.

Algorithm for Diverse SAT

Since local search has exhibited great success in solving various well-known problems for diversity (Wu et al. 2020; Zhou et al. 2023; Wu et al. 2023), we propose an effective local search algorithm, *Diver*SAT, for the Diverse SAT problem. We first introduce the top-level design of the algorithm, and then present the core algorithmic mechanisms.

Top-Level Design of *Diver*SAT

The algorithm, as described in Algorithm 1, consists of two phases: the initialization phase and the optimization phase. The algorithm first initializes the solution S, performs the initial simplification of the formula and assigns the necessary truth values at the first level (line 1). In the initialization phase, an efficient *GSMD* (Generate a Single Model with Diversity) algorithm is carried out to initialize the solution S with k diverse models (lines 3–5). Then *Diver*SAT proceeds to the optimization phase to update the solution S, where cnt is used in Algorithm 3 (lines 6-7). When the time limit is reached, the solution S is output (line 8).

Actually, there are two challenges that affect the performance of our local search algorithm.

- **Diversity Challenge**. Clearly, "diversity" is an important influential factor in improving the quality of the solution achieved by our local search algorithm, *Diver*SAT. However, current SAT algorithms are unable to obtain diverse models. Essentially, the goal of these algorithms is to find a model quickly, which leads them often search in a solution subspace with multiple models, resulting in the generated models that are similar to each other. This lack of diversity is not suitable for the purpose of the Diverse SAT problem. Hence, it is desirable to construct models as diverse as possible. To address this issue, an advisable solution is to focus on analyzing the key elements that influence the diversity of the models and designing the corresponding strategies to increase their dissimilarity with each other.

Algorithm 1: The basic framework of *Diver*SAT

Input: A CNF formula ϕ, parameter k, and the timelimit $cutoff$
Output: Solution S containing k models

```
1  S ← ∅ ; φ ← Preprocess(φ) ;
2  while elapsed time < cutoff do
       // initialization phase
3      while |S| < k do
4          α ← GSMD(φ, S, k) ;
5          S ← S ∪ {α} ;
       // optimization phase
6      cnt = 0 ;
7      S ← UpdateSolution(φ, S, k, cnt) ;
8  return S ;
```

- **Metric Challenge**. Generally speaking, this challenge is how to efficiently evaluate a model with respect to the other individuals already in the solution S. Indeed, the goal of the optimization phase is to ensure that the solution returned becomes increasingly dissimilar by iteratively substituting a new model for one in *S*. More specifically, given a solution $S = \{\alpha_1, \alpha_2, ..., \alpha_k\}$, the basic strategy of the optimization phase is that we first find a new model α and then evaluate whether α can replace a model in *S* so that more diverse models can be preserved in *S*. A simple evaluation method is to compute $DQ(S_i)$ $(1 \leq i \leq k)$ recursively, where S_i includes $\alpha_1, \alpha_2, ..., \alpha_{i-1}, \alpha_{i+1}, ..., \alpha_k$ and α but excludes α_i . It is clear that the computational cost significantly increases as the number of models generated by *GSMD* increases. Therefore, given a solution with different models, it is critical to design a metric that can effectively quantify the benefit of each model α with regard to *S* if α is kept in *S*.

In order to address the two challenges, we design *three* heuristics to enhance diversity of the search and present a scoring function to reflect the benefit of each model over *S*. Details are described in the following subsections.

Generating a Single Model with Diversity

In this subsection, we propose a local search algorithm, *GSMD*, based on three new strategies, namely, variable decision heuristic, polarity selection heuristic, and SCCA (Strengthening Configuration Checking with Aspiration) heuristic, to generate a more diverse model when it runs. Our strategies are derived from the following two observations:

(*Observation* 1) Different initial candidate models, which are actually multiple assignments, can guide a local search algorithm to explore different solution subspaces, and consequently converge to diverse solutions. Thus, aiming at producing a diverse model, the core idea of *GSMD* is to generate a diverse initial candidate model, and then apply a greedy strategy to compute a model so as to guarantee diversity.

(*Observation* 2) In order to create a dissimilar initial candidate model, we observe that the randomization, different variable assigning orders, and different polarity selections can achieve diversity. Hence, we will introduce a new variable decision heuristic, a new polarity selection heuristic, and a randomization method to build a diverse initial candidate model in *GSMD*. Moreover, we present a greedy SCCA heuristic to maintain diversity in the subsequent search.

The pseudocode of *GSMD* is outlined in Algorithm 2. The algorithm begins with an initial candidate model construction phase to generate a diverse initial candidate model (lines 2–13). Then, it enters the local search phase, which iteratively picks a variable from the initial candidate model to flip and gets an updated assignment based on the flip of the variable (lines 14–25). After the local search phase, a model α is returned (line 17).

Initial Candidate Model Construction Phase The initial candidate model construction phase employs a combination of randomization mechanism and guide mechanism to build an initial candidate model. With the probability pr, an initial candidate model is directly generated by the randomization mechanism (Algorithm 2, lines 2–3). With the probability $1 - pr$, an initial candidate model is obtained by the guide mechanism, which adopts relaxed CDCL (Cai and Zhang 2021) to guide the search (Algorithm 2, lines 4–13). The guide mechanism repeatedly decides a variable by variable decision heuristic (*PickVar*() function) and assigns it by polarity selection heuristic (*PickPol*() function) to increase diversity (Algorithm 2, lines 6–7). Then unit propagation is performed (Algorithm 2, line 9). If a conflict occurs, the algorithm adopts different approaches to guiding the subsequent search according to the number of assigned variables in α. If more than $r \times |X|$ variables are assigned, which is considered that the current branch is promising, the algorithm ignores the conflict and reassigns the variable in the contradictory unit clauses by *PickPol*() (Algorithm 2, lines 10–12). Otherwise, the algorithm analyzes the conflict and backtracks (Algorithm 2, line 13). The initial candidate model construction is finished when all variables are assigned. Next, we present the variable decision heuristic and the polarity selection heuristic.

- **The variable decision heuristic**: The intuition behind this strategy is that different orders of decision variables may be advantageous to generate different models with large distances in the search space. Based on the above consideration, a scoring function is proposed to evaluate each variable in a formula. Given a variable x in a formula ϕ and a last-found model α by *GSMD*, the score of x is defined as $score(x) = \lambda_1 \times$ *PRE_Order*$(x) + \lambda_2 \times$ *VSIDS*(x) (λ_1 and λ_2 are parameters), where *PRE_Order*(x) is the order in which x was assigned in α (position of x in last trail), and *VSIDS*(x) is the activity score for the variable x with the VSIDS heuristic (Moskewicz et al. 2001). Our heuristic prioritizes assigning a variable with a larger $score$ with BMS strategy (Cai 2015). More precisely, the algorithm chooses m variables randomly from all unassigned variables and returns the one with the largest $score$, where

Algorithm 2: *GSMD*(ϕ, S, k)

```
1  α ← ∅ ;
   // initial candidate model
      construction
2  if with probability pr then
       // randomization mechanism
3      α ← a random candidate model ;
4  else // guide mechanism
5      while ∃ unassigned variables do
6          x ← PickVar() ;
7          v ← PickPol() ;
8          α ← α ∪ {(x, v)} ;
9          UnitPropagation(φ,α) ;
10         if a conflict occurs then
11             if |α| > r × |X| then
12             assign the variable related to the
                 contradictory unit clauses by PickPol() ;
13             else analyze conflict and backtrack ;
   // local search
14 step ← 0 ;
15 initialize the weight of each clause ;
16 while step < max_step do
17     if α satisfies φ then return α ;
18     else
19         if the precondition of SCCA is satisfied then
20             x ← a variable selected by SCCA ;
21         else
22             c ← a random falsified clause ;
23             x ← a random variable in c ;
24     α ← α with x flipped ;
25     update the clause weights and step++;
```

m is a parameter. It is clear that a variable with a larger *PRE_Order* is likely to be assigned later.

- **The polarity selection heuristic**: The basic idea of the polarity selection heuristic is that if a variable x takes the value 1 (resp. 0) in most models in the solution S, then the variable x is more expected to take the value 0 (resp. 1) to increase the distance among the models in S. In order to enforce the heuristic, we associate a $pol(x)$ ranging from 0 to 1 for each variable x, defined as the *assigning preference* of x to guide x to take the expected polarity in the returned model. With $pol(x)$, the variable x is assigned to 1; with $1 - pol(x)$, x is 0. Note that if a variable is proved to be a backbone, it must be forced to its invariant value instead of being assigned by pol. Since *GSMD* is invoked by two phase of *Diver*SAT, the updating rules of $pol(x)$ vary based on different phases. (1) In the initialization phase, the $pol(x)$ of each variable x is initialized to 0.5. Then, $pol(x)$ is increased by $\frac{1}{2k}$ if x is assigned to 0; otherwise, it is decreased by $\frac{1}{2k}$. (2) In the optimization phase, the $pol(x)$ of each variable x is initialized to $pol(x) = 1 - \frac{\sum_{j=1}^{k} \alpha_j(x)}{k}, \alpha_j \in S$. Then $pol(x)$ is updated based on the values of x in the exchanging models in the phase. Assume that α_{old} is the replaced model and α_{new} is the new model. (i) If $\alpha_{old}(x)$ is the same as $\alpha_{new}(x)$, $pol(x)$ remains unchanged; (ii) if $\alpha_{new}(x) = 0$ and $\alpha_{old}(x) = 1$, $pol(x)$ is increased by $\frac{1}{k}$; (iii) if $\alpha_{new}(x) = 0$ and $\alpha_{old}(x) = 1$, $pol(x)$ is decreased by $\frac{1}{k}$.

Local Search Phase When a diverse initial candidate model is constructed, *GSMD* steps into the local search phase to iteratively flip a variable until it either finds a model or reaches the step limit (Algorithm 2, lines 14–25). In each iteration, *GSMD* selects a variable x to flip its value under the candidate model α using our greedy heuristic, called SCCA (Strengthening Configuration Checking with Aspiration) heuristic (Algorithm 2, lines 19–20). When a local optimum is reached, which means *GSMD* fails to find a model in current search subspace, *GSMD* randomly chooses a variable x in a random falsified clause c (Algorithm 2, lines 21–23). Finally, *GSMD* obtains the updated assignment with x flipped, and updates the clause weights used for SCCA heuristic (Algorithm 2, lines 24–25). Before describing the details of the SCCA heuristic, we first propose some definitions.

Definition 3 (**Gain of variable**). *Given a formula ϕ and a candidate model α of ϕ, the gain of the variable x is defined as $gain(x) = make(x) - break(x)$, where $make(x)$ and $break(x)$ are the total weights of the clauses which would become satisfied and falsified by flipping x, respectively.*

The weights of clauses are set based on the clause weighting mechanism (Cai, Luo, and Su 2015; Fu et al. 2021), which works as follows. At first, the weight of each clause $w(c) = 0$ if the clause c is satisfied under the current candidate model α, and $w(c) = 1$ otherwise. Then, when flipping a variable x, the update rules involve: $w(c) = w(c) + 1$ if c remains falsified, otherwise $w(c)$ does not change.

Definition 4 (**Flipping preference of variable**). *Given a formula ϕ, a current candidate model α, and a current Diverse SAT solution S of ϕ, the flipping preference of variable x under α, denoted by $fp(x)$, is defined as the preference to flip $\alpha(x)$ to $\neg\alpha(x)$ in the returned model according to the ones in the solution S.*

Based on the assigning preference $pol(x)$, which is obtained by statistical analysis on models in the current solution S, we use the following equation to compute $fp(x)$.

$$fp(x) = \begin{cases} pol(x) & \text{if} \quad \alpha(x) = 0; \\ 1 - pol(x) & \text{if} \quad \alpha(x) = 1. \end{cases} \tag{2}$$

Definition 5 (**Configuration of variable**). *Given a variable x and a candidate model α, the configuration of a variable x is a vector containing values of all variables in $N(x)$ under α, where $N(x)$, named neighbouring variables, is the set of variables that together with x appear in at least one clause.*

SCCA heuristic is a variant of CCA (Configuration Checking with Aspiration), which is originally introduced in (Cai and Su 2012) to effectively perform the local search for SAT. CCA allows the selection of two types of variables. One type of variables are the variables with $gain > 0$

and the configuration has been changed (i.e., at least one of their neighbouring variables has been flipped) since their last flip; the other type of variables are those with a significant $gain$ and the unchanged configuration. Although the original CCA successfully forbids the cycling phenomenon from which local search suffers, it is not suitable to be directly applied in the local search for the Diversity SAT problem. In fact, the local search for the Diversity SAT problem should try its best to maintain the diversity of the initial candidate models. To address the issue, we introduce a strengthening version of CCA, named SCCA, which picks variables according to not only the CCA mechanism but also the flipping preference of variables through statistical analysis on models in the current solution. The SCCA heuristic is implemented according to the following two rules.

Rule 1: If there is variable x with $gain(x) > 0$ and its configuration has been changed, then a variable with the largest $gain$ is selected, breaking ties in favor of the variable with the largest fp;

Rule 2: Else if there is a variable x with unchanged configuration and $gain(x) > w$, where w is the average weight of all clauses, then a variable with the largest $gain$ is selected, breaking ties in favor of the variable with the largest fp.

Updating Solution

After the initialization phase, a solution S is created. However, its quality is not diverse enough. Therefore, *Diver*SAT enters the optimization phase (the algorithm *UpdateSolution*), in which the unpromising model replacement and the adaptive perturbation mechanism are used to improve the solution.

In the algorithm *UpdateSolution*, how to efficiently evaluate a model with respect to the rest of the individuals already in the solution S is a core problem. In order to address the core problem, we design a scoring function $score$ to reflect the diversity of each model over a solution S.

Definition 6 (Score of model). *Given a formula ϕ and a solution S with k models of ϕ, the score of a model $\alpha \in S$ is defined as*

$$score(\alpha, S) = \frac{\sum_{i=1}^{k-1} dH(\alpha,\alpha_i)}{k-1}, \alpha_i \in S \setminus \{\alpha\},$$

where $dH(\alpha, \alpha_i)$ is the Hamming distance between two models α and α_i.

According to the above definition, our scoring function measures the uniqueness of a model with respect to the rest of the individuals, thereby indirectly quantifies each model's contribution to the optimization objective of the Diverse SAT problem. It is obvious that a model with a higher $score$ is promising, which indicates the average distance is far between α and the rest of models in S.

The pseudo-code of the algorithm *UpdateSolution* is outlined in Algorithm 3. The algorithm works in an iterative local search manner to update the solution *S*. At first, a new model α is obtained by *GSMD* (line 1), then it proceeds to the updating process (lines 2-11). After adding the new model α to the solution S (line 2), the algorithm selects the unpromising model with the lowest $score$ in S, denoted as α_{worst} (line 4). In the extended set S', if $score(\alpha, S')$ is higher than $score(\alpha_{worst}, S')$, α_{worst} is replaced by α (line 5), thus the solution is improved by the replacement. However, this case may occur during the update process: diversity of the solution does not improve significantly for a period of time. To settle this case, the algorithm adopts an adaptive perturbation mechanism. The intuition of this mechanism is that *UpdateSolution* attempts to finish updating without destroying the overall distribution of the solution space in *S*. More precisely, the improvement of diversity is not more than 0.5% over max_cnt iterations, *UpdateSolution* substitutes α for a model α_{close} closest to α instead of the unpromising model α_{worst} in *S*, which helps to escape from local optimums (lines 9-11).

Algorithm 3: *UpdateSolution*(ϕ, S, k, cnt)

```
1  α ← GSMD(φ, S, k) ;
2  S' ← S ∪ {α} ;
   // replace the worst model
3  if cnt < max_cnt then
4  |   α_worst ← arg min_{α_i ∈ S} score(α_i, S) ;
5  |   if score(α, S') > score(α_worst, S') then
   |     S ← S' \ {α_worst} ;
6  |   else S' ← S ;
   // adaptive perturbation mechanism
7  else
8  |   cnt = 0 ;
9  |   α_close ← arg min_{α_i ∈ S} dH(α, α_i) ;
10 |   if score(α, S') > score(α_close, S') then
   |     S' ← S' \ {α_close} ;
11 |   else S' ← S ;
12 if (DQ(S') − DQ(S)) / DQ(S) < 0.5% then cnt++ ;
13 else cnt = 0 ;
14 S ← S' ;
15 return S ;
```

Experimental Evaluation

In this section, we carry out experiments to evaluate the effectiveness of our algorithm, *Diver*SAT.

Experiment Preliminaries

Benchmarks. Our experiments are conducted on semiformal hardware verification benchmark (66 instances) used for testing the solving efficiency of Diverse SAT solvers in Nadel (2011), which are available at the author's homepage, and SAT application benchmarks (2670 instances) chosen from SATLIB according to the number of models each instance possesses. Since we set parameter *k* to 10, 50, and 100 in our experiments, all instances that have more than 100 models are picked from SATLIB. All of these instances comprise totally 17 types of problems, including logistics planning, bounded model checking, circuit fault analysis, and so on. Table 1 shows the number of instances ($\#inst$), and the average number of variables (avg_vars) and clauses (avg_cls) of each problem. Note that the semiformal hard-

Problem	$\#inst$	avg_vars	avg_cls	Problem	$\#inst$	avg_vars	avg_cls	Problem	$\#inst$	avg_vars	avg_cls
ais	2	223	4409	flat125-301	100	375	1403	II	41	656	7994
bmc	13	30077	160366	flat150-360	101	450	1680	logistics	4	1881	11682
flat30-60	100	90	300	flat175-417	100	525	1951	qg	1	729	15580
flat50-115	999	150	545	flat200-479	100	600	2237	ssa	4	13091	3192
flat75-180	100	225	840	GCP	4	3844	206256	SW100-8	901	500	3100
flat100-239	100	300	1117	hardware	66	213047	738862				

Table 1: Description of each problem of benchmarks.

	$k = 10$			$k = 50$			$k = 100$		
Problem	Diverse$k$Set	randAllSAT	*Diver*SAT	Diverse$k$Set	randAllSAT	*Diver*SAT	Diverse$k$Set	randAllSAT	*Diver*SAT
ais	1453(1.1)	1372(14.1)	**1817**(158.2)	44131(4.3)	43301(58.3)	**47012**(362.2)	167099(20.4)	158484(194.1)	**188914**(422.4)
bmc	23283(10.2)	25003(0.4)	**31917**(56.2)	1583201(14.4)	1333920(0.4)	**1712108**(73.1)	2510458(17.5)	2326856(0.5)	**2714649**(83.3)
flat30-60	1877(0.1)	1832(0.1)	**1978**(0.3)	49771(0.1)	47475(0.1)	**49974**(0.4)	191367(0.1)	191054(0.1)	**199954**(0.4)
flat50-115	3188(0.1)	3099(0.1)	**3300**(0.3)	74211(0.1)	79089(0.1)	**83252**(0.3)	331004(0.1)	324560(0.1)	**333216**(0.4)
flat75-180	4691(0.1)	4708(0.1)	**4942**(0.3)	121888(0.1)	118634(0.1)	**124878**(0.5)	483455(0.1)	494286(0.1)	**499868**(0.5)
flat100-239	6302(0.1)	6218(0.1)	**6590**(0.8)	162497(0.1)	158120(0.1)	**166488**(0.8)	644682(0.1)	636188(0.1)	**666402**(0.8)
flat125-301	8075(0.1)	8019(0.1)	**8229**(0.8)	197582(0.1)	197686(0.1)	**208090**(1.0)	827407(0.1)	828897(0.1)	**832943**(1.0)
flat150-360	9812(0.1)	9800(0.1)	**9874**(1.0)	233165(0.1)	237251(0.1)	**249738**(1.0)	904483(0.1)	903268(0.1)	**999654**(1.0)
flat175-417	10357(0.3)	10420(0.1)	**11514**(2.2)	267835(0.3)	276621(0.1)	**291311**(1.4)	961680(0.3)	968059(0.1)	**1166018**(2.8)
flat200-479	11092(0.6)	11804(0.1)	**13168**(3.7)	261857(0.6)	316331(0.1)	**332980**(3.4)	1036365(0.7)	1009661(0.1)	**1332922**(3.4)
GCP	20015(2006.6)	19984(1805.1)	**23478**(1011.2)	571837(2202.1)	532868(2302.3)	**581966**(1329.4)	2102148(2487.7)	2098932(2477.8)	**2331740**(1691.1)
hardware	2876134(172.2)	2638725(29.7)	**2988763**(725.1)	69595353(390.1)	66372142(30.3)	**70723412**(877.4)	210916530(433.1)	153926155(31.2)	**270244214**(1025.8)
II	8203(1.8)	8025(0.1)	**10547**(95.2)	210484(2.4)	209540(0.1)	**253324**(103.2)	834421(3.1)	816528(0.3)	**997079**(82.3)
logistics	9122(3.2)	9706(0.2)	**10758**(188.2)	242478(5.8)	231193(0.4)	**275876**(502.5)	1048665(6.1)	957251(0.4)	**1099846**(577.4)
qg	5080(3.3)	5001(0.1)	**5250**(284.2)	103084(3.6)	99157(0.3)	**136993**(733.2)	507707(4.1)	499901(0.3)	**545680**(695.2)
ssa	26614(0.1)	24159(0.1)	**28118**(310.2)	707024(0.1)	707723(0.1)	**708355**(277.2)	2714713 (0.3)	2467201(0.3)	**2834851**(322.4)
SW100-8	7716(1.1)	7694(0.1)	**7904**(401.7)	199284(1.3)	197066(0.1)	**199826**(522.1)	799516(1.9)	798811(0.5)	**799765**(577.9)

Table 2: Experimental results of three solvers on all benchmarks, comparing the average diversity of solutions and the average runtime.

ware verification benchmark is labelled as "hardware".

Experimental Setup. In our experiments, *Diver*SAT is compared against two competitive solvers for Diverse SAT, i.e., Diverse$k$Set and randAllSAT. Diverse$k$Set is the best competitor for Diverse SAT so far, using the PBCPGUIDE_100-VRANDGLOB_30 heuristic in Nadel (2011). Because of failing to obtain the source code, we implement Diverse$k$Set on the top of a SAT solver, Maple_LCM (Xiao et al. 2017). randAllSAT is a diverse SAT solver by invoking an AllSAT solver to generate k model, whose diversification is achieved by adopting a random branching heuristic. We also implemented randAllSAT, constructing it based on a backtracking-based AllSAT solver (Toda and Soh 2016). We did not modify the state-of-the-art solver tabularAllSAT (Spallitta, Sebastiani, and Biere 2024), because it enumerates partial assignments and relies on the ordering for enumeration, which is not easy to adapt to Diverse SAT problem.

Our proposed algorithm, *Diver*SAT, and two competitors are all implemented in C++ and compiled by g++ with '-O3' option. According to our preliminary experiments for parameter tuning, the parameters in *Diver*SAT are set as follows: $pr = 0.15$, $\lambda_1 = 100$, $\lambda_2 = 1$, $m = 50$, $r = 0.7$, $max\_step = 2 \times 10^5$, and $max\_cnt = 20$. All experiments were conducted on a server equipped with Intel(R) Xeon(R) 2.20 GHz CPU and 10 GB of memory, running CentOS 6.10. For each instance, all solvers were executed 10 times with a cutoff time of 3600 CPU seconds due to the randomization components. The parameter for the number of models k was set to 10, 50, and 100, consistent with the settings introduced in Nadel (2011). In all experiments, the unit of the runtime is expressed in seconds.

Experimental Results

Part 1: Table 2 illustrates the comparison results of *Diver*SAT with two competitors, Diverse$k$Set and randAllSAT. For each type of problem, we report the average diversity quality $DQ(S)$ and the average runtime (shown in brackets) for each parameter value of k over 10 runs, where the best values are marked in bold. The results show that *Diver*SAT obtains solutions with higher diversity quality than the other two algorithms on all instances, although it consumes more time. More encouragingly, as k increases, *Diver*SAT is also able to produce good-quality solutions. In particular, the performance of *Diver*SAT is surprisingly good for large-scale instances. For example, for bmc and hardware problems with large average numbers of variables and clauses, *Diver*SAT achieves much better solutions than the other two algorithms, which indicates that *Diver*SAT has good scalability.

Part 2: In this subsection, we conduct ablation experiments to investigate the role of the key strategies proposed in our algorithm. Since the initialization phase of our algorithm *Diver*SAT adopts three heuristics (variable decision heuristic, polarity selection heuristic, and SCCA heuristic) and the optimization phase employs an adaptive perturbation mechanism, we compare *Diver*SAT with two alternative versions: *Diver*SAT$\backslash H$ and *Diver*SAT$\backslash M$. *Diver*SAT$\backslash H$

Problem	$k = 10$			$k = 50$			$k = 100$		
	*Diver*SAT\M	*Diver*SAT\H	*Diver*SAT	*Diver*SAT\M	*Diver*SAT\H	*Diver*SAT	*Diver*SAT\M	*Diver*SAT\H	*Diver*SAT
ais	1588	1461	**1817**	46253	43317	**47012**	172917	162967	**188194**
bmc	30102	28886	**31917**	1710656	1684097	**1712108**	2714344	2674788	**2714649**
flat30-60	1959	1893	**1978**	49959	49719	**49974**	199942	198836	**199954**
flat50-115	3290	3258	**3300**	83244	82994	**83252**	333171	332247	**333216**
flat75-180	4902	4811	**4942**	124849	124788	**124878**	499844	498423	**499868**
flat100-239	6496	6482	**6590**	166367	166374	**166488**	666336	665195	**666402**
flat125-301	8117	8089	**8229**	208008	207980	**208090**	832794	831751	**832943**
flat150-360	9833	9679	**9874**	249705	249632	**249738**	999626	998633	**999654**
flat175-417	10819	10940	**11514**	290628	288290	**291311**	1164987	1162072	**1166018**
flat200-479	12166	12155	**13168**	332178	320153	**332980**	1332203	1326713	**1332922**
GCP	21129	20267	**23478**	580087	573287	**581966**	2330051	2220147	**2331740**
hardware	2916232	2838425	**2988763**	69190888	68193144	**70723412**	251325612	254196325	**270244214**
II	9453	9212	**10547**	220307	214885	**253324**	882386	872151	**997079**
logistics	9686	9818	**10758**	250506	256421	**275876**	1076090	1059449	**1099846**
qg	5095	5096	**5250**	104124	103388	**136993**	518716	515376	**545680**
ssa	27431	27491	**28118**	707005	708323	**708355**	2834638	2692826	**2834851**
SW100-8	7811	7789	**7904**	199749	199681	**199826**	799681	794265	**799765**

Table 3: Experimental results of three solvers on all benchmarks, comparing the average diversity of solutions.

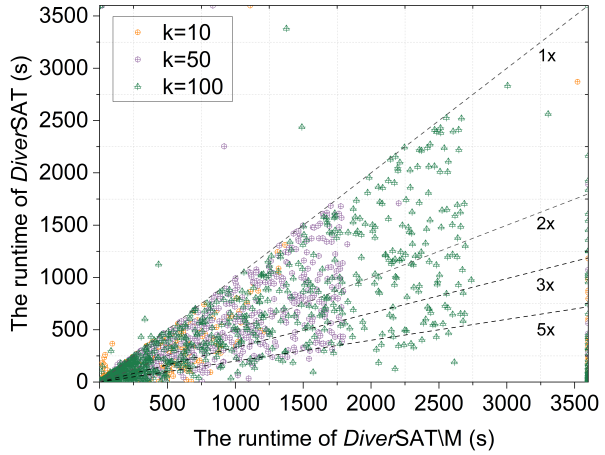

Figure 1: Comparing the runtime of *Diver*SAT against *Diver*SAT\M on all benchmarks.

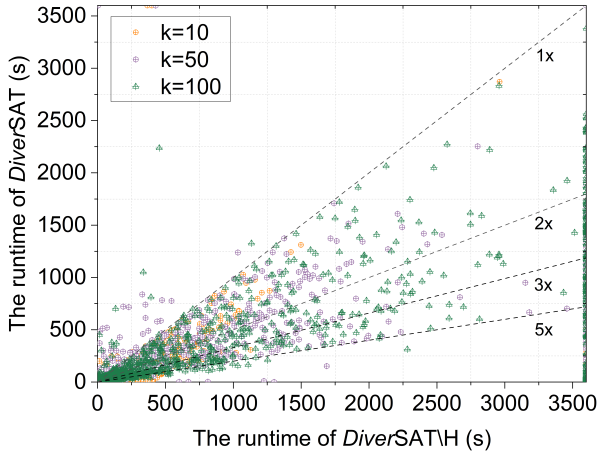

Figure 2: Comparing the runtime of *Diver*SAT against *Diver*SAT\H on all benchmarks.

refers to *Diver*SAT without three heuristics, but with VSIDS and CCA heuristic. *Diver*SAT\$M$ refers to the version where the adaptive perturbation mechanism is removed, meaning it only replaces the worst model. The results are displayed in Figure 1, Figure 2, and Table 3, where Figure 1 and Figure 2 plot the average runtime required by the three versions of *Diver*SAT to achieve the best solution over 10 runs for each instance, Table 3 records the average diversity quality for each value of the parameter k over 10 runs. As seen from Figure 1 and 2, *Diver*SAT runs faster than both of the other two versions on most instances, which declares that the core strategies used in *Diver*SAT do not significantly increase the runtime while ensuring diversity of the solutions. According to Table 3, *Diver*SAT achieves the best results in terms of solution quality on all instances. In addition, the solutions obtained by *Diver*SAT\$H$ do not always show a great advantage over those obtained by *Diver*SAT\$M$, which shows *Diver*SAT\$H$ and *Diver*SAT\$M$ have their own advantages in different types of problems. Moreover, with the varying of k, their advantages are stable. These results indicate the effectiveness of the three heuristics and the adaptive perturbation mechanism, each of which has different strengths and they complement each other well.

Conclusion

In this work, we propose a novel and effective local search algorithm, *Diver*SAT, for the Diverse SAT problem. *Diver*SAT incorporates three novel heuristics: the variable decision heuristic, the polarity selection heuristic, and the SCCA heuristic, as well as a new adaptive perturbation mechanism to enhance diversity. The experiments clearly present that *Diver*SAT achieves better solutions than the existing algorithms. Moreover, the core techniques are confirmed to be effective. In the future, we will aim for clearer directions for model generation and faster convergence to better solutions.

Acknowledgements

We would like to thank all the anonymous reviewers for their helpful comments, and we also appreciate Dr. Alexander Nadel's generous assistance. This work is supported by Science and Technology Development Program of Jilin Province under Grants 20230101060JC and YDZJ202201ZYTS412, NSFC under Grant No.61976050, Science and Technology Department Project of under Grant 20240602005RC, and Philosophy and Social Science Project under Grant 2023ZD15.

References

Agbaria, S.; Carmi, D.; Cohen, O.; Korchemny, D.; Lifshits, M.; and Nadel, A. 2010. SAT-based semiformal verification of hardware. In *Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2010)*, 25–32. IEEE.

Bailleux, O.; and Marquis, P. 1999. DISTANCE-SAT: Complexity and Algorithms. In *Proceedings of the 16th National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, 642–647. AAAI Press.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2021. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Cai, S. 2015. Balance between Complexity and Quality: Local Search for Minimum Vertex Cover in Massive Graphs. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 747–753.

Cai, S.; Luo, C.; and Su, K. 2015. CCAnr: A Configuration Checking Based Local Search Solver for Non-random Satisfiability. In *Proceedings of the 18th Theory and Applications of Satisfiability Testing (SAT 2015)*, volume 9340 of *Lecture Notes in Computer Science*, 1–8. Springer.

Cai, S.; and Su, K. 2012. Configuration Checking with Aspiration in Local Search for SAT. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 434–440. AAAI Press.

Cai, S.; and Zhang, X. 2021. Deep Cooperation of CDCL and Local Search for SAT. In Li, C.; and Manyà, F., eds., *Proceedings of the 24th International Conference on Theory and Applications of Satisfiability Testing, SAT 2021*, volume 12831 of *Lecture Notes in Computer Science*, 64–81. Springer.

Chakraborty, S.; Fremont, D. J.; Meel, K. S.; Seshia, S. A.; and Vardi, M. Y. 2015. On Parallel Scalable Uniform SAT Witness Generation. In Baier, C.; and Tinelli, C., eds., *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2015*, volume 9035 of *Lecture Notes in Computer Science*, 304–319. Springer.

Crescenzi, P.; and Rossi, G. 2002. On the Hamming distance of constraint satisfaction problems. *Theoretical Computer Science*, 288(1): 85–100.

Dahllöf, V. 2005. Algorithms for Max Hamming Exact Satisfiability. In Deng, X.; and Du, D., eds., *Proceedings of the 16th International Symposium on Algorithms and Computation, ISAAC 2005*, volume 3827 of *Lecture Notes in Computer Science*, 829–838. Springer.

Devroey, X.; Perrouin, G.; Legay, A.; Schobbens, P.; and Heymans, P. 2016. Search-based Similarity-driven Behavioural SPL Testing. In *Proceedings of the 10th International Workshop on Variability Modelling of Software-intensive Systems*, 89–96. ACM.

Dutra, R.; Laeufer, K.; Bachrach, J.; and Sen, K. 2018. Efficient sampling of SAT solutions for testing. In Chaudron, M.; Crnkovic, I.; Chechik, M.; and Harman, M., eds., *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018*, 549–559. ACM.

Fu, H.; Wu, G.; Liu, J.; and Xu, Y. 2021. More efficient stochastic local search for satisfiability. *Appl. Intell.*, 51(6): 3996–4015.

Ghasemi, M.; Crafts, E. S.; Zhao, B.; and Topcu, U. 2021. Multiple Plans are Better than One: Diverse Stochastic Planning. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling, (ICAPS 2021)*, 140–148. AAAI Press.

Hanaka, T.; Kiyomi, M.; Kobayashi, Y.; Kobayashi, Y.; Kurita, K.; and Otachi, Y. 2023. A Framework to Design Approximation Algorithms for Finding Diverse Solutions in Combinatorial Problems. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI 2023)*, 3968–3976. AAAI Press.

Hoi, G.; and Stephan, F. 2019. Measure and Conquer for Max Hamming Distance XSAT. In Lu, P.; and Zhang, G., eds., *Proceedings of the 30th International Symposium on Algorithms and Computation, ISAAC 2019*, volume 149 of *LIPIcs*, 15:1–15:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Misra, N.; Mittal, H.; and Rai, A. 2024. On the Parameterized Complexity of Diverse SAT. In Mestre, J.; and Wirth, A., eds., *Proceedings of the 35th International Symposium on Algorithms and Computation, ISAAC 2024*, volume 322 of *LIPIcs*, 50:1–50:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, 530–535. ACM.

Nadel, A. 2011. Generating Diverse Solutions in SAT. In *Proceedings of the 14th International ConferenceTheory and Applications of Satisfiability Testing (SAT 2011)*, volume 6695 of *Lecture Notes in Computer Science*, 287–301. Springer.

Petit, T.; and Trapp, A. C. 2015. Finding Diverse Solutions of High Quality to Constraint Optimization Problems. In Yang, Q.; and Wooldridge, M. J., eds., *Proceedings of the 34th International Joint Conference on Artificial Intelligence, IJCAI 2015*, 260–267.

Spallitta, G.; Sebastiani, R.; and Biere, A. 2024. Disjoint Partial Enumeration without Blocking Clauses. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI 2024)*, 8126–8135. AAAI Press.

Toda, T.; and Soh, T. 2016. Implementing Efficient All Solutions SAT Solvers. *ACM J. Exp. Algorithmics*, 21(1): 1.12:1–1.12:44.

Wu, J.; Li, C.; Jiang, L.; Zhou, J.; and Yin, M. 2020. Local search for diversified Top-*k* clique search problem. *Comput. Oper. Res.*, 116: 104867.

Wu, J.; Li, C.; Wang, L.; Hu, S.; Zhao, P.; and Yin, M. 2023. On solving simplified diversified top-k s-plex problem. *Comput. Oper. Res.*, 153: 106187.

Xiao, F.; Luo, M.; Li, C.-M.; Manyá, F.; and Lü, Z. 2017. MapleLRB_LCM, Maple_LCM, Maple_LCM_Dist, MapleLRB_LCMoccRestart and Glucose-3.0+width in SAT Competition 2017. In Balyo, T.; Heule, M. J. H.; and Järvisalo, M., eds., *Proceedings of SAT COMPETITION 2017: Solver and Benchmark Descriptions*, volume B-2017-1 of *Department of Computer Science Series of Publications B*, 22–23.

Zhou, J.; Liang, J.; Yin, M.; and He, B. 2023. LS-DTKMS: A Local Search Algorithm for Diversified Top-k MaxSAT Problem. In *Proceedings of the 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023)*, volume 271 of *LIPIcs*, 29:1–29:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.