IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 20, 2021, accepted March 21, 2021, date of publication March 24, 2021, date of current version April 6, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3068824

A Stochastic Local Search Algorithm for the Partial Max-SAT Problem Based on Adaptive Tuning and Variable Depth Neighborhood Search

HAIFA HAMAD ALKASEM AND MOHAMED EL BACHIR MENAI
Department of Computer Science, College of Computer and Information Sciences, King Saud University, Riyadh 11375, Saudi Arabia

Corresponding author: Haifa Hamad Alkasem (hhkasem@imamu.edu.sa)

This work was supported by the Deanship of Scientific Research through the initiative of DSR Graduate Student Research Support (GSR), King Saud University.

ABSTRACT The Partial Max-SAT (PMSAT) problem is an optimization variant of the well-known Propositional Boolean Satisfiability (SAT) problem. It holds an important place in theory and practice, because a huge number of real-world problems, such as timetabling, planning, routing, bioinformatics, fault diagnosis, etc., could be encoded into it. Stochastic local search (SLS) methods can solve many real-world problems that often involve large-scale instances at reasonable computation costs while delivering good-quality solutions. In this work, we propose a novel SLS algorithm called *adaptive variable depth SLS* for PMSAT problem solving based on a dynamic local search framework. Our algorithm exploits two algorithmic components of an SLS method: parameter tuning and neighborhood search. Our first contribution is the design of an adaptive parameter tuner that searches for the best parameter setting for each instance by considering its features. The second contribution is a variable depth neighborhood search (VDS) algorithm adopted for PMSAT problem, which our empirical evaluation proves is a more efficient w.r.t. single neighborhood search. We conducted our experiments on the PMSAT benchmarks from MaxSAT Evaluation 2014 to 2019, including more than 3600 instances which have been encoded from a broad range of domains such as verification, optimization, graph theory, automated-reasoning, pseudo Boolean, etc. Our experimental evaluation results show that AVD-SLS solver, which is implemented based on our algorithm, outperforms state-of-the-art PMSAT SLS solvers in most benchmark classes, including random, crafted, and industrial instances. Furthermore, AVD-SLS reports remarkably better results on weighted benchmark, and shows competitive results with several well-known hybrid PMSAT solvers.

INDEX TERMS Partial Max-SAT, adaptive parameter tuning, variable depth search, stochastic local search, Max-SAT evaluation.

I. INTRODUCTION

Partial Max-SAT (PMSAT) problem is an optimization variant of Propositional Boolean Satisfiability (SAT) problem, which is a fundamental problem in computer science and artificial intelligence [1], [2]. PMSAT problem is an NP-hrd problem that is important for the theory and practice of a range of applications, including timetabling [3], scheduling [4], planning [5], routing [6], software debugging [7], and bioinformatics [8]. Actually, many optimization problems can be naturally expressed as a PMSAT problem. PMSAT asks to find an assignment to the Boolean variables of a given Boolean formula expressed in the Conjunctive Normal Form (CNF), which satisfies all hard (mandatory) clauses and the maximum number of soft (non-mandatory) clauses. Maximum Boolean Satisfiability (Max-SAT) problem is a specialization of PMSAT problem, where all clauses are soft and the goal is to satisfy the maximum number of clauses. PMSAT is a designation given to Max-SAT problem with hard and soft clauses in 1996 by Miyazaki *et al.* [9].

There are two state-of-the-art approaches for solving PMSAT problem: exact methods and stochastic local search (SLS) methods. There are also hybrid methods that combined both exact and SLS methods [10]–[17]. Exact methods

The associate editor coordinating the review of this manuscript and approving it for publication was Xujie Li.

(also known as complete methods) implicitly enumerate all the solutions of the considered instance of an optimization problem using a search tree to explore the entire search space and to prove the satisfiability. Recently, almost all Max-SAT and PMSAT solvers that participated in MaxSAT Evaluation (MSE) are SAT-based [11], [16], [18]–[21], where a given problem instance is solved through successive calls to a SAT solver. Exact methods can provide optimal solutions to small or medium-sized problems with reasonable computational costs. However, many real-world applications, especially engineering and industrial applications, often involve far larger scales which exact methods cannot handle, hence the need for SLS methods [22]–[26].

SLS methods (also known as incomplete methods) have become more popular because of their ability to provide high-quality solutions to large-size problems with reasonable computation costs. An SLS is a local search method that incorporates a stochastic (i.e. randomness) property. Local search methods are general methods that are widely used to solve hard combinatorial optimization problems [27]–[31]. A local search method is defined by four main components: search space, neighborhood relation, objective function, and move method [30]. Each component may have one or more parameters that determine its functioning [32]. In a local search method for a propositional Boolean problem, and starting from a complete assignment, a neighborhood solution is obtained by flipping the truth value of one variable ($1-flip$) or a small set of variables ($k-flip$). At each step, the neighborhood is examined for a truth assignment that decreases the number (or total weight) of unsatisfied clauses. If such an assignment is found, the algorithm flips the value of the corresponding variable (or set of variables), and continues the search until a stopping criterion is encountered [33], [34].

Our investigation shows that there are three main state-of-the-art PMSAT SLS-based methods: distinction-based method [35], configuration checking-based method [36], and dynamic local search [37] method. These SLS methods are built around two algorithmic components: a variable-pick heuristic and a weighting scheme. However, compared to the breakthrough progress of SLS methods on random and crafted benchmark instances, the performance of SLS methods on industrial benchmark instances lags far behind, especially on weighted industrial benchmark instances. Those instances are often large-sized instances that involve huge neighborhood size or highly complex structures.

In this paper, we introduce a novel SLS algorithm named adaptive variable depth SLS, that employs an extended framework of dynamic local search method. We chose the dynamic local search SLS method, because it showed competitive performance on unweighted industrial benchmarks in MSE 2018. Our method is based on the results of our study of the state-of-the-art PMSAT SLS methods' strengths and limitations. A problem of fundamental interest and practical importance is how to exploit SLS method components with respect to constraints in order to manage high complexities and improve algorithm performance [38]. In this work, we propose a novel method based on two components of an SLS method: parameter tuning and neighborhood search.

First, we propose an adaptive parameter tuner that searches for the best possible parameter values per instance. We studied the relationship between parameter setting and features of input instances, and found that parameter setting is related to some features of input instance. Second, we propose a variable depth neighborhood search (VDS) method [39] adopted for PMSAT SLS to explore very large neighborhoods. The VDS method defines a dynamically determined number of sequence of moves that varies from one iteration to another (i.e. variable $k-flips$), where for each step leading to a different trial solution, the compound move that yields the best trial solution is the one chosen [40]. Almost all SLS methods proposed for solving PMSAT problem are based on single neighborhood definition and single ($1-flip$) move, whose performance declines when exploring huge neighborhoods for large-scale problem instances (i.e. industrial instances).

To evaluate the performance of our novel algorithm, we implemented the AVD-SLS solver. We compared AVD-SLS with the state-of-the-art PMSAT SLS solvers and PMSAT hybrid solvers that participated in the MSE 2014-2019, on a broad range of random, crafted and industrial benchmarks. AVD-SLS reported the best results compared with all of the PMSAT SLS solvers on weighted crafted and industrial benchmark instances. Compared to PMSAT hybrid solvers, AVD-SLS shows a competitive performance. Based on our experimental evaluation study, AVD-SLS ranked among the top three best PMSAT solvers for MSE 2015, 2016, 2017, 2018 on the weighted benchmark, and was also among the top three best PMSAT solvers for MSE 2014 and 2017 on the unweighted benchmark.

The reminder of this paper is organized as follows. The next section introduces preliminary knowledge. The related work are presented in Section III. Then, we introduce our AVD-SLS method in details in Section IV. The experimental evaluation study is presented in Section V. Finally, we elaborate on the results and conclude the paper in Section VI and Section VII respectively.

II. PRELIMINARIES

In this section, we briefly introduce the main notations, definitions, and background knowledge.

A Propositional Variable v_i is a Boolean variable assigned a truth value (*true* or *false*).

A Literal l_i is a propositional variable (v_i) or its negation ($\neg v_i$).

Propositional Operators are logical connectives defined as a set of three operators $\{\neg, \wedge, \vee\}$ that represent the negation, conjunction, and disjunction operators respectively.

A Clause c_j is a disjunction of literals defined by $\bigvee_{i=1}^{k} l_i$ (e.g., $c_j = l_1 \vee l_2 \ldots \vee l_i \vee l_k$), where k represents the number of literals in each clause. A clause c_j is said to be satisfied if c_j evaluates to *true*, such that at least one literal $l_i \in c_j$ is assigned to *true*; otherwise c_j is said to be unsatisfied.

If c_j is a **hard (mandatory) clause**, then it must be satisfied. Otherwise, c_j is a **soft clause** that can be satisfied.

A Weighted Clause is a pair (c_j, w_j), where c_j is a clause and w_j is an associated positive number that represents the cost of c_j unsatisfaction.

A Unit Clause c_j is a clause that contains only one literal.

A Conjunctive Normal Form (CNF) formula F consists of conjunction of clauses defined by $\bigwedge_{j=1}^{m} c_j$ (e.g., $F = c_1 \wedge c_2 \ldots \wedge c_j \wedge c_m$), where m represents the number of clauses in the CNF formula F.

The Set of All Variables appearing in F is defined by $V(F) = \{v_1, v_2, \ldots, v_i, \ldots, v_n\}$, where n represents the number of variables in F.

An Assignment α of a given CNF formula F sets the truth value of each variable $v_i \in V(F) : value(v_i) \in \{true, false\}$.

The Cost of an assignment α is defined by $cost(\alpha)$, that denotes the number (or total weight) of the unsatisfied clauses under the current assignment α. We say that α_1 is better than α_2 if $cost(\alpha_1) < cost(\alpha_2)$.

The Score of a variable v_i is defined by $score(v_i)$, that denotes the increment of the number (or total weight) of satisfied clauses by flipping the truth value of a selected variable v_i from *true* to *false* or vice versa. The $score(v_i) = make(v_i) - break(v_i)$. The property $make(v_i)$ is defined as the total number (or total weight) of clauses that would become satisfied if variable v_i is flipped and the $break(v_i)$ is the total number (or total weight) of clauses that would become unsatisfied if variable v_i is flipped.

The Hard Score of a variable v_i is defined by $hscore(v_i)$, that denotes the increment of the number (or total weight) of satisfied hard clauses by flipping the truth value of a selected variable v_i.

The Soft Score of a variable v_i is defined by $sscore(v_i)$, that denotes the increment of the number (or total weight) of satisfied soft clauses by flipping the truth value of a selected variable v_i.

A Decreasing Variable v_i denotes that the $score(v_i) > 0$; if the $score(v_i) < 0$ then v_i is said to be an increasing variable. $hscore(v_i) > 0$ denotes a hard decreasing variable v_i and $sscore(v_i) > 0$ denotes a soft decreasing variable v_i.

A Non-decreasing Variable v_i, defined as $0 - score$, denotes that the number (or total weight) of unsatisfied clauses is not changed by flipping the truth value of a selected variable v_i. $0{-}hscore$ denotes a non-decreasing hard variable v_i.

The Neighborhood of a variable v_i is defined by $N(v_i) = \{v_j \in N(v_i) : i \neq j\}$, which denotes the set of all neighboring variables of $v_i \in V(F)$. Two different variables are neighbors if and only if they appear in at least one clause simultaneously. $HN(v_i) = \{v_j \in HN(v_i) : i \neq j\}$ denotes the set of all hard neighboring variables of $v_i \in V(F)$. Two different variables are hard neighbors if and only if they appear in at least one hard clause simultaneously.

A Configuration Checking of a variable v_i is the state that indicates whether any variable $v_j \in N(v_i) : i \neq j$ has been selected and flipped since v_i was last flipped. If at least one variable $v_j \in N(v_i) : i \neq j$ has been flipped since v_i was last flipped, then the configuration of v_i is changed and $configuration(v_i) = 1$; otherwise, $configuration(v_i) = 0$.

(Weighted) Partial Maximum Boolean Satisfiability (PMSAT) problem is to find an assignment of truth values to $V(F)$ that satisfies a partial set of clauses, called hard clauses, and minimizes the number (or weight) of unsatisfied soft clause. An assignment α for the PMSAT problem is feasible, if and only if, it satisfies all hard clauses. Minimizing (maximizing) the number of unsatisfied (satisfied) soft clauses is a measure of solution quality.

(Weighted) Maximum Boolean Satisfiability (MaxSAT) problem is to find an assignment of truth values to $V(F)$ that minimizes (maximizes) the number (weight) of unsatisfied (satisfied) clauses.

III. RELATED WORK

Our investigation shows that there are three main state-of-the-art SLS-based methods for PMSAT problem solving: distinction-based [35] method, configuration checking-based [36] method, and dynamic local search method [37]. These SLS methods are built around two algorithmic components: a variable-pick heuristic and a weighting scheme. SLS methods differ in the variable-pick heuristic selected as they employ priority-based variable-pick heuristics differently based on the scores of variables. Clause weighting is a weighting scheme that make SLS methods dynamic [41]. The weights are adjusted during the search to determine the search trajectory while maintaining reasonable weight differentials [42]. The following paragraphs review each of state-of-the-art SLS methods.

The distinction-based method was proposed by Cai *et al.* in 2014 [35], which distinguishes between hard and soft clauses by maintaining a weighting scheme for hard clauses only. The weighting scheme for hard clauses is called the hard pure additive weighting scheme (HPAWS), that was inspired by the pure additive weighting scheme (PAWS) algorithm proposed by Thornton *et al.* [41] to solve SAT problem. PAWS is a dynamic local search algorithm that has been used in many SLS algorithms [43]–[46]. Whenever the search stagnates and no improving neighborhood solutions are found, the weights of the unsatisfied (hard) clauses are adjusted to help escape from the local minimum. Additive weighting schemes show better scalability than other weighting schemes such as DLM [29] and SAPS [47]. HPAWS incorporates a diversification probability, called the smoothing parameter (sp is a real number that controls the hard clause weights), to decide whether to increase the weights of unsatisfied hard clauses or to decrease the weights of satisfied hard clauses.

Based on the concept of separation between hard and soft scores, the distinction-based method uses a multi-level priority-based approach for the variable-pick heuristic. Hard decreasing variables ($hscore > 0$) have the highest priority, and the hard variable with the greatest hard score may be selected. In the second priority level come the non-decreasing hard variables ($0 - hscore$) that are also soft decreasing (*sscore* > 0), and the variable with the greatest soft score may be selected. The third level indicates that the search stagnates and two actions take place. First, the weights of the hard clauses are updated according to HPAWS. Second, a random unsatisfied hard clause, if any, is chosen; otherwise, from a randomly selected unsatisfied soft clause, either a random variable v or the variable with the greatest soft score is selected based on a given random walk probability wp (a real number controls the activation of the random walk heuristic). The distinction-based is designed with a simple neighborhood definition and 1-flip move at each step of the search.

There are six PMSAT SLS solvers based on the distinction method: Dist [35], [48], HS-Greedy,[1] Dist1,[2] Dist2,[3] Dist-r,[4] and DistUP [48]. The parameters were tuned either with an automatic offline parameter tuning tool [42] or manually based on researcher's experience. Furthermore, some solvers adopted the Best from Multiple Selections (BMS) heuristic proposed in [49]. Generally, BMS is a probabilistic sampling method that selects randomly t candidate decreasing variables and returns the best one. The source code of distinction-based solvers are not available online.

The configuration checking-based method was first proposed by Cai *et al.* in 2011 [50]. The configuration checking-based method, similar to the distinction-based method, uses HPAWS for hard clauses weighting scheme and maintains separate scores for hard and soft clauses. The variable-pick heuristic is based on the configuration checking property of variables. The configuration checking property serves two purposes: to avoid cycling and to promote the diversification [50]. The configuration checking property is inspired by the tabu search mechanism [36], [50].

The configuration checking-based method starts the search with a probability p, to decide on whether to perform a random move that is biased towards the satisfaction of hard clauses or to switch to a greedy mode. In this mode, the configuration checking-based method uses a multi-level priority-based approach for the variable-pick heuristic. The highest priority is for the hard decreasing variables (*hscore* > 0) whose configuration changed, and the hard variable with the greatest hard score may be selected. If there are no hard decreasing variables, the weights of the hard clauses are updated according to HPAWS. Then, the decreasing variable (*score* > 0) with the greatest score whose configuration changed may be selected, if any. Otherwise, when the search stagnates, a variable v from a random unsatisfied hard clause c, if any, is chosen or from a randomly selected unsatisfied soft clause c. The configuration checking-based method is similar to the distinction-based method, as it implements a simple neighborhood definition and 1-flip move at each step of the search.

However, different priority-levels were adopted in literature [36], [51], [52]. For example the hard clause state-based configuration checking (HCSCC) [51] forbidding mechanism that is similar to the one proposed by Luo *et al.* [53], but it emphasizes on hard clauses with a configuration that consists of the states of hard clauses instead of neighboring variables (i.e., captures the global effect of flipping a selected variable v). This variable-pick heuristic, improves the results on some real-world application of wighted PMSAT problem instances [51].

There are six SLS solvers proposed to solve the PMSAT problem which are based on the configuration checking-based method: configuration checking local search (CCLS) [36], configuration checking with emphasis on hard clauses (CCEHC) [51], and hard neighboring variables with configuration checking (HNVCC) [52]. HNVCC is based on CCEHC with a new forbidding strategy [45] for hard variables only. Details of both solvers CCMPA[5] and SC2016[6] were not published. The parameters were tuned either with an automatic offline parameter tuning tool [42] or manually based on researcher's experience. Furthermore, some solvers adopted the Best from Multiple Selections (BMS) heuristic. And the source code of configuration checking-based solvers are not available online.

The dynamic local search method was proposed by Lei and Cai in 2018 [37]. Unlike the distinction-based and configuration checking-based methods, the dynamic local search method uses one weighting scheme (called Weighting-PMS) for both hard and soft clauses based on PAWS [41], but the increments of soft clauses' weights are limited by a maximum weight threshold ζ, to avoid favoring soft clauses over hard clauses, which may mislead the search. For the hard clauses, each hard clause is associated with value one for the weight that is then updated during the search by a constant amount (h_inc). Whereas the original input weights of the soft clauses are used. However, dynamic local search uses a simpler variable-pick heuristic. This variable-pick heuristic considers three break levels on updating the scores of the variables. SATLike [37], [54], [55] solver is based on this dynamic local search method and was the first SLS solver that competes on industrial benchmarks [56]. The parameters were tuned manually based on researcher's experience. Furthermore, the Best from Multiple Selections (BMS) heuristic was adopted by SATLike. The source code of SATLike solver is available online at MaxSAT Evaluation (MSE) 2018 website.

IV. ADAPTIVE VARIABLE DEPTH SLS ALGORITHM

The novel algorithm proposed in this work extends the state-of-the-art dynamic local search framework by incorporating two main components of an SLS method: parameter tuning and neighborhood search.

[1] HS-Greedy - Max-SAT 2016 - http://www.maxsat.udl.cat/16/solvers/index.html

[2] Dist1 - Max-SAT 2015 - http://www.maxsat.udl.cat/15/solvers/index.html

[3] Dist2 - Max-SAT 2015 - http://www.maxsat.udl.cat/15/solvers/index.html

[4] Dist-r - Max-SAT 2016 - http://www.maxsat.udl.cat/16/solvers/index.html

[5] CCMPA - Max-SAT 2014 - http://www.maxsat.udl.cat/14/results-incomplete/index.html

[6] SC2016 - Max-SAT 2016 - http://www.maxsat.udl.cat/16/solvers/6/README.txt-201603260957

In local search algorithms, even relatively minor deviations from optimal parameter settings (i.e., parameter values) can lead to a substantial increase in the expected time at which a given problem instance is solved; this sensitivity seems to increase with the size and hardness of an instance to be solved [57]. In PMSAT SLS solvers, two main methods are used to tune parameters: manual tuning (i.e., based on a researcher's experience) and automatic offline parameter tuning [58]. The former is confronted with a serious difficulty; that is, a parameter setting that results from the tuning of some problem instances may not be applicable to other (difficult) instances of the same class. This means that this process is time consuming. By contrast, automatic offline parameter tuning methods determine parameter settings on the grounds of a representative set of benchmark instances (i.e., per-set tuning) during a training phase before algorithm deployment. Generally, these methods entail significant computational cost and experimentation effort, but their outcomes are nevertheless often reusable in a wide range of similar problems [59]. Some PMSAT SLS solvers [48], [51], [52] have been incorporated with an automatic offline parameter tuner called the sequential model-based algorithm configuration (SMAC) [60].

In contrast to automatic offline methods, adaptive parameter tuning is aimed at dynamically establishing the parameter values of an algorithm on the basis of feedback derived during its run. Such tuning is easier to deploy because limited user intervention is required. However, automatic adaptive methods may suffer from two major drawbacks: overspecialization if developed for a specific algorithm or problem and the requirement for additional parameters, which may expand the parameter domain that also requires tuning [58], [61].

Several online tuners have been proposed in the literature. McAllester *et al.* [62] proposed a tuning method for six different variable-pick heuristics based on a WalkSAT algorithm to solve the SAT problem. They found that when the noise parameter p is tuned well for each heuristic, the performance will be approximately the same. They called this a *noise level invariant*. They suggested that given the best performance of a heuristic, p can be tuned for the other heuristics for a given measure of performance. The proposed tuning method is based on the statistical properties of the search: mean and variance of the violation count (i.e., *invariant ratio*). Their empirical study showed that the best parameter tuning method is related to both the problem instance and the underlying algorithm.

Patterson and Kautz [63] extended the work of McAllester *et al.* [62], and proposed the Auto-WalkSAT tuning method. This method is based on the standard deviation of the *invariant ratio* and Brents method [64]. The tuner searches the invariant ratio space by two mathematical methods: recursive bracketed and parabolic interpolation. Then, the optimized p-value is used during the search. The experimental evaluation showed that the best parameter tuning method is related to whether p should be decreased on increased during the tuning process.

Hoos [65] proposed a simple self-tuning noise mechanism for the noise parameter p. This adaptive noise mechanism was applied for different variants of a WalkSAT algorithm. The basic idea is to increase the value of p to high values when the search stagnates, leading to more diversification. The empirical results showed that in some cases, the method significantly improves the results in comparison to the basic WalkSAT. Similarly, the reactive scaling and probabilistic smoothing (RSAPS) method, proposed by Hutter *et al.* [47], is also based on a simple self-tuning noise mechanism, but for the smoothing parameter p_{smooth} that controls the weighting scheme. The experimental evaluation showed that the performance was better for the RSAPS method than the basic WalkSAT.

Two other research studies were based on Hoos's method [65]. Li *et al.* [66] proposed adaptG^{2}WSAT, which combines the adaptive noise tuning in [65] and Look-Ahead for a variable-pick heuristic to solve the SAT problem. The empirical results show that adaptG^{2}WSAT method significantly improved the results in comparison to G^{2}WSAT and other variants of the basic WalkSAT. Furthermore, the adaptive memory-based local search (AMLS) method, which Lu and Hao [67] proposed for solving the Max-SAT problem, employs a variable-pick heuristic based on a tabu mechanism. The parameters p and p_w are both adaptively tuned based on Hoos's method [65]. Moreover, the authors proposed an additional random perturbation operator for diversification. AMLS showed better results for some Max-SAT instances compared to the basic WalkSAT. However, the empirical evaluation showed that the initial setting of some other parameters may need to be better tuned.

Prestwich [68] proposed a reinforcement learning algorithm to automate the tuning process of the parameters using the average-reward method. The basic idea is to learn better noise levels against objective function values. That method considers the average progress towards a solution, which is called *gain optimality*. The experimental evaluation showed that this method achieves a better performance than other learning methods, such as Q-learning.

Based on the discussion presented above, we developed a novel adaptive parameter tuner (Section IV-A) that considers Hoos's tuning method [65] and a set of features for an input PMSAT problem instance for initial parameter setting selection.

How a neighborhood is defined and explored is a critical factor that affects the performance of local search algorithms. The larger the neighborhood, the better the local optima but the higher the computational cost. The key feature is to improve neighborhood search for large problem instances without explicitly evaluating all neighbors each time a search is trapped in a local minimum [69]. State-of-the-art large neighborhood search (LNS) methods have shown outstanding results in solving various problems from different domains. An LNS search method can explore complex neighborhoods and find better candidate solutions in each iteration, thereby guiding searches toward more promising search paths

[70], [71]. LNS methods are grouped into different categories, such as variable neighborhood search (VNS) and variable depth neighborhood search (VDS). In this work, we put forward a VDS algorithm for PMSAT problem solving. An AVD algorithm can search deeper neighborhoods in a heuristic way using only one parameter that controls the depth of the neighborhood search [39], [70]. Our proposed VDS algorithm is presented in Section IV-B.

A. ADAPTIVE PARAMETER TUNING

Building an adaptive parameter tuner for a PMSAT SLS algorithm is a challenging task for two main reasons: the time constraint involved and the number of parameters needed to be tuned. Our aim is to design a simple yet efficient tuning algorithm. In our proposed algorithm, we build an adaptive tuner for the underlying dynamic local search SLS solver to tune nine parameters. Four parameters are used in the weighting scheme, two random walk parameters used by a variable pick heuristic, one parameter for the BMS heuristic, one parameter that controls the depth of the VDS algorithm, and one parameter used by the search algorithm for diversification.

We have studied the relationship between the parameters' values and some features of input instances on the basis of a subset of weighted and unweighted benchmarks from MSE 2017 and 2018 (see Appendix A and Appendix B). For many instances, some parameter settings are more related to some features of an input instance. One of the most important features for our adaptive tuner is the hard ratio, which is the ratio of the number of hard clauses to the number of hard variables of an input PMSAT problem instance.

Fig. 1 shows the main steps of the proposed adaptive parameter tuner. Starting from initial default parameters' values and extracted features of the input PMSAT problem instance (F), the tuner calls the underlying solver to solve F and find a solution (α). The tuner then checks the quality of the output solution, and if a feasible solution is found, the tuner stops and the variable adaptive SLS search starts. Otherwise, the tuner will re-tune each parameter separately and recall the solver again until a feasible solution is found or the time limit is reached. In this re-tune step of each parameter, the tuner is looking for improved solutions and a parameter that results in a better solution is set as a sensitive parameter of F and added to the list of sensitive parameters for future recall of adaptive parameter tuner.

B. VARIABLE DEPTH NEIGHBORHOOD SEARCH (VDS)

As presented in Section III, state-of-the-art PMSAT SLS algorithms rely on a single neighborhood search with single ($1 - flip$) move in each iteration.However, this technique fails to explore huge neighborhoods when solving large-sized PMSAT problem instances. To tackle this limitation, we propose a variable depth neighborhood search (VDS) [39] adopted for PMSAT problem solving. To the best of our knowledge, this is the first research work adopted VDS for PMSAT problem solving.

Algorithm 1 VDS

```
Input: $\alpha*$, a selected variable $v_i$ by adaptive variable depth
    SLS algorithm
Output: return $\alpha*$, $cost*$
    // let $\alpha*$ the best solution found so far
    // let $cost$ the number (or total weight) of unsatisfied
    clauses of $\alpha$ and $cost*$ is the cost of $\alpha*$
    // let $BEST$ a list of decreasing neighborhood variables
1:  $BEST \leftarrow v_i$
2:  while $depth > 0$ do
3:    if $BEST \neq \phi$ then
4:      $v_{best} \leftarrow$ remove a random variable from $BEST$
5:      $\alpha \leftarrow \alpha$ with $v_{best}$ flipped
6:      $BEST \leftarrow BEST \bigcup \{v_j \in N(v_{best}) : score(v_j) >$
        $0 \; and \; best \neq j\}$
7:    end if
8:    $depth \leftarrow depth - 1$
9:  end while
10: return $\alpha*$, $cost*$
```

The VDS (Algorithm 1) maintains a list called *BEST* that keeps deceasing neighborhood variables [line 1] of a selected variable v_{best}. Starting with a selected variable v_i by adative variable depth SLS algorithm (Algorithm 2), VDS flips the selected variable and then adds to the best list all of its decreasing neighborhood variables [lines 4-6]. In the next iteration, a random variable from the best list is selected, flipped, and then removed from the best list and all of its decreasing neighborhood variables are addded to the best list. The process is repeated until the depth is reached [line 2] or the best list becomes empty [line 3]. Finally, VDS returns the best solution ($\alpha*$) found. We believe that our proposed VDS algorithm works simply and is naturally similar to Distinction-based method and Configuration Checking-based method which maintain a list of hard decreasing variables, but without using any additional data structures for VDS. Algorithm 1 has a time complexity of $O(depth)$ and a memory complexity of $O(depth)$.

C. ADAPTIVE VARIABLE DEPTH SLS ALGORITHM

The complete algorithm of adaptive variable depth SLS is presented in this section. Our novel algorithm extends the framework of state-of-the-art dynamic local search algorithm SATLike [37] by using two main components: adaptive parameter tuning and variable depth search (VDS). The adaptive variable depth SLS (Algorithm 2) works as follows. First, an initial assignment α is randomly generated with unit propagation-based decimation preprocessing [line 1]. Second, the adaptive parameter tuner is activated [lines 8-9] before the search begin as described in section IV-A.

Then the search begins; based on the variable-pick heuristic, a decreasing variable v is selected, if any. Otherwise, the weighing scheme is activated, and the best variable v is chosen from c a randomly selected unsatisfied hard clause, if any. If not, it is chosen from a randomly selected unsatisfied

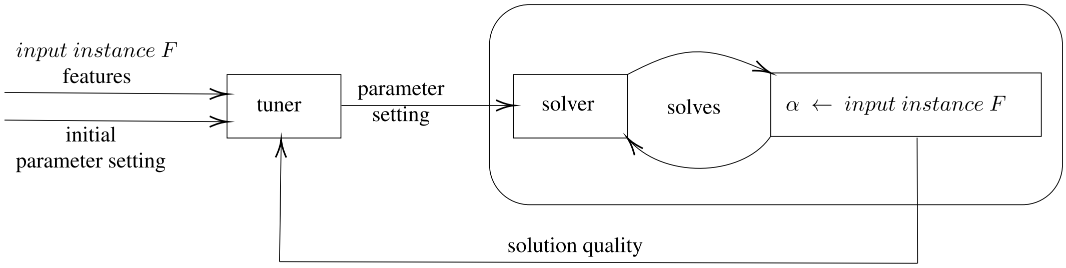

FIGURE 1. Adaptive parameters tuning.

TABLE 1. Execution environment for MSE 2014-2019.

MaxSAT Evaluation	Execution Environment
2014-2016	Operating System: CentOS release 6.3 - 2.6.32 x86_64 GNU/Linux Processor: Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz Memory: 3.5 GB Cache: 15360 KB Compilers: GCC 4.4.6, javac J2RE 2.3
2017-2019	Operating System: CentOS Linux release 7.7.1908 (Core) kernel Processor: Starexec nodes - Intel(R) Xeon(R) CPU E5-2609 0 @ 2.40GHz (2393 MHZ) Memory: 263932744 kB Cache: 10240 KB Compilers: glibc: glibc-2.17-292.el7.x86_64 gcc-4.8.5-39.el7.x86_64 glibc-2.17-292.el7.i686

TABLE 2. PMS classes and statistics for MSE 2014-2019.

		Benchmark		
	random	crafted	industrial	
Year	#inst. (#families)	#inst. (#families)	#inst. (#families)	Total
2014	210 (7)	421 (10)	568 (22)	**1199 (39)**
2015 2016	210 (7)	678 (13)	601 (23)	**1489 (43)**
2017	-	194 (24)		**194 (24)**
2018	-	153 (23)		**153 (23)**
2019	-	299 (38)		**299 (38)**

soft clause [lines 11-21]. The selected variable v is then sent to VDS (Algorithm 1) [line 22]. VDS explores the (huge) neighborhood deeply with a variable number of moves ($k - flips$) at each iteration as described in Section IV-B. If no feasible solution is found before the *cutoff -time* is reached, the adaptive parameter tuner is re-activated to re-tune the parameters [lines 8-9]. Algorithm 2 has a time complexity of $O(N)$ and a memory complexity of $O(N + C)$, where N represents the total number of variables, and C represents the total number of clauses.

V. EXPERIMENTAL EVALUATION OF ADAPTIVE VARIABLE DEPTH-BASED SLS SOLVER

In this section, we empirically evaluate our algorithm presented in Section IV by implementing a PMSAT SLS solver that is based on it. The solver is called AVD-SLS, which is evaluated on both unweighted and weighted PMSAT benchmarks from MaxSAT Evaluation (MSE) 2014-2019. In this evaluation,, we refer to the unweighted PMSAT benchmark as PMS and to the weighted PMSAT benchmark as WPMS. The reported results are then compared to the results of the PMSAT solvers that participated in MSE 2014-2019. In this experimental evaluation, we do not consider the published results of state-of-the-art PMSAT SLS solvers because of two main reasons. The first reason is the inconsistency of published results. A solver was ran more than once on each benchmark with different results reported for each run, although all adopted the MSE rules including the "run once" rule, as described in Section V-A under Evaluation Methodology. Moreover, detailed results are not available for comparison with per-instance solution for the number of solved instances, best solutions found, and best family results.

Algorithm 2 Adaptive Variable Depth SLS

```
Input: PMSAT problem CNF instance $F$, initial parameter setting, cutoff-time
Output: return $\alpha*$, $cost*$ if $\alpha*$ is a feasible solution. Otherwise print "No solution found"
    // let $\alpha*$ the best solution found so far
    // let $cost*$ the number (or total weight) of unsatisfied clauses of $\alpha*$
$\alpha \leftarrow$ a randomly-generated complete initial assignment of $F$ with UP-based decimation preprocessing
$\alpha* \leftarrow \alpha$
$cost* \leftarrow +\infty$
while elapsed time < cutoff-time do
    if $\nexists$ unsatisfied hard clauses and $cost(\alpha) < cost*$ then
        $\alpha* \leftarrow \alpha$, $cost* = cost(\alpha)$
    end if
    if tuner activated then
        new parameter setting $\leftarrow$ tuner(current parameter setting, $\alpha*$)
    end if
    if $\exists$ decreasing variables then
        $v \leftarrow$ a variable with the greatest score; breaking ties for the one that is least recently flipped
    else
        update weights of clauses
        if $\exists$ unsatisfied hard clauses then
            $c \leftarrow$ a random-seleted unsatisfied hard clause
        else
            $c \leftarrow$ a random-seleted unsatisfied soft clause
        end if
        $v \leftarrow$ the variable from c with greatest score; breaking ties for the one that is least recently flipped
    end if
    VDS($\alpha*$, $v$);
end while
return $\alpha*$, $cost*$
```

The second reason is that the state-of-the-art SLS solvers that participated in MSE 2014–2019 showed better results than the published ones.

A. EXPERIMENTAL SETUP

Our experiments were conducted on Shaheen, a supercomputer consisting of a 36 rack Cray XC40 system. The front-end environment is running SUSE Linux Enterprise Server 15.The system has 6,174 dual sockets compute nodes based on 16 core Intel Haswell processors running at 2.3GHz. Each node has 128GB of DDR4 memory running at 2300MHz.[7] AVD-SLS solver was implemented in C++ and compiled by g++ with '−O3' option. The execution environment for MSE 2014-2019 benchmarks are shown in Table 1. The time to find the best solution is not considered here, since the processing time is machine-dependent. As such, it was not considered by the latest MSE since 2017.

1) BENCHMARKS

We consider in this experimental evaluation all the benchmarks from incomplete track of MSE 2014-2019 for two time limits 60 and 300 CPU seconds. The MSEs are affiliated events of the International Conference on Theory and Applications of Satisfiability Testing (SAT) that is held every year since 2006 [72]. The MSEs are devoted to empirically evaluate MaxSAT and PMSAT solvers, and to publish public benchmarks. There are two main tracks in the MSE: the complete track and the incomplete track. The complete track includes all complete solvers that are based on exact methods and the incomplete track for hybrid and local search solvers.

Table 2 and Table 3 summarize each of MSE 2014-2019 benchmarks classes and statistics. The total number of distinct instances made public by MSE 2014-2019 for the incomplete track is 3633 instances (1741 unweighted instances, and 1892 weighted instances) under three classes: random, crafted, and industrial instances. However, since MSE 2017 only two benchmark classes of instances for PMSAT problem are considered and merged: crafted and industrial benchmark instances under both weighed and unweighted categories [73]. The crafted and industrial benchmark instances are encoded from other domains and from real-world applications.

In this evaluation study, subsets of MSE 2017 and MSE 2018 benchmarks were used for the initial tuning of the AVD-SLS solver. Appendix A shows a subset (76 instances) of the PMS benchmark instances from MSE 2017 and MSE 2018, where some of the instances are also a subset of the MSE 2014, 2015, 2016, and 2019 PMS benchmarks as shown in Table 14 - Appendix A. Appendix B shows a subset (81 instances) of the WPMS benchmark instances from MSE 2017 and MSE 2018, where some of the instances are also a subset of MSE 2014, 2015, 2016, and 2019 WPMS benchmarks as shown in Table 15 - Appendix B. We have excluded these two subsets of instances from this evaluation study; thus, a total of 3476 instances were used in this evaluation: 1665 instances from the PMS benchmark and 1811 from the WPMS benchmark.

2) EVALUATION METHODOLOGY

We follow for each MSE the same methodology adopted in the incomplete track:

- Each solver is executed once on each instance within a time limit which is set to 60 CPU second for MSE 2017-2019 benchmarks and to 300 CPU seconds for MSE 2014-2019 benchmarks.
- In each run, the solver prints successively the best solution it has found so far.
- The total number of instances in each benchmark is denoted by #inst. and number of families by #families.
- For each solver on each benchmark, we report number of solved instances denoted by #sol., within parentheses the number of best solutions denoted by (#wins), and

[7] https://www.hpc.kaust.edu.sa/content/shaheen-ii

TABLE 3. WPMS classes and statistics from MSE 2014-2019.

	Benchmark			
	random	crafted	industrial	
Year	#inst. (#families)	#inst. (#families)	#inst. (#families)	Total
2014	280 (8)	310 (17)	410 (8)	**1000 (33)**
2015	662 (16)	319 (16)	610 (11)	**1591 (43)**
2016	502 (12)	331 (17)	630 (13)	**1463 (42)**
2017	-	156 (18)		**156 (18)**
2018	-	172 (20)		**172 (20)**
2019	-	297 (29)		**297 (29)**

score. The best results is presented in **bold** font face for each benchmark and each evaluation criteria.

In this work, we consider three evaluation criteria for each MSE result: number of solved instances, number of best solutions found, and score. Score is based on the cost of the output solutions and measures how far are the solutions found by a solver from the best ones, taking into account the number of solved instances. Solvers with a higher number of best solutions and solved instances have higher scores.

- MSE 2014-2016: We adopt the same score measure as MSE 2018-2019.
- MSE 2017: $score = \Sigma_i \frac{cost\ of\ best\ solution\ for\ i\ found\ by\ any\ solver}{cost\ of\ solution\ for\ i\ found\ by\ solver} : i \in instances\ and\ score \in [0, 1]$.
- MSE 2018-2019: $score = \Sigma_i \frac{(cost\ of\ best\ solution\ for\ i\ found\ by\ any\ solver+1)}{(cost\ of\ solution\ for\ i\ found\ by\ solver\ +1)}$: $i \in instances$, $score \in [0, 1]$, and the best solution found by all incomplete solvers within 300 seconds is considered for each instance.

In this evaluation, for MSE 2014, MSE 2015, and MSE 2016, we merged the benchmarks for both crafted and industrial classes under weighted and unweighted categories to have a coherent and consistent comparison with MSE 2017-2019 results.

3) PMSAT SLS SOLVERS

The state-of-the-art PMSAT SLS solvers experimental results, that compete with AVD-SLS are the participated solvers in MSE 2014-2019: CCLS and its variants, CCMPA, Dist, Dist1, Dist2, Dist-r, DistUP, Ramp, HS-Greedy, CCEHC, SC2016, and SATLike.

4) INITIAL PARAMETER SETTING FOR ADAPTIVE TUNER

The following nine parameters' values are adjusted by the adaptive tuner starting from initial default values adopted from [37] and based on our experimental evaluation on tuning our solver on a subset of benchmark instances from MSE 2017-2018 (Appendix A and Appendix B).

- number of samplings for BMS heuristic: $t = 42$ for PMS and $t = 15$ for WPMS : $t \in [10, 500]$.
- smooth probability for weighting scheme: $sp = 0.000003$ for PMS and $sp = 0.0000001$ iff the number of variables > 2000; otherwise $sp = 0.01$ for WPMS : $sp \in [0, 1]$.
- increment for each falsified clause in the weighting scheme: $h\_inc = 1$ for PMS and $h\_inc = 3$ for WPMS : $h\_inc \in [1, 500]$.
- limit on maximum value on soft clause weight in the weighting scheme: $\zeta = 400$ for PMS and $\zeta = 1$ for WPMS : $\zeta \in [1, 500]$.
- random walk parameter: $wp = 0.091$ for PMS and $wp = 0.1$ for WPMS : $wp \in [0, 1]$.
- noise parameters: $rdprob = 0.01$ and $rwprob = 0.091$ for PMS and $rdprob = 0.01$ and $rwprob = 0.1$ for WPMS : $rdprob\ and\ rwprob \in [0, 1]$.
- threshold of maximum flips without improvement: $max\_non\_improve\_flip = 10000000\ or\ 10M$ for both PMS and WPMS : $max\_non\_improve\_flip \in [1M, 100M]$.
- weight multiplier for initial weights of soft clauses: $weight\_multiplier = 100$ for PMS and $weight\_multiplier = 1$ for WPMS : $weight\_multiplier \in [1, 100]$.
- PMSAT VDS algorithm depth parameter: $depth = 1$ for both PMS and WPMS : $depth \in [1, 35]$.

However, based on our experimental evaluation, we set the initial parameters' values for some PMS and WPMS instances to different initial values based on an instance hard ratio (i.e. ratio of number of hard clauses to hard variables). We found that for some different ranges of hard ratios, the PMS initial parameters' values with the value of $max\_non\_improve\_flip = 65M$ and $depth = 3$ are better as initial values. The hard ratio ranges that have been found to be better with this initialization values are 1.9 to 9.5 for WPMS instances, and 2 to 6.25 for PMS instances. Then, the adaptive tuner will adjust the initialized values to guide the search towards better (feasible) search regions. The adaptive tuner is called before the search begin and then whenever the $max\_non\_improve\_flip$ is reached and no feasible solution found. We set the maximum time limit for the adaptive tuner to 10% of time limit for each call based on our experiments.

TABLE 4. AVD-SLS performance evaluation on MSE 2014-2019 PMS benchmarks by families.

Performance	%solved	#families	Comments
Excellent	100%	45	6 random + 23 crafted + 16 industrial #families
Very good	80-99%	5	1 random + 1 crafted + 2 industrial #families 1 random instance of pmax2sat-hi was never solved
Good	66-79%	4	4 industrial #families 2 industrial instances of bcp-msp were never solved
Poor	51-65%	1	1 industrial #families
Fail	0-50%	7	2 crafted + 5 industrial #families 2 industrial instances of atcoss-mesat were never solved All (5) crafted instances of pseudo-Boolean-primes were never solved
	Total	62	10 unweighted instances were never solved

TABLE 5. AVD-SLS performance evaluation on MSE 2014-2019 WPMS by families.

Performance	%solved	#families	Comments
Excellent	100%	54	15 random + 23 crafted + 16 industrial #families
Very good	80-99%	4	1 random + 3 industrial #families 2 industrial instances of time-tabling were never solved 1 random instance of wpmax2sat-hi was never solved
Good	66-79%	1	1 crafted #families 5 crafted instances of lisbon-wedding were never solved
Poor	51-65%	1	1 industrial #families
Fail	0-50%	3	1 crafted + 1 industrial #families 2 crafted instances of miplib were never solved
	Total	63	10 weighted instances were never solved

B. PERFORMANCE EVALUATION OF AVD-SLS SOLVER

In this section, we evaluate the performance of AVD-SLS solver on both PMS and WPMS, and on each class: random, crafted, and industrial instances. Then, we further focus on the evaluation results based on benchmarks' families. We classify the performance of AVD-SLS based on the results into 5 different classes as shown in Table 4 and Table 5. We consider AVD-SLS fails to solve a benchmark family and categorize a benchmark family as a hard family, if AVD-SLS is unable to solve more than 50% of benchmark's instances for 300 CPU seconds.

Fig. 2 shows the overall performance results of AVD-SLS on both PMS and WPMS. For PMS, the total number of solved instances is 1492 (89.55%), and the average time to find the best solution is below 50 CPU seconds. On the other hand, for WPMS the total number of solved instances is 1774 (97.96%), and the average time to find the best solution is also below 50 CPU seconds as the case of PMS. The results indicate that AVD-SLS, in general, is an efficient solver w.r.t time and percentage of solved instances. Moreover, AVD-SLS remarkably perform better on WPMS than PMS.

1) AVD-SLS RESULTS ON PMS

For **random class**, AVD-SLS was able to solve all the instances from random PMS except one instance that was never solved by any solver. The average time to solve all random PMS instances is 2.35 seconds. Table 4 shows that AVD-SLS has an excellent performance on random families, that were completely solved by AVD-SLS.

For **crafted class**, AVD-SLS was able to solve 94.13% of instances from crafted PMS. The average time to solve crafted PMS instances is 44.34 seconds. Table 4 shows that AVD-SLS has an excellent performance on 88.5% of crafted families, that were completely solved by AVD-SLS. However, AVD-SLS fails to solve many instances from reversi family (multi-agent endgame). Excluding pseudo-Boolean-primes family (as it was never solved by any solver), Fig. 3 shows that the number of solved instances from reversi family is increased with time.

For **industrial class**, AVD-SLS was able to solve 81.26% of instances from industrial PMS and the average time to solve them is 63.49 seconds. Table 4 shows that AVD-SLS has an excellent performance on 55.17% of industrial families.

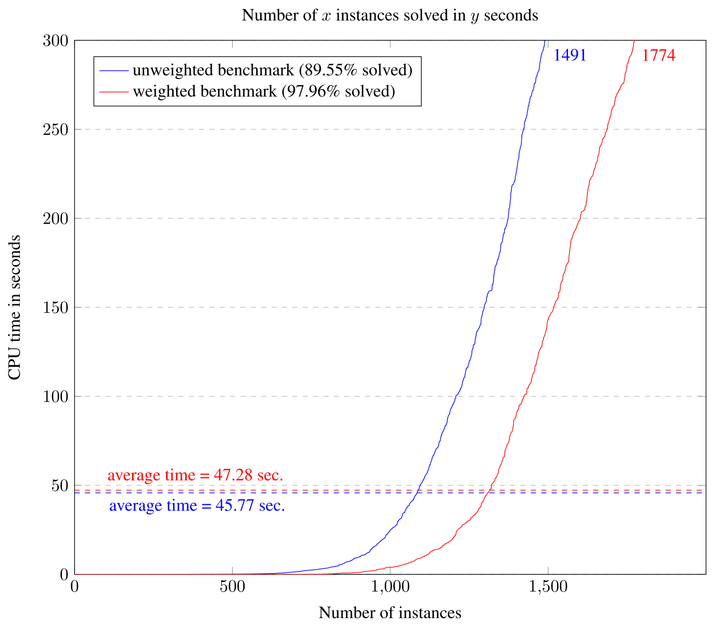

FIGURE 2. Variation of the running time of AVD-SLS with the number of MSE 2014-2019 benchmark instances.

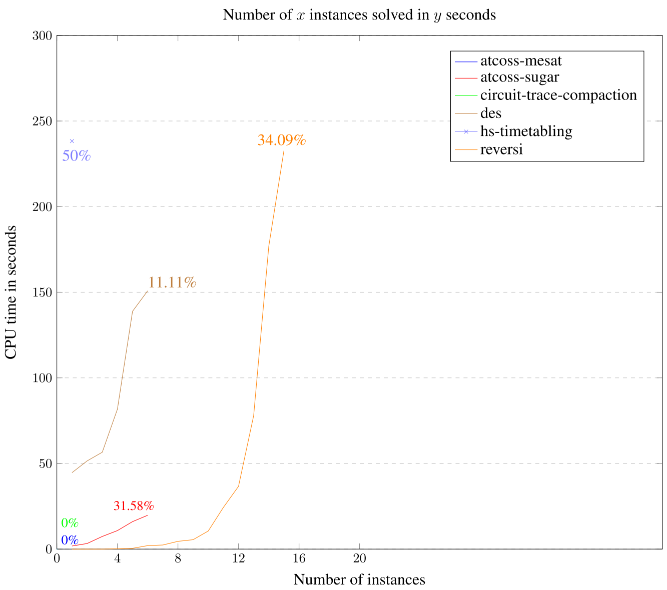

FIGURE 3. Hard family-based PMS instances solving over time.

However, five industrial families have fail performance classification. AVD-SLS was unable to solve any instances from atcoss mesat (air traffic controller shift scheduling) and circuit trace compaction families. Des (diagnosis of discrete event systems) family and atcoss-sugar also reported very poor results. However, the hs-timetabling family has only two instances where only one is solved, which results in 50%. As shown in Fig. 3, we can see that as the time increased, the numbers of solved hard instances from industrial families are modest or non-existent.

2) AVD-SLS RESULTS ON WPMS

For **random class**, AVD-SLS was able to solve all the instances from random WPMS except one instance that was never solved by any solver. The average time to solve all random WPMS instances is 2.11 seconds. Table 5 shows that AVD-SLS has an excellent performance on random families, that were completely solved by AVD-SLS.

For **crafted class**, AVD-SLS was able to solve 97.95% of instances from crafted WPMS. The average time to solve crafted WPMS instances is 54.74 seconds. Table 5 shows

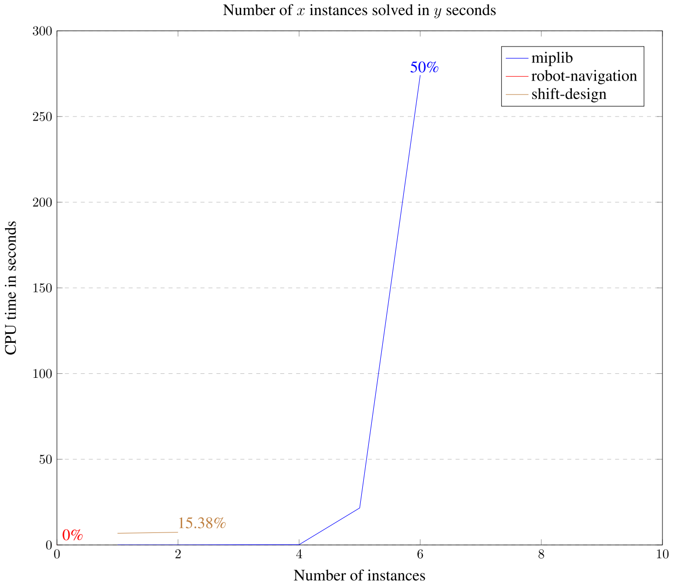

FIGURE 4. **Hard family-based WPMS instances solving over time.**

that AVD-SLS has an excellent performance on 92% of crafted families, that were completely solved by AVD-SLS. However, AVD-SLS was unable to solve more than 50% of mip-lib-mps family from PB domain. However, two miplib instances were never solved before, and we can see that (Fig. 4) as the time increased, the number of solved instances from miplib family is increased.

For **industrial class**, AVD-SLS was able to solve 96.20% of instances from industrial WPMS and the average time to solve them is 86.30 seconds. Table 5 shows that AVD-SLS has an excellent performance on 72.72% of industrial families. However, none of robot navigation family three instances was solved by AVD-SLS. And, as shown in Fig. 4 as the time increased, the numbers of solved hard instances from industrial families robot navigation and shift design are modest or non-existent.

In concluding this section, we show that AVD-SLS has an excellent performance results on both PMS and WPMS. However, AVD-SLS performing remarkably better on WPMS than PMS; especially on industrial families. Out of 125 benchmark families, AVD-SLS shows an excellent performance results on about 80% of benchmark families which completely solved by AVD-SLS. On the other hand, few hard instances were solved over time.

C. COMPARISON OF AVD-SLS WITH STATE-OF-THE-ART SLS SOLVERS

AVD-SLS is compared to each SLS solver participated in MSE 2014-2019. The **results on PMS** are shown in Table 6 and Table 7. The detailed results by family-based benchmarks for PMS instances are shown in Appendix C. We highlight here best and worse results of AVD-SLS on family-based benchmarks compared to SLS solvers results.

For **PMS 2014**: Most SLS solvers were able to solve random instances and AVD-SLS was the best SLS solver for all three evaluation criteria except for total number of solved crafted instances, where Dist solver was able to solve two more instances. Dist was the second best solver that is competitive for random and crafted benchmarks. However, AVD-SLS is a prominent solver for merged benchmark for all three evaluation criteria. For family-based results as shown in Table 16 - Appendix C, AVD-SLS has the best number of solved instances and average score with 20 best families, where second come Dist with 10 best families. Moreover, only Dist and AVD-SLS were able to solve a number of instances from atcoss-sugar family. And all SLS solvers in this MSE failed to solve any of atcoss-mesat and circuit-trace-compaction families instances.

For **PMS 2015**: Most SLS solvers were able to solve random instances. For crafted and industrial benchmarks, AVD-SLS, Dist2 and DistUP were competitive. For the merged benchmark, Dist2 solver was able to solve one more instance but AVD-SLS have best number of best solutions and best score. Dist2 was the second best solver that is competitive for random and crafted benchmarks. For family-based results as shown in Table 17 - Appendix C, AVD-SLS has the best number of solved instances and average score with 20 best families, where second come Dist2 with 10 best families. Moreover, all Distinction-based solvers, CCEHC and AVD-SLS were able to solve a number of instances from atcoss-sugar family, where CCEHC has the best results. For Des family, most of SLS solvers including AVD-SLS were able to solve a number of Des instances, where DistUP has the best results. And as in MSE 2014, all SLS solvers in this MSE failed to solve any of atcoss-mesat and circuit-trace-compaction families instances.

TABLE 6. Summary results of AVD-SLS and PMSAT SLS solvers on MSE 2014-2016 PMS.

	MSE 2014-2016 - Unweighted Benchmark (300 seconds)			
	PMS_random_2014 #inst. (210)	PMS_crafted_2014 #inst. (414)	PMS_industrial_2014 #inst. (553)	**PMS_crafted_industrial_2014** #inst. (967)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**209 (209) - 0.995**	375 **(364) - 0.905**	**434 (394) - 0.770**	**816 (764) - 0.827**
CCLS2014	209 (207) - 0.995	307 (294) - 0.739	94 (58) - 0.155	405 (355) - 0.405
CCMPA	208 (204) - 0.990	305 (277) - 0.733	103 (80) - 0.170	414 (362) - 0.413
Dist	**209 (209) - 0.995**	**377** (349) - 0.899	386 (195) - 0.615	768 (548) - 0.735
	PMS_random_2015 #inst. (210)	PMS_crafted_2015 #inst. (669)	PMS_industrial_2015 #inst. (584)	**PMS_crafted_industrial_2015** #inst. (1253)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**209 (209) - 0.995**	630 **(503) - 0.925**	465 **(403) - 0.773**	1095 **(906) - 0.854**
CCEHC	**209 (209) - 0.995**	548 (431) - 0.779	402 (165) - 0.580	950 (596) - 0.686
CCLS2015	208 (207) - 0.990	308 (293) - 0.459	114 (77) - 0.180	422 (370) - 0.329
Dist1	**209 (209) - 0.995**	582 (426) - 0.834	453 (183) - 0.637	1035 (609) - 0.742
Dist2	**209** (205) - 0.994	**632** (437) - 0.885	464 (178) - 0.635	**1096** (615) - 0.768
DistUP	**209 (209) - 0.995**	583 (418) - 0.837	**474** (191) - 0.673	1057 (609) - 0.760
	PMS_random_2016 #inst. (210)	PMS_crafted_2016 #inst. (669)	PMS_industrial_2016 #inst. (584)	**PMS_crafted_industrial_2016** #inst. (1253)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**209 (209) - 0.995**	**630 (532) - 0.923**	**465 (398) - 0.773**	**1095 (930) - 0.853**
CCEHC	**209 (209) - 0.995**	546 (425) - 0.772	411 (170) - 0.583	957 (595) - 0.684
CCLS	208 (207) - 0.990	308 (290) - 0.459	113 (79) - 0.180	421 (369) - 0.329
Dist	**209** (208) - **0.995**	590 (456) - 0.851	463 (180) - 0.646	1053 (636) - 0.755
Dist-r	**209 (209) - 0.995**	411 (343) - 0.590	411 (122) - 0.520	822 (465) - 0.558
HS-Greedy	**209** (22) - 0.795	622 (103) - 0.760	451 (53) - 0.366	1073 (156) - 0.577
Ramp	**209** (206) - **0.995**	312 (282) - 0.463	121 (90) - 0.189	433 (372) - 0.336
SC2016	208 (208) - 0.990	-	130 (99) - 0.208	-

For **PMS 2016**: Similar to PMS 2014 and 2015, most SLS solvers were able to solve random instances. For all benchmarks, AVD-SLS was the best solver for all three evaluation criteria. Dist was the second competitive solver. For family-based results as shown in Table 18 - Appendix C, AVD-SLS has the best number of solved instances and average score with 17 best families, where second come Dist with 10 best families. Moreover, Dist, CCEHC and AVD-SLS were able to solve a number of instances from atcoss-sugar family, where CCEHC has the best results. For Des family, most of SLS solvers including AVD-SLS were able to solve a number of Des instances, where Dist has the best results. And as in MSE 2014 and 2015, all SLS solvers in this MSE failed to solve any of atcoss-mesat and circuit-trace-compaction families instances.

For **PMS 2017**: AVD-SLS was the best solver for all three evaluation criteria. The results of CCEHC were improved as time limit increased to 300 seconds. In this MSE, AVD-SLS is a prominent solver. For family-based results as shown in Table 19 - Appendix C, AVD-SLS has the best number of solved instances and average score with 14 best families, where second come CCEHC with 6 best families. Moreover, AVD-SLS and CCEHC were able to solve a number of instances from atcoss-sugar family. And all SLS solvers in this MSE failed to solve any of atcoss-mesat, Des, and close-solutions (from Satisfiabily domain) families instances.

For **PMS 2018**: AVD-SLS was the best solver for all three evaluation criteria. And the number of best solutions found so far were approximately more than 1.7 times for both solvers as time limit increased to 300 seconds. It is remarkable to see that the number of solved instances are not increased relatively as the number of best solutions found is increased. For family-based results as shown in Table 20 - Appendix C, AVD-SLS has the best number of solved instances and average score with 14 best families, where second come SATLike with 8 best families. AVD-SLS was able to solve a number of reversi instances that was never solved by SATLike. Moreover, AVD-SLS and CCEHC were able to solve a number of instances from atcoss-sugar family. And both AVD-SLS and SATLike in this MSE were failed to solve any of atcoss-mesat, Des, and close-solutions families instances.

For **PMS 2019**: AVD-SLS is a competitive solver, although no other SLS solver participated in this evaluation. The scores and number of best solutions are better than some other well-known hybrid solvers. In this evaluation, AVD-SLS has the best number of solved instances and average score with 7 best families as shown in Table 21 - Appendix C. As AVD-SLS is the only SLS solver in this section, the results shown in Table 7 - section PMS_2019 - are based on the best solution found so far for each instance by any SLS solver during the past MSE 2014-2018. If an instance is never tested before by any SLS solver, we set the solution found by AVDS-SLS as the best solution found so far (about 12% of unweighted benchmark instances were never tested before by SLS solvers). In this evaluation, AVD-SLS failed to solve any of atcoss-mesat, and pseudo-Boolean-primes families instances, that never solved by any SLS solver. Three new benchmark families were included in this MSE, where AVD-SLS perform very well and the reported results has a score range from 0.860 to 0.987 as shown in Table 21 - Appendix C.

Fig. 5 shows the performance comparison on MSE 2014-2018 crafted and industrial instances, where PMSAT SLS solvers have been participated. For each column of each solver per year:

- length of a column represents the percentage (%) of total solved instances per year.
- the colored part (e.g. light violet for AVD-SLS) represents the percentage (%) of total number of best solutions found by a solver.

From this summary, we show that AVD-SLS has the best results throughout MSE 2014-2019 w.r.t MSE three evaluation measures: number of solved instances, score and number of best solutions found. AVD-SLS report the best performance results on many graph-theory families (such as max-clique, maxcut, ramsey, and set-covering), on some verification families (such as mbd and bcp-syn), and on some optimization problems such as uaq (User Authorization Query problem). However, the **hardest PMS families** for AVD-SLS are: atocss-mesat, circuit trace compaction, and robot navigation. However competitive PMSAT SLS solvers such as AVD-SLS, SATLike, Dist, and CCEHC were able to solve some instances from other hard benchmark families such as atcoss-sugar, des, close-solutions, and reversi.

Next, the **results on WPMS** are shown in Table 8 and Table 9. The detailed results by family-based benchmarks for WPMS problem instances are shown in Appendix D.

TABLE 7. Summary results of AVD-SLS and PMSAT SLS solvers on MSE 2017-2019 PMS.

	MSE 2017-2019 - Unweighted Benchmark	
	60 seconds	300 seconds
	PMS_2017 #inst. (134)	PMS_2017 #inst. (134)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**115 (92) - 0.840**	**115 (96) - 0.841**
CCEHC	86 (42) - 0.521	103 (48) - 0.602
Dist	101 (37) - 0.533	104 (33) - 0.552
	PMS_2018 #inst. (100)	PMS_2018 #inst. (100)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**83 (41) - 0.762**	**83 (68) - 0.820**
SATLike	80 (26) - 0.710	81 (50) - 0.751
	PMS_2019 #inst. (253)	PMS_2019 #inst. (253)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	215 (121) - 0.785	215 (179) - 0.831

We highlight here best and worse results of AVD-SLS on family-based benchmarks compared to SLS solvers results.

For **WPMS 2014**: AVD-SLS was the best SLS solver for all three evaluation criteria for industrial and merged benchmarks. However the number of best solutions found for random benchmark is the worse among all solvers. For crafted benchmarks, Dist has the best score. However, AVD-SLS was the best solver on crafted benchmark for two evaluation criteria. For family-based results as shown in Table 22 - Appendix D, AVD-SLS has the best number of solved instances and best average score with 19 best families, where second come Dist and CCMPA with 6 best families. We found that all SLS solvers have reported the best results for two auctions benchmark families. Moreover, AVD-SLS was able to solve more instances from hard pseudo-miplib-mps and hs-timetabling benchmark families than other SLS solvers. And for upgrade-ability problem family, AVD-SLS was the only solver that solved all instances, where non solved by other SLS solvers. It is remarkable that both Dist and Dist-r were not able to solve any instance from frb, ramsey, and maxcut benchmark families.

For **WPMS 2015**: Most SLS solvers were able to solve random instances but AVD-SLS, as in MSE 2014, has the worse number of best solutions among all solvers. For crafted benchmark, CCEHC has the best score. However, AVD-SLS was the best solver for crafted benchmark for two evaluation criteria and was the best solver for all three evaluation criteria on industrial and merged benchmarks. For family-based

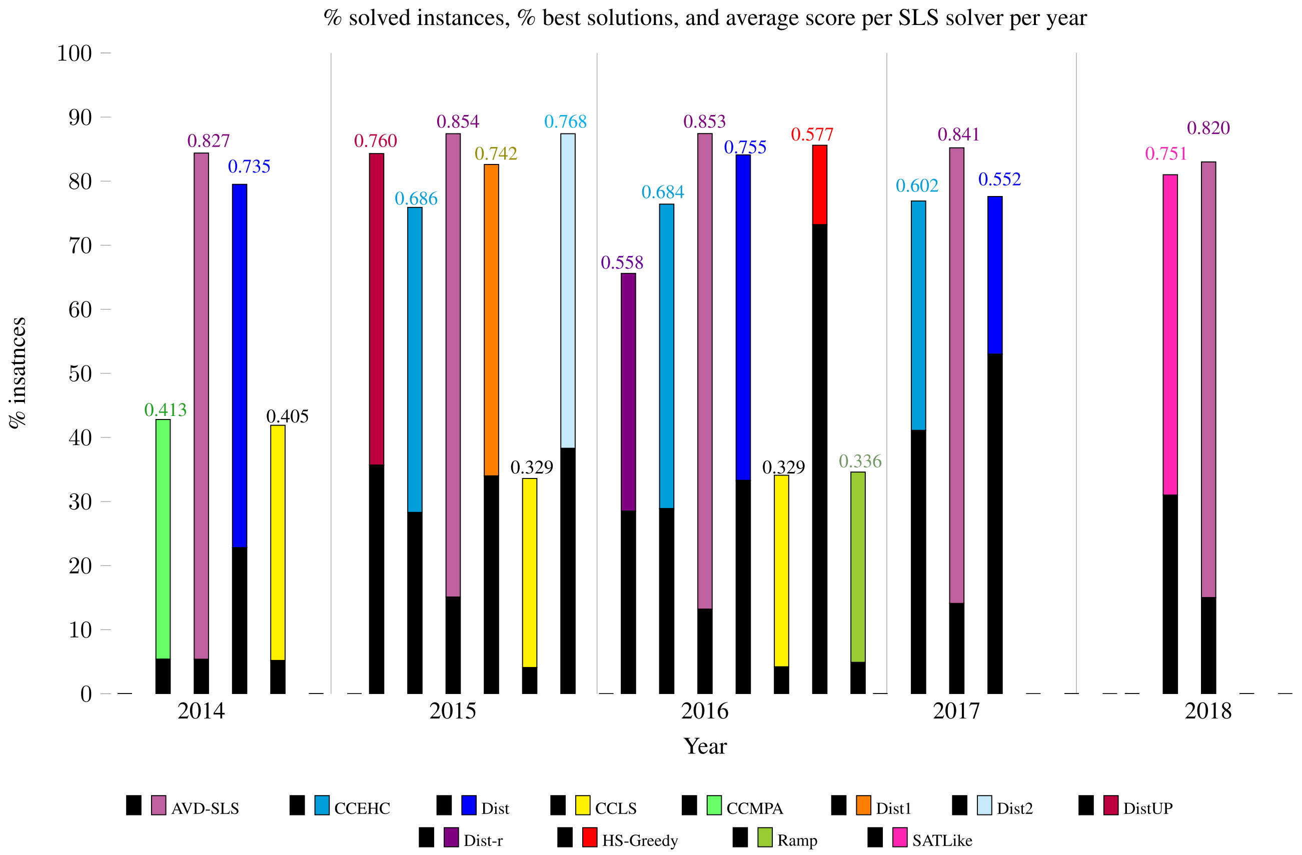

FIGURE 5. Performance comparison of AVD-SLS with state-of-the-art PMSAT SLS solvers on MSE 2014-2018 PMS.

results as shown in Table 23 - Appendix D, AVD-SLS has the best number of solved instances and best average score with 17 best families, where second come CCEHC with 7 best families. And as in MSE 2014, all SLS solvers have reported the best results for two auctions benchmark families. Moreover, AVD-SLS were able to solve more instances from hard hs-timetabling benchmark families than other SLS solvers. Both HS-Greedy and AVD-SLS were able to solve all instances from timetabling family.

For **WPMS 2016**: For the random benchmark, only three solvers have the best results: CCEHC, Ramp, and SC2016. For crafted benchmark, CCEHC has the best score. However, AVD-SLS was the best solver on crafted benchmark for two evaluation criteria and was the best solver for all three evaluation criteria on industrial and merged benchmarks. For family-based results as shown in Table 24 - Appendix D, AVD-SLS has the best number of solved instances and best average score with 22 best families, where second come Ramp with 7 best families. And as in MSE 2014 and 2015, all SLS solvers have reported the best results for two auctions benchmark families. And for abstraction-refinement family, AVD-SLS was the only solver that solved all instances, where non solved by other SLS solvers. Also, HS-Greedy were able to solve more instances from hard hs-timetabling benchmark family than AVD-SLS solver, that was the best for this family. However, AVD-SLS was the only solver that was able to solve all instances from relational-inference family. And as in MSE 2014, both Dist and Dist-r were not able to solve any instance from frb, ramsey, and maxcut benchmark families.

For **WPMS 2017**: AVD-SLS was the best solver for all three evaluation criteria. The results of all solvers has very slight improvement as time limit increased to 300 seconds. In this MSE, AVD-SLS is a prominent solver. Generally, many good-solutions were found by SLS solvers which indicated by the enhanced score. For family-based results as shown in Table 25 - Appendix D, AVD-SLS has the best number of solved instances and best average score with 14 best families, where second come CCEHC with 6 best families. Moreover, AVD-SLS the only solver that was able to solve all instances from min-width family. And AVD-SLS solved a number of instances from pseudo-miplib-mps and shift design hard benchmark families, where non solved by other SLS solvers.

For **WPMS 2018**: AVD-SLS was the best solver for all three evaluation criteria. And the number of best solutions found so far were approximately doubled for AVDS-SLS and 4.4 times increased for SATLike as time limit increased to 300 seconds. It is also remarkable as for PMS, that the number of solved instances are not increased relatively as the number of best solutions found increased by double or more. For

TABLE 8. Summary results of AVD-SLS and PMSAT SLS solvers on MSE 2014-2016 WPMS.

	MSE 2014-2016 - Weighted Benchmark (300 seconds)			
	WPMS_random_2014 #inst. (280)	WPMS_crafted_2014 #inst. (302)	WPMS_industrial_2014 #inst. (402)	**WPMS_crafted_industrial_2014** #inst. (704)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**279** (119) - 0.688	**298 (225)** - 0.818	**391 (369) - 0.963**	**689 (589) - 0.900**
CCLS2014	278 (277) - 0.993	221 (138) - 0.700	71 (40) - 0.159	292 (178) - 0.391
CCMPA	278 (274) - 0.993	247 (148) - 0.730	76 (47) - 0.179	323 (195) - 0.415
Dist	**279 (279) - 0.996**	276 (182) - **0.824**	236 (57) - 0.541	512 (239) - 0.663

	WPMS_random_2015 #inst. (662)	WPMS_crafted_2015 #inst. (311)	WPMS_industrial_2015 #inst. (586)	**WPMS_crafted_industrial_2015** #inst. (897)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**661** (119) - 0.514	**307 (213)** - 0.780	**573 (420) - 0.905**	**880 (633) - 0.862**
CCEHC	**661** (659) - **0.998**	292 (192) - **0.894**	411 (131) - 0.539	703 (323) - 0.662
CCLS2015	660 (660) - 0.997	206 (131) - 0.631	73 (36) - 0.102	279 (167) - 0.285
Dist1	**661 (660) - 0.998**	284 (165) - 0.808	525 (75) - 0.624	809 (240) - 0.688
Dist2	**661 (660) - 0.998**	296 (165) - 0.844	524 (88) - 0.637	820 (253) - 0.709
DistUP	-	-	-	-

	WPMS_random_2016 #inst. (502)	WPMS_crafted_2016 #inst. (319)	WPMS_industrial_2016 #inst. (606)	**WPMS_crafted_industrial_2016** #inst. (925)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**501** (119) - 0.532	**315 (213)** - 0.778	**593 (411) - 0.875**	**908 (624) - 0.841**
CCEHC	**501** (497) - **0.998**	302 (192) - **0.897**	434 (100) - 0.515	736 (292) - 0.647
CCLS	500 (498) - 0.996	216 (131) - 0.639	73 (42) - 0.103	289 (173) - 0.288
Dist	119 (119) - 0.237	204 (88) - 0.518	566 (114) - 0.705	770 (202) - 0.640
Dist-r	119 (119) - 0.237	199 (76) - 0.512	519 (113) - 0.562	718 (189) - 0.545
HS-Greedy	**501** (45) - 0.897	312 (69) - 0.778	566 (14) - 0.441	878 (83) - 0.557
Ramp	500 (496) - 0.996	228 (134) - 0.650	84 (47) - 0.124	312 (181) - 0.305
SC2016	**501 (500) - 0.998**	199 (137) - 0.596	102 (48) - 0.145	301 (185) - 0.300

family-based results as shown in Table 26 - Appendix D, AVD-SLS has the best number of solved instances and best average score with 11 best families, where SATLike was the best for 7 families. For causal-discovery family, AVD-SLS was the best and found optimal solutions for all instances (zero cost). Also, AVD-SLS was able to solve more instances from hard hs-timetabling benchmark family. The hardest families that never solved by SLS solvers in this MSE are: robot navigation and pseudo-miplib-mps families.

For **WPMS 2019**: AVD-SLS is a competitive solver, although no other SLS solver participated in this evaluation. The scores and number of best solutions are better than some other well-known hybrid solvers. In this evaluation, AVD-SLS has the best number of solved instances and average score with 5 best families as shown in Table 27 - Appendix D. As AVD-SLS is the only SLS solver in this MSE, the results shown in Table 9 - section WPMS_2019 - are based on the best solution found so far for each instance by any SLS solver during the past MSE 2014-2018. If an instance is never tested before by any SLS solver, we set the solution found by AVDS-SLS as the best solution found so far (about 20% of weighted benchmark instances were never tested before by SLS solvers). The hardest families that never solved by SLS solvers in this MSE are: robot navigation and shift design families.

Fig. 6 shows the performance comparison on MSE 2014-2018 crafted and industrial instances, where PMSAT SLS solvers have been participated. For each column of each solver per year:

- length of a column represents the percentage (%) of total solved instances per year.
- the colored part (e.g. light violet for AVD-SLS) represents the percentage (%) of total number of best solutions found by a solver.

TABLE 9. Summary results of AVD-SLS and PMSAT SLS solvers on MSE 2017-2019 WPMS.

	MSE 2017-2019 - Weighted Benchmark	
	60 seconds	300 seconds
	WPMS_2017 #inst. (100)	**WPMS_2017** #nst. (100)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**89 (58) - 0.857**	**90 (58) - 0.867**
CCEHC	57 (27) - 0.486	65 (29) - 0.553
Dist	64 (30) - 0.555	65 (29) - 0.568
	WPMS_2018 #inst. (108)	**WPMS_2018** #inst. (108)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**94 (38) - 0.815**	**95 (65) - 0.844**
SATLike	90 (11) - 0.696	92 (46) - 0.741
	WPMS_2019 #inst. (261)	**WPMS_2019** #inst. (261)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	233 (90) - 0.754	235 (181) - 0.850

% solved instances, % best solutions, and average score per SLS solver per year

From this summary, we show that AVD-SLS has the best results throughout MSE 2014-2019 w.r.t MSE three evaluation measures: number of solved instances, score and number of best solutions found. AVD-SLS report the best performance results on many graph-theory families (such as maxcut, and set-covering), on realizability optimization problems such as power distribution, and other optimization problems such as random network, min-width, abstraction refinement, causal discovery, and CSG families. However, the **hardest PMS families** for AVD-SLS are: robot naviagtion and shift design. Competitive PMSAT SLS solvers such as AVD-SLS, SATLike, Dist, and CCEHC were able to solve some instances from other hard benchmark families such as pseudo-miblip-mps and hs-timetabling.

We conclude this experimental evaluation results of AVD-SLS and PMSAT SLS solvers that participated in MSE 2014-2019 on both PMS and WPMS with more than 3400 instances, that AVD-SLS reported the best results and outperform all PMSAT SLS solvers, based on the evaluation criteria presented in this study that were adopted from MSE 2014-2019. Generally, AVD-SLS improves the quality of found solutions when the time limit increased. The minimum score for AVD-SLS in this evaluation, as shown in Fig. 5 and Fig. 6, ranges from 0.820 to 0.900. This evaluation study shows that AVD-SLS perform remarkably better on WPMS instances.

D. COMPARISON OF AVD-SLS WITH ALL STATE-OF-THE-ART MaxSAT EVALUATION 2014-2019 PMSAT SOLVERS

We also compare AVD-SLS with all PMSAT solvers participated in MSE 2014-2019. The **results on PMS** are shown in Table 10 and Table 11. In this section, we plot distribution of scores per instances figures for each MSE; the same used by latest MSE since 2017. In each figure, each solver results are represented by a curve, where each point represent the score of each solved/unsolved instance. We encoded each solver results with a unique color, where results of AVD-SLS are represented with a violet color in all figures. We ordered PMSAT solvers in the legend from best-scored to worse-scored.

For **PMS 2014**: Both AVD-SLS and Dist have best results on random instances. However, AVD-SLS has the best number of best solutions for crafted benchmark. AVD-SLS is competitive to a number of hybrid solvers. Based on score measure results shown in Fig. 7, AVD-SLS is the third best solver in this Evaluation after WPM-2014-in and optimax2-rn-i. For family-based results as shown in Table 16 - Appendix C, AVD-SLS has the best average score for 4 benchmark families.

For **PMS 2015**: Most SLS solvers have best results on random instances. AVD-SLS is competitive to a number of hybrid solvers. Based on score measure results shown

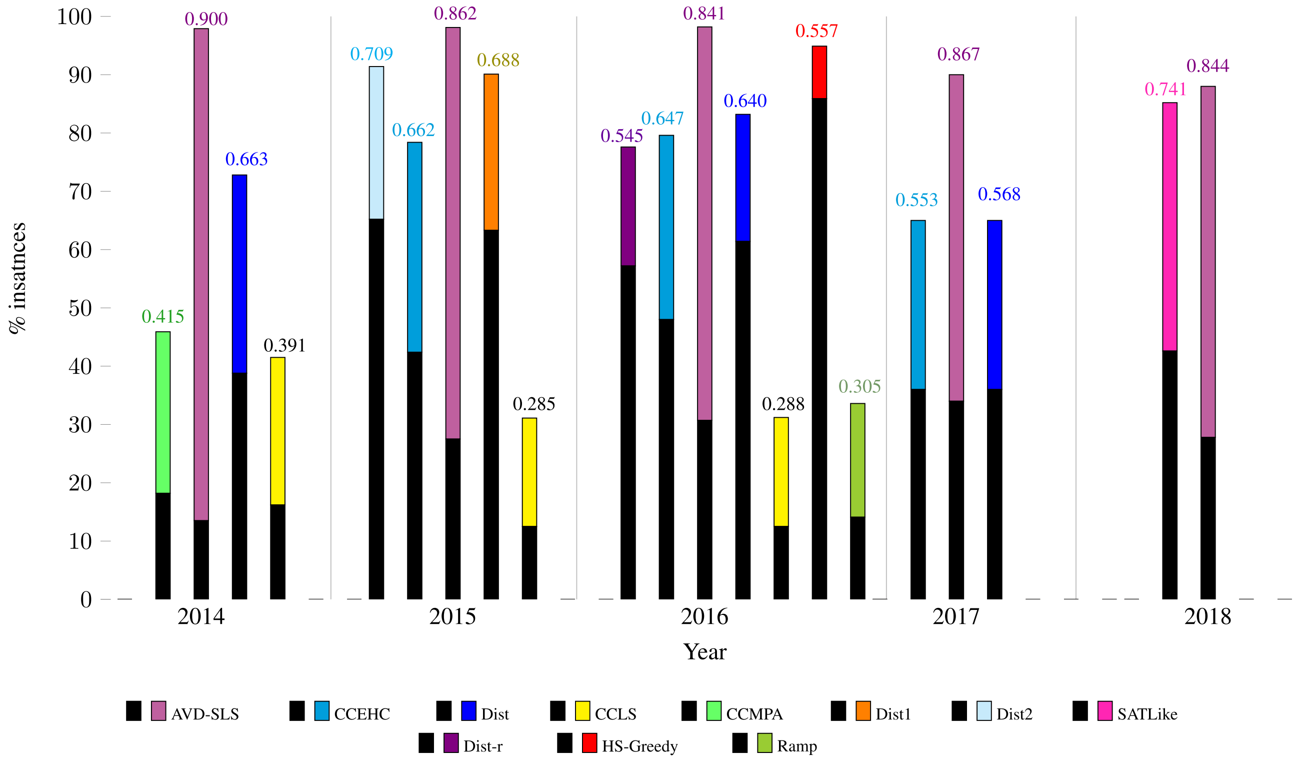

FIGURE 6. Performance comparison of AVD-SLS with state-of-the-art PMSAT SLS solvers on MSE 2014-2018 WPMS.

in Fig. 8, AVD-SLS is ranked the fourth best solver in this Evaluation. For family-based results as shown in Table 17 - Appendix C, AVD-SLS has the best average score for 2 benchmark families.

For **PMS 2016**: For random benchmark, almost all SLS solves random instances with best solutions. For crafted benchmark, AVD-SLS was able to solve the maximum number of instances. Based on score measure results shown in Fig. 9, AVD-SLS ranked in the fifth position in this Evaluation. For family-based results as shown in Table 18 - Appendix C, AVD-SLS has the best average score for 4 benchmark families.

For **PMS 2017**: For 60 seconds time limit, AVD-SLS has the best number of best solutions and ranked the first based on score measue. For 300 seconds, maxroster solver results improved significantly as the time increased, and AVD-SLS ranked as the second best solver as shown in Fig. 10. For family-based results as shown in Table 19 - Appendix C, AVD-SLS has the best average score for 3 benchmark families.

For **PMS 2018**: AVD-SLS is a competitive solver with number of hybrid solvers, where the best number of best solutions is achieved by AVD-SLS for both time limits. AVD-SLS ranked as the third best solver for 60 seconds time limit and fourth for 300 seconds time limit as shown in Fig. 11; maxroster solver results improved significantly as the time increased. For family-based results as shown in Table 20 - Appendix C, AVD-SLS has the best average score for 4 benchmark families.

For **PMS 2019**: Similar to MSE 2018 results, AVD-SLS has shown it is a competitive solver with number of hybrid solvers, where the best number of best solutions is achieved by AVD-SLS for both time limits. The increasing of best solutions found by AVD-SLS is remarkable as the time limit increased. Based on score measure results shown in Fig. 12, AVD-SLS was competitive for a number of hybrid solvers. For family-based results as shown in Table 21 - Appendix C, AVD-SLS has the best average score for 7 benchmark families.

Next, the **results on WPMS Benchmark** are shown in Table 12 and Table 13.

For **WPMS 2014**: Only SAT4J-ms-inc solver was unable to solve the random instances. However, AVD-SLS shows competitive results and has the best number of best crafted solutions. For industrial instances, AVD-SLS shows competitive results with best performing hybrid solvers as shown in Fig. 13, where AVD-SLS ranked as the fourth best solver. For family-based results as shown in Table 22 - Appendix D, AVD-SLS has the best average score for 8 benchmark families.

For **WPMS 2015**: AVD-SLS is ranked the first based on score measure results as shown in Fig. 14 and was able to

TABLE 10. Summary results of AVD-SLS with PMSAT solvers on MSE 2014-2016 PMS.

	MSE 2014-2016 - Unweighted Benchmark (300 seconds)			
	PMS_random_2014 #inst. (210)	PMS_crafted_2014 #inst. (414)	PMS_industrial_2014 #inst. (553)	**PMS_crafted_industrial_2014** #inst. (967)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**209 (209) -0.995**	375 **(350)** - 0.896	434 (275) - 0.730	809 (624) - 0.801
CCLS2014	**209** (207) - **0.995**	307 (294) - 0.739	94 (58) - 0.155	401 (352) - 0.405
CCMPA	208 (204) - 0.990	305 (276) - 0.731	103 (80) - 0.170	408 (356) - 0.410
Dist	**209 (209) -0.995**	377 (346) - 0.892	386 (168) - 0.588	763 (514) - 0.718
SAT4J-ms-inc	0 (0) - 0.0	200 (200) - 0.483	117 (117) - 0.212	317 (317) - 0.328
WPM-2014-in	**209** (0) - 0.696	**414** (298) - 0.960	**523 (400) - 0.890**	**937 (698) - 0.920**
antom_nc	**209** (1) - 0.807	406 (94) - 0.881	479 (106) - 0.697	885 (200) - 0.776
optimax2-r-i	186 (0) - 0.485	390 (99) - 0.813	503 (193) - 0.677	893 (292) - 0.735
optimax2-rn-i	**209** (0) - 0.805	**414** (290) - **0.967**	**523** (336) - 0.850	**937** (626) - 0.900
	PMS_random_2015 #inst. (210)	PMS_crafted_2015 #inst. (669)	PMS_industrial_2015 #inst. (584)	**PMS_crafted_industrial_2015** #inst. (1253)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**209 (209) - 0.995**	630 (400) - 0.781	465 (277) - 0.736	1095 (677) - 0.760
CCEHC	**209 (209) - 0.995**	548 (393) - 0.736	402 (155) - 0.556	950 (548) - 0.652
CCLS2015	208 (207) - 0.990	308 (293) - 0.459	114 (75) - 0.179	422 (368) - 0.328
Dist1	**209 (209) - 0.995**	582 (393) - 0.717	453 (173) - 0.612	1035 (566) - 0.668
Dist2	**209** (205) - 0.994	632 (411) - 0.751	464 (163) - 0.605	1096 (574) - 0.683
DistUP	**209 (209) - 0.995**	583 (396) - 0.726	474 (174) - 0.640	1057 (570) - 0.686
ILP-2015-in	53 (53) - 0.252	389 (389) - 0.581	238 (237) - 0.407	627 (626) - 0.500
WPM3-2015-in	145 (35) - 0.567	629 **(552) - 0.909**	563 **(511) - 0.945**	1192 **(1063) - 0.926**
optiriss-def-i	178 (2) - 0.463	641 (492) - 0.908	566 (470) - 0.915	1207 (962) - 0.911
optiriss-sel-i	74 (0) - 0.194	**647** (302) - 0.830	**572** (329) - 0.753	**1219** (631) - 0.794
	PMS_random_2016 #inst. (210)	PMS_crafted_2016 #inst. (669)	PMS_industrial_2016 #inst. (584)	**PMS_crafted_industrial_2016** #inst. (1253)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**209 (209) - 0.995**	**630** (400) - 0.779	465 (279) - 0.736	1095 (679) - 0.759
CCEHC	**209 (209) - 0.995**	546 (395) - 0.730	411 (150) - 0.556	957 (545) - 0.649
CCLS	208 (207) - 0.990	308 (290) - 0.459	113 (79) - 0.180	421 (369) - 0.329
Dist	**209** (208) - **0.995**	590 (415) - 0.749	463 (152) - 0.621	1053 (567) - 0.689
Dist-r	**209 (209) - 0.995**	411 (342) - 0.577	411 (121) - 0.511	822 (463) - 0.546
HS-Greedy	**209** (22) - 0.795	622 (100) - 0.645	451 (46) - 0.343	1073 (146) - 0.504
Ramp	**209** (206) - **0.995**	312 (281) - 0.463	121 (90) - 0.189	433 (371) - 0.335
SC2016	208 (208) - 0.990	-	130 (99) - 0.208	-
Naps-1.02-ms	0 (0) - 0.0	240 (240) - 0.359	22 (22) - 0.038	262 (262) - 0.209
Optiriss6-in	179 (1) - 0.462	584 (499) - 0.829	523 (433) - 0.818	1107 (932) - 0.824
SsMonteCarlo	**209** (139) - 0.972	308 (208) - 0.450	57 (17) - 0.056	365 (225) - 0.267
WPM3-2015-in	160 (34) - 0.629	622 (536) - 0.893	**564** (499) - **0.941**	**1186** (1035) - 0.915
dsat-wpm3-in-pms	153 (127) - 0.723	623 **(575) - 0.926**	532 **(513)** - 0.906	1155 **(1088) - 0.917**
dsat-wpm3-s-in-pms	158 (121) - 0.735	608 (519) - 0.888	471 (425) - 0.784	1079 (944) - 0.840

solve the maximum number of instances on all classes. For family-based results as shown in Table 23 - Appendix D, AVD-SLS has the best average score for 5 benchmark families.

For **WPMS 2016**: Similar results to MSE 2015 are reported to AVD-SLS. AVD-SLS was able to solve the maximum number of instances on all classes and ranked here as the third best solver after WPM3-2015-in, based on score measure

TABLE 11. Summary results of AVD-SLS with PMSAT solvers on MSE 2017-2019 PMS.

	MSE 2017-2019 - Unweighted Benchmark	
	60 seconds	300 seconds
	PMS_2017 #inst. (134)	**PMS_2017** #nst. (134)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	115 **(71)** - **0.784**	115 (71) - 0.777
CCEHC	86 (36) - 0.499	103 (37) - 0.568
Dist	101 (29) - 0.504	104 (28) - 0.514
Open-WBO-LSU	123 (30) - 0.687	122 (31) - 0.689
MaxHS-inc	123 (6) - 0.611	**127** (17) - 0.642
maxroster	100 (49) - 0.645	125 **(74)** - **0.834**
WPM3-in	**133** (8) - 0.554	133 (8) - 0.529
SAT4J	113 (8) - 0.559	123 (11) - 0.581
LMHS-inc	112 (10) - 0.535	123 (15) - 0.567
	PMS_2018 #inst. (100)	**PMS_2018** #inst. (100)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	83 **(31)** - 0.702	83 **(44)** - 0.752
SATLike	80 (20) - 0.657	81 (37) - 0.692
SATLike-c	87 (21) - **0.729**	**96** (41) - **0.862**
LinSBPS	**92** (12) - 0.714	95 (30) - 0.783
Open-WBO-Inc-OBV	**92** (8) - 0.662	95 (11) - 0.719
Open-WBO-Inc-MCS	**92** (8) - 0.636	95 (13) - 0.691
Open-WBO-Gluc	**92** (6) - 0.623	95 (8) - 0.679
Open-WBO-Riss	90 (7) - 0.577	95 (9) - 0.643
maxroster	70 (20) - 0.562	93 (37) - 0.831
	PMS_2019 #nst. (253)	**PMS_2019** #nst. (253)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	215 **(91)** - 0.718	215 (118) - 0.758
Loandra	**241** (81) - **0.813**	245 **(119)** - **0.878**
LinSBPS2018	**241** (76) - 0.780	**246** (110) - 0.839
SATLike-c	236 (84) - 0.766	245 (121) - 0.837
Open-WBO-g	239 (39) -0.693	245 (53) - 0.747
sls-mcs-lsu	195 (75) - 0.680	227 (99) - 0.777
sls-mcs	195 (75) - 0.680	227 (107) - 0.790
Open-WBO-ms	217 (32) - 0.606	242 (49) - 0.722

results shown in figure 15. For family-based results as shown in Table 24 - Appendix D, AVD-SLS has the best average score for 7 benchmark families.

For **WPMS 2017**: For 60 seconds time limit, AVD-SLS has the best number of best solutions and ranked as the best solver. For 300 seconds time limit, AVD-SLS ranked the third based on score measure results shown in Fig. 16. For family-based results as shown in Table 25 - Appendix D, AVD-SLS has the best average score for 2 benchmark families.

For **WPMS 2018**: AVD-SLS is a competitive solver with number of hybrid solvers, and is ranked here as the third best solver for 300 seconds time limit as shown in Fig. 17. While AVD-SLS is the best solver for 60 seconds time limit based on score measure and number of best solutions. For family-based results as shown in Table 26 - Appendix D, AVD-SLS has the best average score for 5 benchmark families.

For **WPMS 2019**: In this evaluation, no SLS solver was participated. A number of well-known hybrid solvers participated in this evaluation such as TT-Open-WBO-Inc, Loandra, LinSBPS2018, and Open-WBO solvers variants. However, AVD-SLS was able to compete with a number of solvers on the number of best solutions found as shown in Fig. 18. For family-based results as shown in Table 27 - Appendix D, AVD-SLS has the best average score for 5 benchmark families.

We conclude this experimental evaluation of AVD-SLS and PMSAT solvers that participated in MSE 2014-2019 on both PMS and WPMS with more than 3400 instances, that AVD-SLS is a competitive solver as shown in Fig.s 7 to

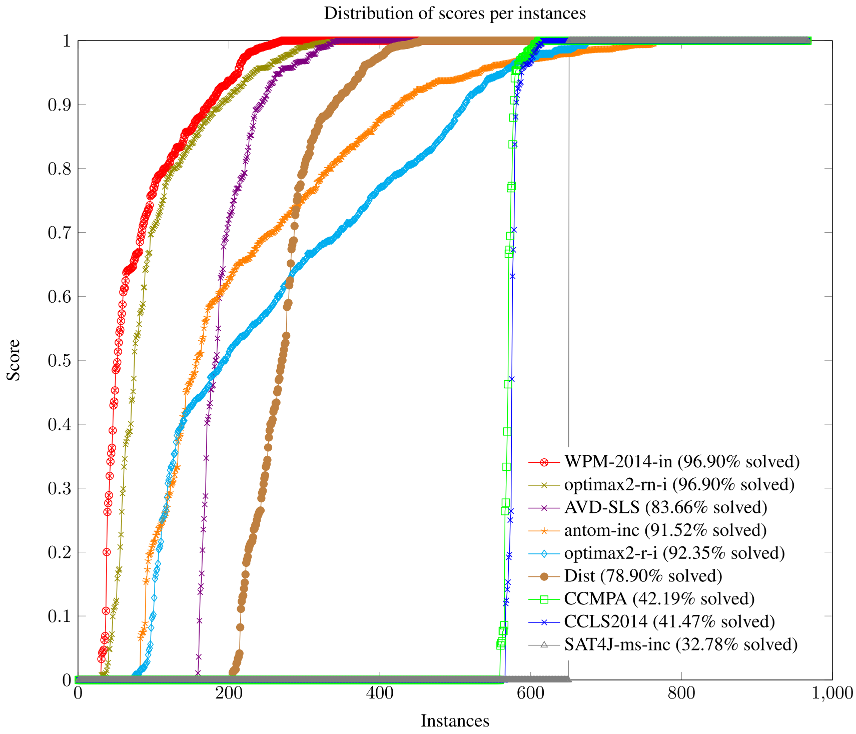

FIGURE 7. Distribution of scores for PMSAT solvers on MSE 2014 PMS (300s).

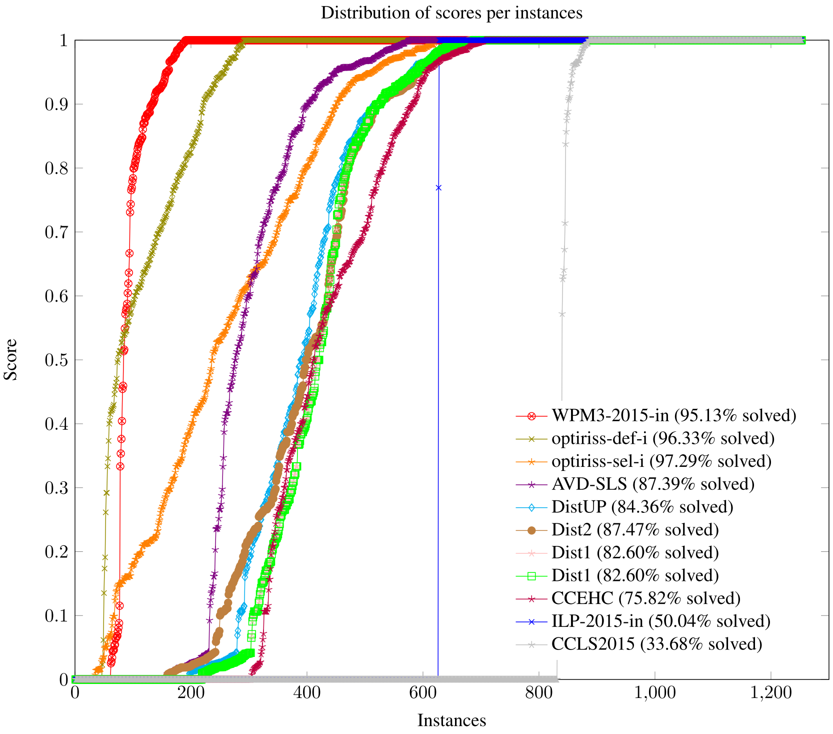

FIGURE 8. Distribution of scores for PMSAT solvers on MSE 2015 PMS (300s).

18 w.r.t evaluation criteria presented in this study which are adopted from MSE. AVD-SLS solver, improves the performance of SLS solvers especially on WPMS.

VI. DISCUSSION

The MaxSAT Evaluation events (MSE) added great value to the literature, where thousands of benchmark instances

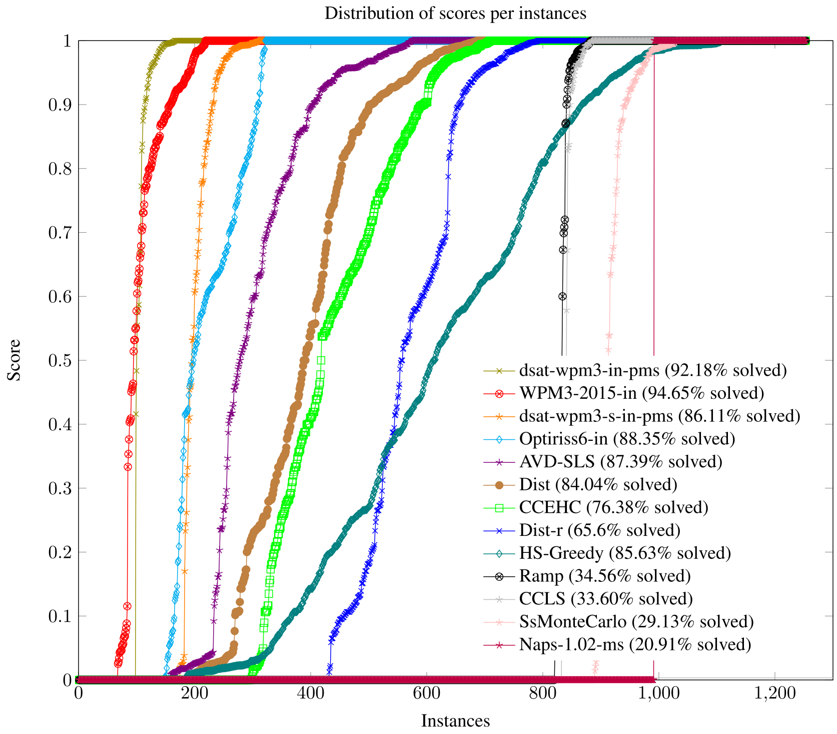

FIGURE 9. Distribution of scores for PMSAT solvers on MSE 2016 PMS (300s).

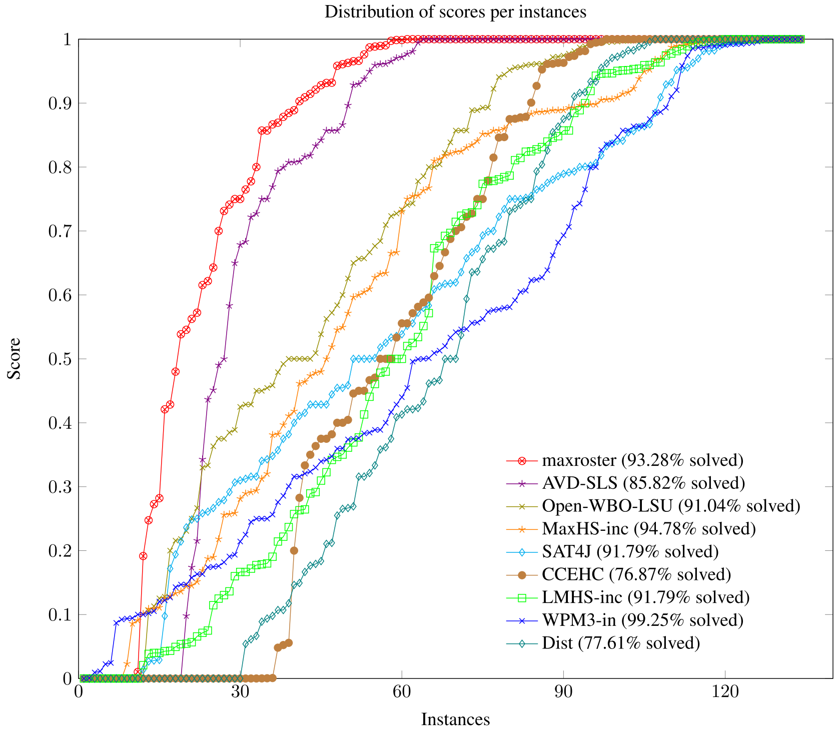

FIGURE 10. Distribution of scores for PMSAT solvers on MSE 2017 PMS (300s).

made available publicly. Problems from a large range of domains were encoded as Partial Max-SAT (PMSAT) problem, including hardware and software verification, optimization, graph theory, Satisfiability, automated reasoning, multi-agent solving, etc. Overall, 125 benchmark families (shown in Appendex C and D) were encoded from various

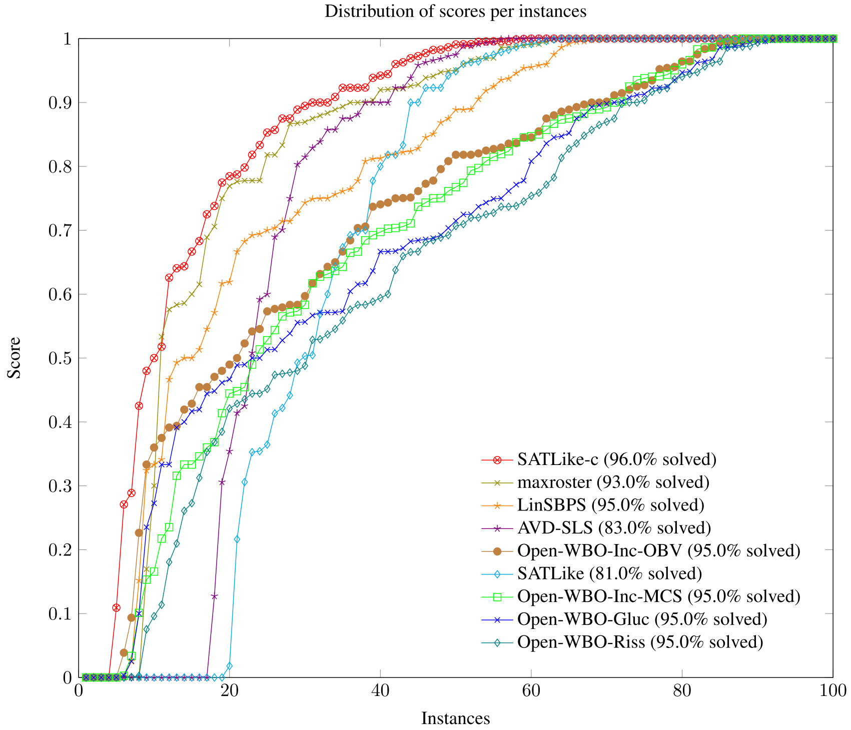

FIGURE 11. Distribution of scores for PMSAT solvers on MSE 2018 PMS (300s).

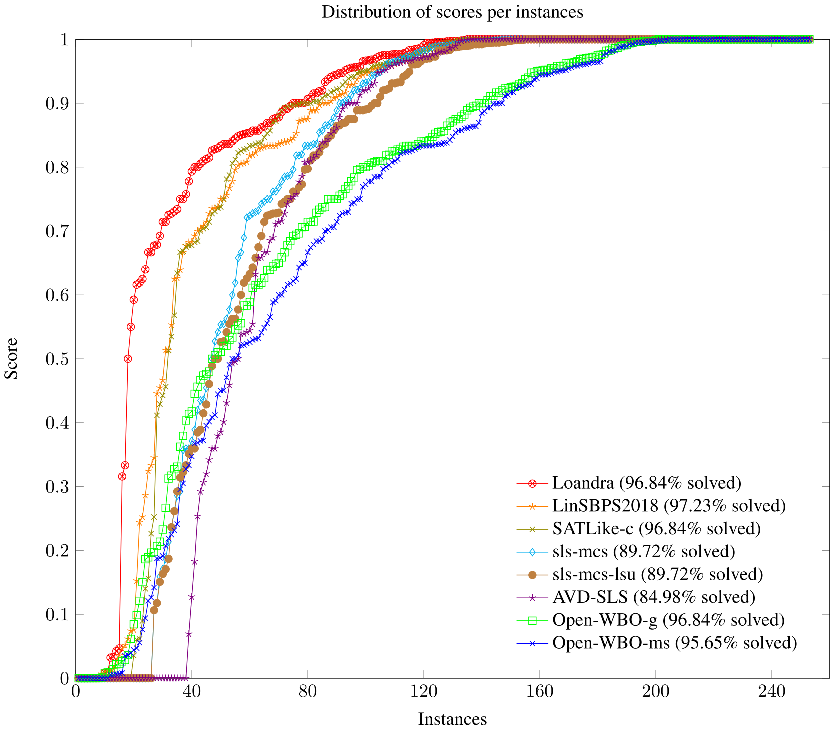

FIGURE 12. Distribution of scores for PMSAT solvers on MSE 2019 PMS (300s).

domains under three classes: random, crafted, and industrial instances.

Furthermore, the MSE represent the evolution of PMSAT solvers development, which can be traced and studied throughout the years. State-of-the-art solvers have shown great advancements in solving PMSAT problem instances. However, the evaluation results of MSE 2014-2019 show that many benchmarks instances related to weighted industrial

TABLE 12. Summary results of AVD-SLS with PMSAT solvers on MSE 2014-2016 WPMS.

	MSE 2014-2016 - Weighted Benchmark (300 seconds)			
	WPMS_random_2014 #inst. (280)	WPMS_crafted_2014 #inst. (302)	WPMS_industrial_2014 #inst. (402)	**WPMS_crafted_industrial_2014** #inst. (704)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**279** (119) - 0.688	298 **(195)** - 0.805	391 (173) - 0.823	689 (368) - 0.816
CCLS2014	278 (277) - 0.993	221 (137) - 0.695	71 (40) - 0.158	292 (177) - 0.389
CCMPA	278 (274) - 0.993	247 (144) - 0.717	76 (47) - 0.179	323 (191) - 0.409
Dist	**279 (279) - 0.996**	276 (180) - 0.819	236 (38) - 0.507	512 (218) - 0.641
SAT4J-ms-inc	0 (0) - 0.0	82 (82) - 0.272	66 (66) - 0.164	148 (148) - 0.210
WPM-2014-in	**279** (0) - 0.764	297 (154) - **0.883**	**399 (357) - 0.974**	**696 (511) - 0.935**
optimax2-g-i	**279** (0) - 0.680	298 (130) - 0.868	395 (253) - 0.939	693 (383) - 0.908
optimax2w-r-i	**279** (0) - 0.688	**299** (130) - 0.873	386 (190) - 0.850	685 (320) - 0.860
	WPMS_random_2015 #inst. (662)	WPMS_crafted_2015 #inst. (311)	WPMS_industrial_2015 #inst. (586)	**WPMS_crafted_industrial_2015** #inst. (897)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**661** (119) - 0.514	**307** (166) - 0.757	**573** (209) - 0.778	**880** (375) - **0.771**
CCEHC	**661** (659) - **0.998**	292 (183) - **0.873**	411 (74) - 0.477	703 (257) - 0.614
CCLS2015	660 **(660)** - 0.997	206 (131) - 0.630	73 (36) - 0.102	279 (167) - 0.285
Dist1	**661 (660) - 0.998**	284 (158) - 0.794	525 (47) - 0.520	809 (205) - 0.615
Dist2	**661 (660) - 0.998**	296 (158) - 0.828	524 (52) - 0.525	820 (210) - 0.630
ILP-2015-in	63 (63) - 0.095	189 **(189)** - 0.608	253 (251) - 0.430	442 (440) - 0.492
WPM3-2015-in	634 (10) - 0.800	232 (123) - 0.672	501 **(409) - 0.797**	733 **(532)** - 0.754
optiriss-def-i	**661** (0) - 0.685	-	438 (313) - 0.705	-
optiriss-sel-i	**661** (1) - 0.657	265 (57) - 0.723	438 (226) - 0.657	703 (283) - 0.680
	WPMS_random_2016 #inst. (502)	WPMS_crafted_2016 #inst. (319)	WPMS_industrial_2016 #inst. (606)	**WPMS_crafted_industrial_2016** #inst. (925)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	**501** (119) - 0.532	**315** (181) - 0.744	**593** (225) - 0.784	**908** (406) - 0.770
CCEHC	**501** (497) - **0.998**	302 (179) - **0.858**	434 (61) - 0.474	736 (240) - 0.606
CCLS	500 (498) - 0.996	216 (131) - 0.638	73 (42) - 0.103	289 (173) - 0.288
Dist	119 (119) - 0.237	204 (80) - 0.496	566 (61) - 0.618	770 (141) - 0.576
Dist-r	119 (119) - 0.237	199 (72) - 0.493	519 (66) - 0.504	718 (138) - 0.500
HS-Greedy	**501** (45) - 0.897	312 (67) - 0.744	566 (9) 0.376	878 (76) - 0.503
Ramp	500 (496) - 0.996	228 (134) - 0.648	84 (46) - 0.124	312 (180) 0.305
SC2016	**501** (500) - **0.998**	199 (137) - 0.595	102 (48) - 0.145	301 (185) - 0.300
Naps-1.02-ms	1 (1) - 0.002	80 (80) - 0.251	35 (35) - 0.058	115 (115) - 0.124
Optiriss6-in	**501** (0) - 0.680	280 (0) - 0.741	320 (222) - 0.483	600 (0) - 0.572
SsMonteCarlo	460 (458) - 0.916	208 (22) - 0.361	8 (6) - 0.012	216 (28) - 0.133
WPM3-2015-in	474 (8) - 0.783	258 (127) - 0.697	559 **(391) - 0.822**	817 (518) - **0.779**
dsat-wpm3-in-wpms	**501** (500) - **0.998**	259 **(196)** - 0.781	516 (390) - 0.771	776 **(586)** - 0.775
dsat-wpm3-s-in-wpms	**501 (501) 0.998**	281 (178) - 0.822	463 (329) - 0.681	744 (507) - 0.730

applications are still beyond the capacity of existing PMSAT SLS solvers. This present work is a contribution that aims to advance the development of PMSAT SLS solvers. Our proposed solver incorporates an adaptive parameter tuner and a variable depth neighborhood search (VDS) method adopted for solving PMSAT problem, which were combined with the dynamic local search solver SATLike. To the best of our knowledge, our proposed components have never adopted previously for PMSAT solving by SLS methods.

In this study, we evaluate the performance of our AVD-SLS solver that is implemented based on the proposed algorithm presented in Section IV. First, we evaluate the performance of AVD-SLS on all unweighted and weighted instances. Second, as the main goal of this paper, we compare the results obtained by AVD-SLS to those drawn by state-of-the-art SLS solvers that participated in the MSE 2014–2019. Finally, we compare the results of AVD-SLS to those obtained by all state-of-the-art PMSAT solvers that participated in MSE 2014-2019,

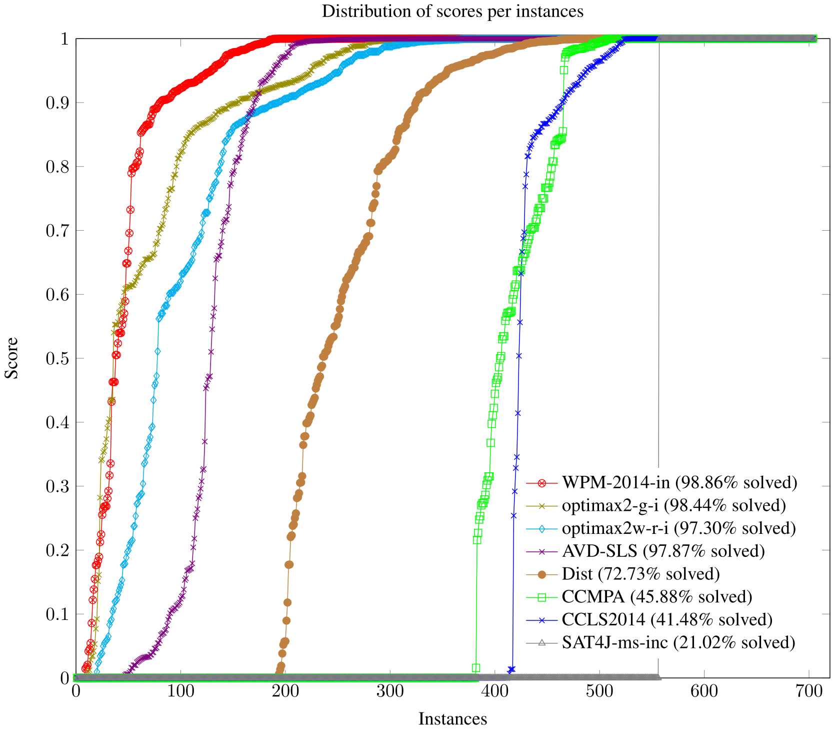

FIGURE 13. Distribution of scores for PMSAT solvers on MSE 2014 WPMS (300s).

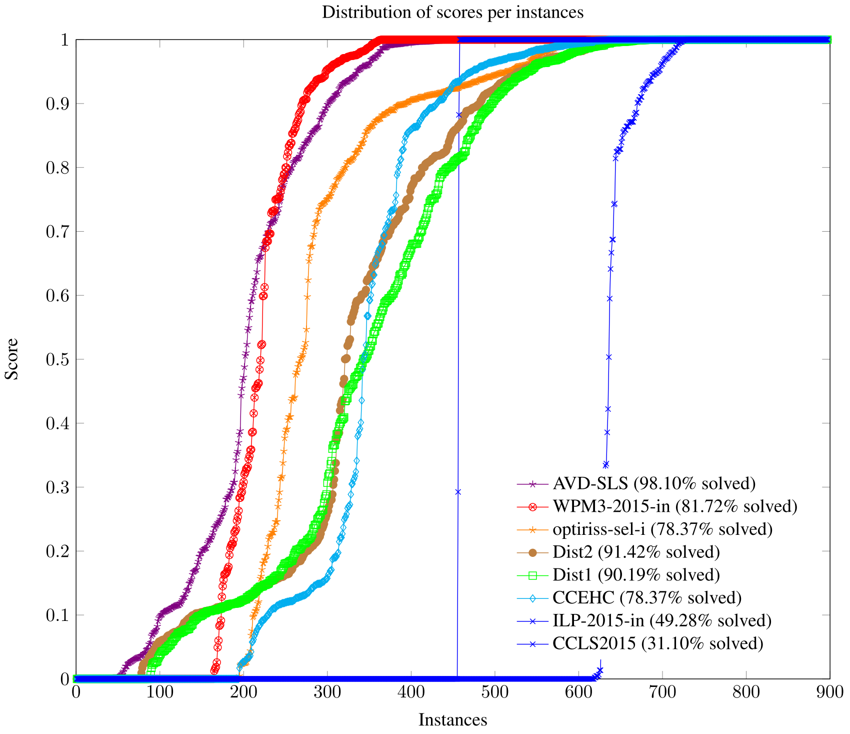

FIGURE 14. Distribution of scores for PMSAT solvers on MSE 2015 WPMS (300s).

including many well-known hybrid solvers. Almost all hybrid solvers are SAT-based solvers [18].

We classify the performance of the AVD-SLS solver in this study based on the percentage of solved instances per benchmark family into five different classes, as shown in Table 4 and Table 5: excellent (100%), very good (80-99%), good (66-79%), poor (51-65%), and fail (0-50%). We consider that AVD-SLS has failed to solve a benchmark family and

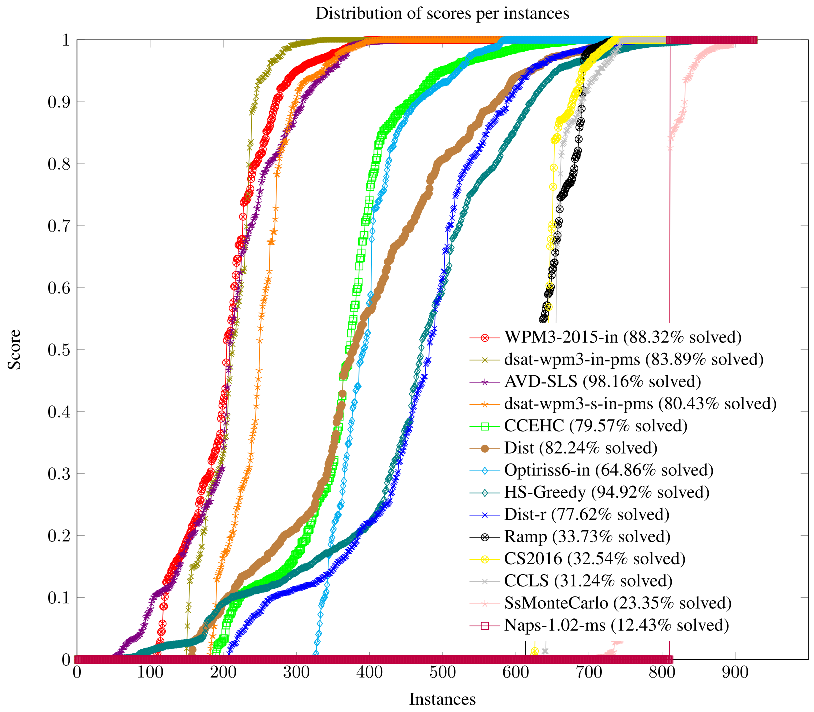

FIGURE 15. Distribution of scores for PMSAT solvers on MSE 2016 WPMS (300s).

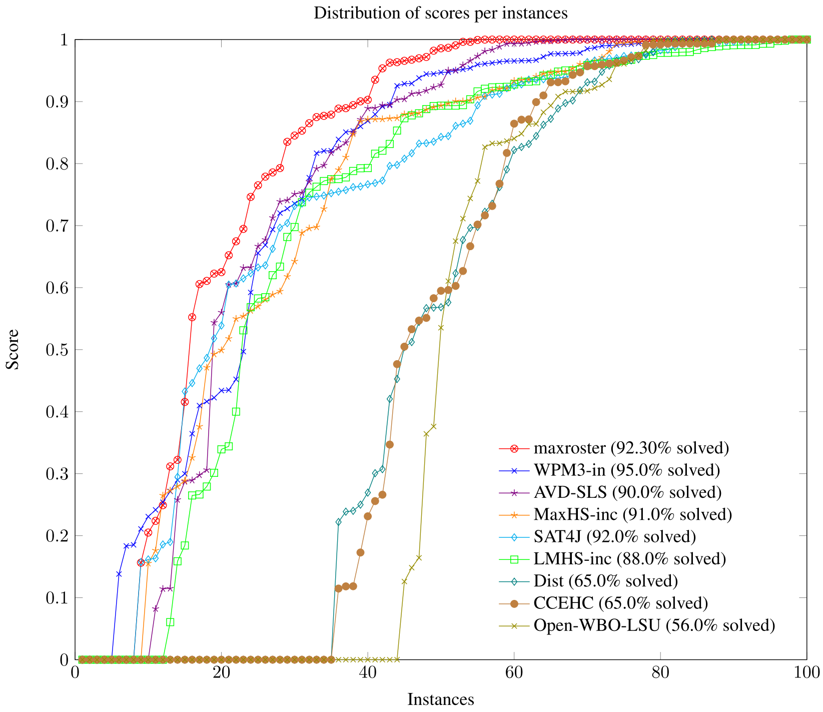

FIGURE 16. Distribution of scores for PMSAT solvers on MSE 2017 WPMS (300s).

categorize the benchmark family as hard if AVD-SLS is unable to solve more than 50% of the benchmark's instances for 300 CPU seconds.

A. DISCUSSION OF THE RESULTS ON PMS

AVD-SLS shows excellent performance on **random** instances like most SLS solvers, whereas the hybrid solvers were not competitive in this class.

For the **crafted** class, AVD-SLS is competitive with all PMSAT solvers with regards to the number of best solutions found. However, Dist and Dist2 SLS solvers solved a few more instances than AVD-SLS from MSE 2014 and 2015 PMS. Furthermore, based on our categorization of AVD-SLS performance, AVD-SLS shows excellent performance in 23 crafted families. Only one benchmark family is a hard family: reversi. We found that some reversi instances can

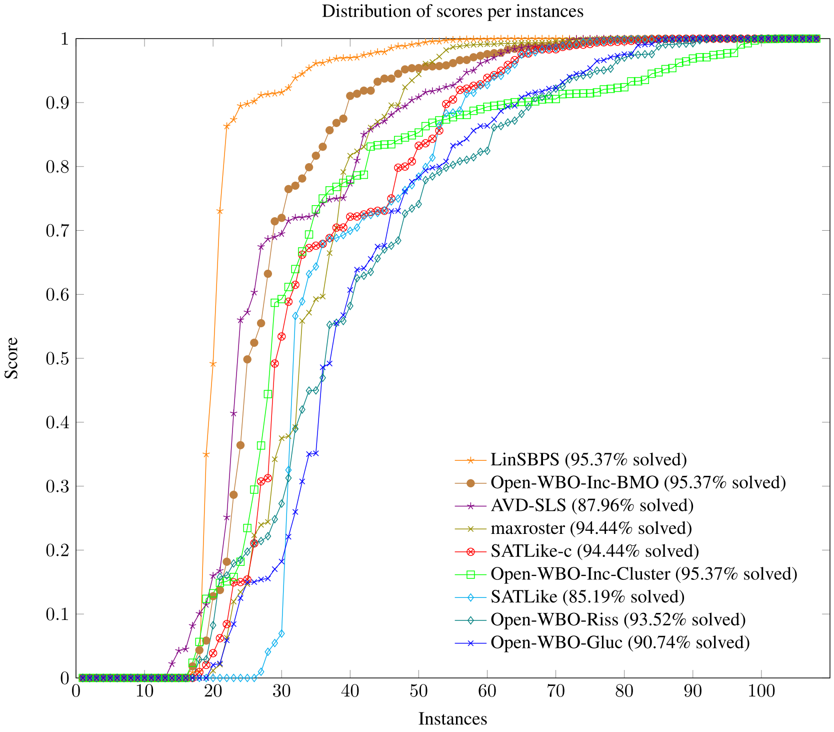

FIGURE 17. Distribution of scores for PMSAT solvers on MSE 2018 WPMS (300s).

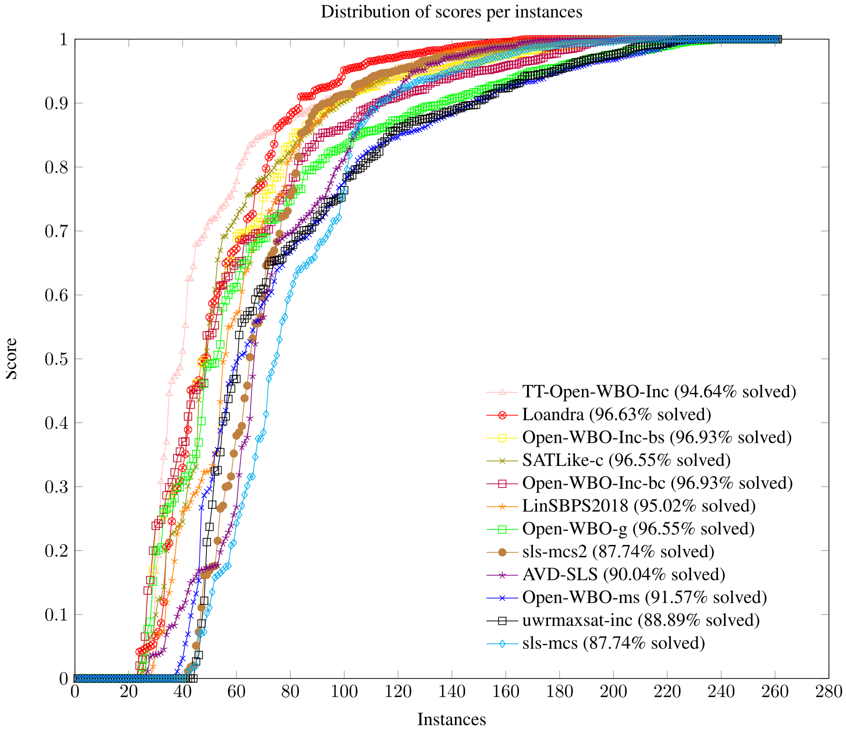

FIGURE 18. Distribution of scores for PMSAT solvers on MSE 2019 WPMS (300s).

be solved when the time limit is increased. However, we did not include the pseudo-Boolean-primes benchmark family as hard here, as it was never solved by any solver.

For the **industrial** class, AVD-SLS shows excellent performance in 16 benchmark families and is the best SLS solver on this class. Moreover, AVD-SLS is competitive with the

TABLE 13. Summary results of AVD-SLS with PMSAT SLS solvers on MSE 2017-2019 WPMS.

	MSE 2017-2019 - Weighted Benchmark	
	60 seconds	300 seconds
	WPMS_2017 #inst. (100)	**WPMS_2017** #inst. (100)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	89 **(39)** - **0.777**	90 (33) - 0.766
CCEHC	57 (14) - 0.468	65 (12) - 0.514
Dist	64 (13) - 0.506	65 (14) - 0.515
Open-WBO-LSU	90 (20) - 0.668	56 (22) - 0.480
MaxHS-inc	88 (7) - 0.633	88 (3) - 0.719
maxroster	**94** (28) - 0.761	92 **(45) - 0.809**
WPM3-in	**94** (8) - 0.721	**95** (12) - 0.772
SAT4J	93 (8) - 0.721	92 (6) - 0.742
LMHS-inc	92 (4) - 0.721	91 (13) - 0.744
	WPMS_2018 #inst. (108)	**WPMS_2018** #inst. (108)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	94 (19) - **0.717**	95 (32) - 0.733
SATLike	90 (8) - 0.617	92 (18) - 0.650
SATLike-c	99 (5) - 0.638	102 (18) - 0.680
LinSBPS	101**(26)** - 0.706	**103 (47) - 0.806**
Open-WBO-Inc-Cluster	**102** (4) - 0.661	**103** (6) - 0.678
Open-WBO-Inc-BMO	**102** (11) - 0.712	**103** (24) - 0.740
Open-WBO-Gluc	**102** (10) - 0.620	98 (19) - 0.624
Open-WBO-Riss	101 (9) - 0.595	101 (15) - 0.629
maxroster	99 (19) - 0.673	102 (32) - 0.691
	WPMS_2019 #nst. (261)	**WPMS_2019** #nst. (261)
Solver	**#sol. (#wins) - score**	**#sol. (#wins) - score**
AVD-SLS	233 (55) - 0.635	235 (78) - 0.721
TT-Open-WBO-Inc	241 (42) - **0.740**	247 (81) - **0.808**
Loandra	**248** (40) - 0.717	**253 (92)** - 0.794
LinSBPS2018	246 **(52)** - 0.676	248 (84) - 0.765
SATLike-c	240 (35) - 0.682	252 (60) - 0.782
Open-WBO-g	**248** (11) - 0.671	252 (25) - 0.747
sls-mcs2	212 (46) - 0.677	229 (69) - 0.734
sls-mcs	212 (46) - 0.677	229 (53) - 0.692
Open-WBO-ms	226 (13) - 0.617	239 (24) - 0.699
Open-WBO-inc-complete	247 (25) - 0.698	**253** (54) - 0.784
Open-WBO-inc-satlike	247 (31) - 0.704	**253** (56) - 0.770
uwrmaxsat-inc	209 (7) - 0.597	232 (36) - 0.699

PMSAT solvers with regards to the number of best solutions found. AVD-SLS have five hard benchmark families: atcoss-mesat, atcoss-sugar, circuit-compaction, des, and hs-timetabling. The hs-timetabling benchmark family only has two instances, and AVD-SLS solved one; we believe this result does not reflect the performance of AVD-SLS in this family. On the other hand, instances from atcoss-mest and circuit compaction families were never solved by any SLS solver. In atcoss-mesat, we found very large-sized instances with hundred thousands to millions of variables and clauses. Meanwhile, the circuit compaction instances have only a few thousands of variables and clauses, but had complex structures. Such hard instances may require more pre-processing techniques or structure extraction methods for them to be solved by SLS solvers. Most SLS solvers only consider unit propagation (UP) for pre-processing, as is the case for our solver AVD-SLS.

This evaluation study shows that AVD-SLS is the best solver among SLS solvers (Fig. 5 and Fig. 6) and that it is competitive with a number of well-known hybrid solvers, as shown in Fig.s 7 to 12. For example, AVD-SLS is among the top three solvers for MSE 2014 and MSE 2017. It is remarkable that for most of solved instances, AVD-SLS is able to find the best solution or near best solutions. AVD-SLS may be improved to solve more hard instances either by means of new pre-processing techniques [74], [75] to reduce

TABLE 14. A subset of PMS benchmark from MSE 2017-2018 used for the initial tuning of the parameters for our solver.

instance name	MSE 2014	MSE 2015	MSE 2016	MSE 2017	MSE 2018	MSE 2019
aes-mul_8_9.wcnf	✓	✓	✓	✓	✓	✓
aes-mul_8_13.wcnf	✓	✓	✓	✓	✓	✓
atcoss_mesat_04.wcnf	✓	✓	✓	✓	✓	✓
atcoss_mesat_15.wcnf	✓	✓	✓	✓	✓	✓
atcoss_sugar_04.wcnf	✓	✓	✓	✓	-	✓
atcoss_sugar_15.wcnf	✓	✓	✓	✓	-	✓
bcp-hipp-yRa1-SU3__simp-genos.haps.63.wcnf	✓	✓	✓	✓	-	-
bcp-msp-normalized-f1000.wcnf	✓	✓	✓	✓	✓	-
bcp-msp-normalized-f600.wcnf	✓	✓	✓	✓	✓	-
bcp-msp-normalized-g250.15.wcnf	✓	✓	✓	✓	✓	-
bcp-msp-normalized-ii16a2.wcnf	✓	✓	✓	✓	✓	✓
bcp-msp-normalized-par32-1.wcnf	✓	✓	✓	✓	✓	-
bcp-msp-normalized-par32-2-c.wcnf	✓	✓	✓	✓	-	-
bcp-syn-normalized-m200_500_10_10.r.wcnf	✓	✓	✓	✓	-	-
des-cnf.17.p.10.wcnf	✓	✓	✓	✓	-	✓
extension-enforcement_non-strict_stb_150_0.1_2_8_2.wcnf	-	-	-	✓	✓	✓
extension-enforcement_non-strict_stb_200_0.1_4_10_4.wcnf	-	-	-	✓	✓	-
extension-enforcement_non-strict_stb_200_0.05_1_10_1.wcnf	-	-	-	✓	✓	✓
extension-enforcement_non-strict_stb_200_0.05_3_10_0.wcnf	-	-	-	✓	✓	✓
extension-enforcement_non-strict_stb_200_0.05_4_10_1.wcnf	-	-	-	✓	✓	✓
extension-enforcement_non-strict_stb_200_0.05_4_10_4.wcnf	-	-	-	✓	✓	✓
fault-diagnosis-s38584_nan_explicit_15_0.wcnf	-	✓	✓	✓	-	-
fault-diagnosis-s38584_nan_explicit_5_0.wcnf	-	✓	✓	✓	✓	✓
gen-hyper-tw-GenHyperTW_aim-50-1_6-no-3.wcnf	-	-	-	✓	-	✓
gen-hyper-tw-GenHyperTW_aim-50-1_6-yes1-3.wcnf	-	-	-	-	✓	-
gen-hyper-tw-GenHyperTW_aim-50-3_4-yes1-3.wcnf	-	-	-	✓	-	-
gen-hyper-tw-GenHyperTW_dubois21.wcnf	-	-	-	✓	✓	-
gen-hyper-tw-GenHyperTW_dubois25.wcnf	-	-	-	✓	✓	-
gen-hyper-tw-GenHyperTW_dubois29.wcnf	-	-	-	✓	✓	-
gen-hyper-tw-GenHyperTW_flat30-1.wcnf	-	-	-	✓	✓	✓
gen-hyper-tw-GenHyperTW_grid4d_3.wcnf	-	-	-	✓	-	✓
gen-hyper-tw-GenHyperTW_hole8.wcnf	-	-	-	-	✓	-
gen-hyper-tw-GenHyperTW_par8-3-c.wcnf	-	-	-	✓	✓	-
gen-hyper-tw-GenHyperTW_pret60_25.wcnf	-	-	-	✓	-	-
gen-hyper-tw-GenHyperTW_s208.wcnf	-	-	-	✓	✓	-
maxclique-brock200_1.clq.wcnf	✓	✓	✓	✓	-	✓
maxclique-brock400_4.clq.wcnf	✓	✓	✓	✓	-	-
maxclique-p_hat1000-1.clq.wcnf	✓	✓	✓	✓	✓	✓
maxclique-p_hat700-1.clq.wcnf	✓	✓	✓	✓	-	✓
maxcut-brock400_2.clq.wcnf	-	-	-	✓	✓	✓
maxcut-brock800_3.clq.wcnf	-	-	-	✓	-	-
maxcut-hamming8-4.clq.wcnf	-	-	-	✓	-	-
maxcut-p_hat500-3.clq.wcnf	-	-	-	✓	✓	-
maxcut-san400_0.5_1.clq.wcnf	-	-	-	✓	-	-
maxcut-san400_0.7_3.clq.wcnf	-	-	-	✓	-	-
maxcut-sanr200_0.7.clq.wcnf	-	-	-	✓	-	-
maxcut-t7pm3-9999.spn.wcnf	-	-	-	✓	-	-
min-fill-MinFill_R0_mulsol.i.1.wcnf	-	-	-	✓	-	-
min-fill-MinFill_R0_mulsol.i.4.wcnf	-	-	-	✓	-	-
min-fill-MinFill_R0_myciel6.wcnf	-	-	-	✓	✓	✓
min-fill-MinFill_R0_queen11_11.wcnf	-	-	-	✓	✓	✓
min-fill-MinFill_R0_queen8_8.wcnf	-	-	-	✓	✓	✓
min-fill-MinFill_R3_miles750.wcnf	-	-	-	✓	✓	✓
optic-gen_mult_3_6_9999.wcnf	-	-	-	-	✓	✓
optic-gen_mult_4_5_399.wcnf	-	-	-	-	✓	✓
optic-gen_mult_4_5_9999.wcnf	-	-	-	-	✓	✓
optic-gen_mult_4_7_991.wcnf	-	-	-	-	✓	✓
rev66-16.wcnf	✓	✓	✓	✓	✓	✓
rev66-26.wcnf	✓	✓	✓	✓	✓	✓
scheduling-cnf_12.wcnf	✓	✓	✓	✓	✓	✓
sean-safarpour-rsdecoder-problem.dimacs_38.filtered.wcnf	-	-	-	✓	-	✓
sean-safarpour-wb_4m8s4.dimacs.filtered.wcnf	-	-	-	✓	✓	✓
set-covering-scpclr13_maxsat.wcnf	-	-	-	✓	✓	✓
set-covering-scpcyc09_maxsat.wcnf	-	-	-	✓	✓	✓
treewidth-computation-TWComp_1c75_N69.wcnf	-	✓	✓	✓	✓	✓
treewidth-computation-TWComp_queen5_5_N25.wcnf	-	✓	✓	✓	✓	✓
uaq-uaq-ppr-nr200-nc66-n5-k2-rpp4-ppr6-plb100.wcnf	-	-	-	-	✓	-
uaq-uaq-ppr-nr200-nc66-n5-k2-rpp4-ppr8-plb100.wcnf	-	-	-	-	✓	-
uaq-uaq-ppr-nr200-nc66-n5-k2-rpp6-ppr6-plb100.wcnf	-	-	-	-	✓	✓
uaq-uaq-ppr-nr200-nc66-n5-k2-rpp6-ppr13-plb100.wcnf	-	-	-	-	✓	✓
xai-mindset-bnn-last-layer-f10.wcnf	-	-	-	-	✓	-
xai-mindset-cleveland.wcnf	-	-	-	-	✓	✓
xai-mindset-ecoli.wcnf	-	-	-	-	✓	✓
xai-mindset-heart-statlog.wcnf	-	-	-	-	✓	✓
xai-mindset-liver-disorder.wcnf	-	-	-	-	✓	✓
xai-mindset-shuttleM.wcnf	-	-	-	-	✓	✓
Total	22	26	26	60	53	46
%	2.22%	2.03%	2.03%	30.93%	34.64%	15.38%

TABLE 15. A subset of WPMS benchmark from MSE 2017-2018 used for the initial tuning of the parameters for our solver.

instance name	MSE 2014	MSE 2015	MSE 2016	MSE 2017	MSE 2018	MSE 2019
af-synthesis-af-synthesis_stb_50_120_2.wcnf	-	-	-	-	✓	-
af-synthesis-af-synthesis_stb_50_120_7.wcnf	-	-	-	-	✓	✓
af-synthesis-af-synthesis_stb_50_140_0.wcnf	-	-	-	-	✓	✓
af-synthesis-af-synthesis_stb_50_140_5.wcnf	-	-	-	-	✓	-
af-synthesis-af-synthesis_stb_50_140_8.wcnf	-	-	-	-	✓	-
af-synthesis-af-synthesis_stb_50_180_2.wcnf	-	-	-	-	✓	✓
af-synthesis-af-synthesis_stb_50_200_5.wcnf	-	-	-	-	✓	-
af-synthesis-af-synthesis_stb_50_60_2.wcnf	-	-	-	-	✓	✓
BTBNSL_Rounded_BTWBNSL_AbaloneTWBound_4.wcnf	-	✓	✓	-	✓	-
BTBNSL_Rounded_BTWBNSL_adult15N_TWBound_4.wcnf	-	✓	✓	✓	-	-
BTBNSL_Rounded_BTWBNSL_alarm_100_1_3.scores_TWBound_4.wcnf	-	✓	✓	✓	✓	✓
BTBNSL_Rounded_BTWBNSL_Flag.BIC_TWBound_4.wcnf	-	✓	✓	-	✓	-
BTBNSL_Rounded_BTWBNSL_hailfinder_10000_1_3.scores_TWBound_3.wcnf	-	✓	✓	✓	✓	-
BTBNSL_Rounded_BTWBNSL_Horse.BIC_TWBound_3.wcnf	-	✓	✓	✓	✓	✓
BTBNSL_Rounded_BTWBNSL_Housing_TWBound_3.wcnf	-	✓	✓	✓	✓	-
BTBNSL_Rounded_BTWBNSL_insurance_1000_1_3.scores_TWBound_4.wcnf	-	✓	✓	✓	-	-
BTBNSL_Rounded_BTWBNSL_Water_1000_1_2.scores_TWBound_4.wcnf	-	✓	✓	-	✓	-
causal-discovery-causal_Autos_8_159.wcnf	-	-	-	✓	✓	✓
causal-discovery-causal_Image_7_2310.wcnf	-	-	-	✓	✓	-
causal-discovery-causal_Pigs_6_1000.wcnf	-	-	-	-	✓	-
cluster-expansion-IS1_5.0.5.0.0.5_softer_periodic.wcnf	-	-	-	-	✓	-
cluster-expansion-IS2_5.0.5.0.0.5_softer_periodic.wcnf	-	-	-	-	✓	-
cluster-expansion-IS3_5.0.5.0.0.5_softer_periodic.wcnf	-	-	-	-	✓	-
cluster-expansion-IS5_5.0.5.0.0.5_softer_periodic.wcnf	-	-	-	-	✓	-
cluster-expansion-IS8_5.0.5.0.0.5_softer_periodic.wcnf	-	-	-	-	✓	-
cluster-expansion-IS11_5.0.5.0.0.5_softer_periodic.wcnf	-	-	-	-	✓	-
cluster-expansion-IS14_5.0.5.0.0.5_softer_periodic.wcnf	-	-	-	-	✓	-
cluster-expansion-IS17_5.0.5.0.0.5_softer_periodic.wcnf	-	-	-	-	✓	-
CorrelationClustering_Rounded_CorrelationClustering_Ecoli_BINARY_N260_D0.200.wcnf	-	✓	✓	✓	✓	-
CorrelationClustering_Rounded_CorrelationClustering_Ionosphere_BINARY_N220_D0.200.wcnf	-	✓	✓	✓	-	-
CorrelationClustering_Rounded_CorrelationClustering_Protein2_BINARY_N340.wcnf	-	✓	✓	✓	✓	✓
CorrelationClustering_Rounded_CorrelationClustering_Protein4_BINARY_N360.wcnf	-	✓	✓	✓	✓	-
CorrelationClustering_Rounded_CorrelationClustering_Vowel_BINARY_N800_D0.200.wcnf	-	✓	✓	✓	✓	-
hs-timetabling_BrazilInstance2.xml.wcnf	✓	✓	✓	✓	✓	✓
hs-timetabling_BrazilInstance5.xml.wcnf	✓	✓	✓	✓	✓	✓
hs-timetabling_FinlandCollege.xml.wcnf	✓	✓	✓	✓	✓	-
hs-timetabling_ItalyInstance4.xml.wcnf	✓	✓	✓	✓	✓	✓
lisbon-wedding-10-19.wcnf	-	-	-	✓	✓	✓
lisbon-wedding-1-19.wcnf	-	-	-	-	✓	✓
lisbon-wedding-3-18.wcnf	-	-	-	-	✓	✓
lisbon-wedding-4-19.wcnf	-	-	-	✓	-	✓
lisbon-wedding-7-17.wcnf	-	-	-	✓	✓	✓
lisbon-wedding-8-17.wcnf	-	-	-	✓	✓	-
lisbon-wedding-8-19.wcnf	-	-	-	✓	✓	✓
lisbon-wedding-9-18.wcnf	-	-	-	✓	-	-
maxcut_hamming8-2.clq.wcnf	✓	✓	✓	✓	✓	✓
maxcut_MANN_a81.clq.wcnf	✓	✓	✓	✓	✓	✓
maxcut_san200_0.7_2.clq.wcnf	✓	✓	✓	✓	-	✓
maxcut_san200_0.9_3.clq.wcnf	✓	✓	✓	✓	✓	✓
maxcut_sanr200_0.9.clq.wcnf	✓	✓	✓	✓	✓	✓
maxcut_t7g3-9999.spn.wcnf	✓	✓	✓	✓	✓	-
MinWidthCB_milan_200_12_1k_10s_1t_12.wcnf	-	-	-	✓	✓	-
MinWidthCB_mitdbsample_300_43_1k_3s_2t_3.wcnf	-	-	-	✓	-	-
MinWidthCB_mitdbsample_300_43_1k_6s_1t_6.wcnf	-	-	-	✓	-	-
MinWidthCB_mitdbsample_300_43_1k_6s_2t_8.wcnf	-	-	-	✓	✓	-
MinWidthCB_mitdbsample_300_43_1k_15s_2t_15.wcnf	-	-	-	✓	-	-
MinWidthCB_mitdbsample_300_64_1k_3s_2t_5.wcnf	-	-	-	✓	-	-
MinWidthCB_mitdbsample_300_64_1k_3s_3t_5.wcnf	-	-	-	✓	✓	-
MinWidthCB_mitdbsample_300_64_1k_6s_3t_6.wcnf	-	-	-	✓	✓	-
MinWidthCB_mitdbsample_300_64_1k_15s_2t_15.wcnf	-	-	-	✓	✓	-
MinWidthCB_power_1000_24_1k_10s_1t_12.wcnf	-	-	-	✓	✓	✓
MinWidthCB_power_1000_24_1k_20s_2t_22.wcnf	-	-	-	✓	✓	✓
MinWidthCB_power_1000_24_1k_50s_2t_50.wcnf	-	-	-	✓	-	-
miplib_normalized-mps-v2-20-10-mod008.opb.msat.wcnf	✓	✓	✓	✓	✓	✓
miplib_normalized-mps-v2-20-10-sentoy.opb.msat.wcnf	✓	✓	✓	✓	✓	✓
railway-transport_d4.wcnf	-	✓	✓	✓	-	-
railway-transport_r11.wcnf	-	✓	✓	✓	✓	✓
rna-alignment_k100-13-99.rna.pre.wcnf	-	-	-	✓	-	-
rna-alignment_k100-39-93.rna.pre.wcnf	-	-	-	✓	-	-
robot-nagivation-robot-navigation_8.wcnf	-	-	-	-	✓	✓
Spot5_1403.wcsp.log.wcnf	✓	✓	✓	✓	✓	✓
staff-scheduling_instance3.wcnf	-	-	✓	✓	✓	✓
staff-scheduling_instance6.wcnf	-	-	✓	✓	✓	✓
staff-scheduling_instance9.wcnf	-	-	✓	✓	✓	✓
staff-scheduling_instance11.wcnf	-	-	✓	✓	✓	✓
tcp-tcp_students_98_it_11.wcnf	-	-	-	-	✓	✓
tcp-tcp_students_105_it_6.wcnf	-	-	-	-	✓	✓
timetabling_comp03.wcnf	✓	✓	✓	✓	-	-
timetabling_comp12.wcnf	✓	✓	✓	✓	-	✓
timetabling_comp21.wcnf	✓	✓	✓	✓	-	-
timetabling_comp16.lp.sm-extracted.wcnf	-	-	-	-	✓	-
Total	16	32	36	55	64	36
%	2.22%	3.44%	3.75%	35.26%	37.21%	12.12%

TABLE 16. **Detailed results of AVD-SLS and state-of-the-art SLS and hybrid solvers on PMS_2014 (crafted and industrial) benchmark - 300 seconds.**

solver		CCLS2014		CCMPA		Dist		SAT4J-ms-inc		WPM-2014-in		antom-inc		optimax2-r-i		optimax2-rn-i		avdsls	
benchmark family	#Ins.	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score
frb	25	25	1.000	25	1.000	25	0.999	0	0.000	25	0.999	25	0.980	22	0.880	25	0.999	25	1.000
job-shop	3	0	0.000	0	0.000	3	0.217	1	0.333	3	1.000	3	1.000	2	0.667	3	0.980	3	0.235
maxclique-random	96	96	1.000	96	1.000	96	1.000	60	0.625	96	0.996	96	0.972	96	0.967	96	0.997	96	0.998
maxclique-structured	58	58	0.979	58	0.980	58	0.980	14	0.241	58	0.971	58	0.942	45	0.747	58	0.971	58	0.999
maxone-3sat	80	78	0.975	74	0.925	80	1.000	30	0.375	80	0.987	80	0.774	80	0.716	80	0.981	79	0.988
maxone-structured	60	3	0.050	1	0.017	51	0.850	60	1.000	60	1.000	60	0.981	54	0.858	60	1.000	49	0.816
min-enc-kbtree	42	42	1.000	42	0.989	42	1.000	0	0.000	42	0.828	42	0.655	42	0.636	42	0.764	42	1.000
pseudo-miplib	4	4	1.000	4	1.000	4	1.000	3	0.750	4	1.000	4	1.000	4	0.978	4	1.000	4	1.000
reversi	42	1	0.024	5	0.103	14	0.262	32	0.762	42	0.880	35	0.819	42	0.852	42	1.000	15	0.340
scheduling	4	0	0.000	0	0.000	4	0.697	0	0.000	4	0.688	3	0.685	3	0.299	4	0.957	4	0.779
aes	5	1	0.200	4	0.800	1	0.200	0	0.000	4	0.495	4	0.201	5	0.593	4	0.530	5	0.993
atcoss-mesat	16	0	0.000	0	0.000	0	0.000	0	0.000	14	0.875	2	0.125	6	0.138	5	0.138	0	0.000
atcoss-sugar	17	0	0.000	0	0.000	8	0.105	4	0.235	16	0.865	8	0.471	11	0.170	12	0.250	6	0.199
bcp-fir	32	29	0.887	32	0.919	22	0.656	0	0.000	26	0.801	31	0.654	32	0.903	32	0.988	32	0.993
bcp-hipp-yRa1-simp	9	0	0.000	1	0.100	10	1.000	4	0.400	10	0.998	10	0.980	10	0.836	10	1.000	10	0.768
bcp-hipp-yRa1-su	37	0	0.000	1	0.013	37	0.996	7	0.189	37	0.989	37	0.955	37	0.797	37	1.000	37	0.898
bcp-msp	34	0	0.000	0	0.000	27	0.790	3	0.088	28	0.754	28	0.696	16	0.391	28	0.809	29	0.847
bcp-mtg	30	0	0.000	0	0.000	30	0.860	21	0.700	30	1.000	30	0.897	30	0.717	30	1.000	29	0.886
bcp-syn	38	30	0.704	37	0.769	34	0.617	1	0.000	36	0.641	37	0.500	36	0.556	37	0.641	37	1.000
circuit-trace-compaction	4	0	0.000	0	0.000	0	0.000	2	0.500	4	1.000	4	1.000	4	0.857	4	1.000	0	0.000
close-solutions	50	16	0.320	9	0.180	27	0.516	13	0.260	41	0.772	18	0.218	37	0.567	43	0.503	36	0.713
des	49	0	0.000	1	0.020	12	0.214	0	0.000	45	0.871	48	0.824	49	0.793	49	0.986	5	0.087
haplotype-assembly	6	5	0.131	6	0.124	1	0.129	0	0.000	6	0.924	6	0.489	6	0.904	6	0.457	6	0.992
hs-timetabling	2	0	0.000	0	0.000	0	0.000	0	0.000	2	0.726	0	0.000	2	0.814	2	0.444	1	0.500
mbd	46	0	0.000	0	0.000	33	0.246	0	0.000	46	0.968	46	0.286	46	0.959	46	0.886	46	0.873
packup-pms	40	0	0.000	0	0.000	40	0.988	5	0.125	40	1.000	40	0.970	40	0.987	40	1.000	40	0.999
pbo-mqc-nencdr	25	0	0.000	0	0.000	4	0.056	16	0.640	25	0.847	25	0.855	25	0.319	25	0.824	15	0.225
pbo-mqc-nlogencdr	25	0	0.000	0	0.000	24	0.445	25	1.000	25	1.000	25	0.847	25	0.643	25	1.000	25	0.991
pbo-routing	15	1	0.067	0	0.000	15	0.991	12	0.800	15	1.000	15	0.945	15	1.000	15	1.000	14	0.933
protein-ins	12	12	0.882	12	0.911	12	0.923	4	0.308	13	0.940	13	0.968	13	0.610	13	0.941	12	0.882
tpr-Multiple-path	36	0	0.000	0	0.000	25	0.592	0	0.000	36	0.797	28	0.737	34	0.530	36	0.955	25	0.675
tpr-One-path	25	0	0.000	0	0.000	24	0.907	0	0.000	24	0.997	24	0.946	24	0.469	24	0.982	24	0.963
32 families	967	401	0.288	408	0.308	763	0.601	317	0.292	937	0.894	885	0.730	893	0.692	937	0.843	809	0.737

TABLE 17. **Detailed results of AVD-SLS and state-of-the-art SLS and hybrid solvers on PMS_2015 (crafted and industrial) benchmark - 300 seconds.**

solver		CCEHC		CCLS2015		Dist1		Dist2		DistUP		ILP-2015-in		WPM3-2015-in		optiriss-def-i		optiriss-sel-i		avdsls	
benchmark family	#Ins.	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score
aes-key-recovery	76	0	0.000	0	0.000	76	0.061	76	0.059	73	0.084	2	0.026	76	0.797	76	0.968	76	0.980	76	0.086
causal-discovery	81	81	0.499	0	0.000	30	0.133	71	0.311	30	0.146	0	0.000	81	0.992	79	0.884	81	0.695	81	0.614
fault-diagnosis	98	97	0.880	0	0.000	98	0.941	98	0.936	98	0.929	50	0.510	98	0.997	91	0.917	95	0.921	98	0.970
frb	25	25	1.000	25	1.000	25	0.998	25	1.000	25	0.999	10	0.400	8	0.319	25	0.998	22	0.820	25	1.000
job-shop	3	3	0.302	0	0.000	3	0.217	3	0.217	3	0.217	0	0.000	3	1.000	3	1.000	3	1.000	3	0.235
maxclique-random	96	96	1.000	96	1.000	96	1.000	96	1.000	96	1.000	96	1.000	96	1.000	93	0.947	95	0.960	96	0.998
maxclique-structured	58	58	0.980	58	0.979	58	0.980	58	0.981	58	0.978	32	0.552	35	0.602	44	0.729	50	0.822	58	0.999
maxone-3sat	80	79	0.988	78	0.975	79	0.988	80	1.000	80	1.000	80	1.000	80	1.000	80	0.956	80	0.674	79	0.988
maxone-structured	60	47	0.783	4	0.066	52	0.866	56	0.933	53	0.883	58	0.967	60	1.000	60	0.997	53	0.832	49	0.816
min-enc-kbtree	42	42	0.977	42	0.977	42	0.977	42	0.977	42	0.977	42	0.977	42	0.906	40	0.672	42	0.517	42	0.977
pseudo-miplib	4	4	1.000	4	1.000	4	1.000	4	1.000	4	1.000	4	1.000	4	1.000	4	1.000	4	0.939	4	1.000
reversi	42	12	0.282	1	0.024	15	0.305	19	0.411	17	0.375	15	0.357	42	0.972	42	0.952	42	0.944	15	0.340
scheduling	4	4	0.858	0	0.000	4	0.875	4	0.842	4	0.832	0	0.000	4	0.917	4	0.545	4	0.438	4	0.919
aes	5	1	0.200	1	0.200	2	0.400	3	0.495	4	0.800	1	0.200	5	0.683	5	0.654	5	0.654	5	0.993
atcoss-mesat	16	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	13	0.808	14	0.818	14	0.595	0	0.000
atcoss-sugar	17	11	0.247	0	0.000	7	0.084	7	0.057	8	0.083	0	0.000	16	0.867	16	0.838	16	0.763	6	0.187
bcp-fir	32	26	0.710	29	0.892	30	0.887	30	0.880	32	0.948	32	1.000	32	0.997	32	0.983	32	0.965	32	0.981
bcp-hipp-yRa1-simp	10	10	0.995	0	0.000	10	1.000	10	0.993	10	0.987	0	0.000	10	0.997	10	0.984	10	0.984	10	0.768
bcp-hipp-yRa1-su	38	37	0.953	0	0.000	37	0.996	37	0.997	37	0.997	0	0.000	37	0.989	37	0.961	37	0.945	37	0.898
bcp-msp	33	24	0.695	0	0.000	27	0.788	28	0.819	28	0.821	19	0.559	18	0.519	24	0.614	26	0.641	29	0.847
bcp-mtg	30	30	0.907	0	0.000	30	0.846	29	0.698	30	0.889	16	0.533	30	1.000	30	1.000	30	1.000	29	0.886
bcp-syn	37	35	0.919	32	0.767	37	0.924	37	0.926	37	0.919	35	0.940	36	0.902	35	0.748	37	0.809	37	0.998
circuit-trace-compaction	4	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	1	0.250	4	1.000	4	1.000	4	0.995	0	0.000
close-solutions	50	29	0.449	16	0.320	31	0.618	32	0.614	38	0.727	23	0.460	50	1.000	50	0.944	49	0.724	36	0.713
des	49	3	0.050	0	0.000	15	0.257	13	0.230	16	0.274	10	0.204	49	0.987	49	0.941	49	0.837	5	0.087
haplotype-assembly	6	4	0.337	5	0.133	6	0.576	6	0.310	6	0.559	3	0.500	6	0.930	6	0.904	6	0.914	6	0.992
hs-timetabling	2	0	0.000	0	0.000	0	0.000	1	0.419	0	0.000	0	0.000	2	0.688	2	0.723	2	0.671	1	0.500
mbd	46	24	0.166	0	0.000	46	0.306	46	0.312	46	0.344	15	0.326	46	0.997	46	1.000	46	0.951	46	0.855
packup-pms	40	39	0.960	0	0.000	40	0.989	40	0.984	40	0.990	40	1.000	40	1.000	40	1.000	40	1.000	40	0.999
pbo-mqc-nencdr	25	5	0.048	0	0.000	4	0.057	15	0.172	10	0.104	1	0.040	25	1.000	25	0.910	25	0.269	15	0.225
pbo-mqc-nlogencdr	25	24	0.246	0	0.000	24	0.507	23	0.412	25	0.387	0	0.000	25	1.000	25	1.000	25	0.706	25	0.991
pbo-routing	15	8	0.527	1	0.063	15	0.997	15	0.998	15	0.997	15	1.000	15	1.000	15	1.000	15	1.000	14	0.933
protein-ins	12	12	1.000	12	0.956	12	1.000	12	1.000	12	1.000	1	0.083	12	1.000	11	0.885	12	0.601	12	0.955
tpr-Multiple-path	36	24	0.526	0	0.000	24	0.559	24	0.569	24	0.557	1	0.028	36	0.988	34	0.876	36	0.224	24	0.635
tpr-One-path	25	25	0.658	0	0.000	25	0.900	25	0.911	25	0.911	25	1.000	25	1.000	25	1.000	25	0.209	25	0.963
treewidth-computation	31	31	1.000	18	0.570	31	0.256	31	0.271	31	0.389	0	0.000	31	0.927	31	0.907	31	0.799	31	0.901
36 families	1253	950	0.587	422	0.276	1035	0.619	1096	0.631	1057	0.642	627	0.414	1192	0.911	1207	0.896	1219	0.772	1095	0.729

the dimensionality of large-size instances, by extracting complex structure such as Boolean gates of input CNF formula F [76], or by using both. Almost all complete and hybrid PMSAT solver incorporate many pre- or co-processing techniques to reduce the dimensionality of large-sized instances.

B. DISCUSSION OF THE RESULTS ON WPMS

AVD-SLS shows excellent performance on **random** instances like most SLS solvers, with regards to number of solved instances, but is able to find only few best solutions. Only few hybrid solvers were able to solve all random instances.

For the **crafted** class, AVD-SLS perform competitively with all PMSAT solvers with regards to the number of best solutions found and is the best SLS solver on this class. Furthermore, based on our categorization of AVD-SLS' performance, it demonstrates excellent performance in 23 crafted families. In fact, only one benchmark family is hard: miplib.

TABLE 18. Detailed results of AVD-SLS and state-of-the-art SLS and hybrid solvers on PMS_2016 (crafted and industrial) benchmark - 300 seconds.

solver		CCEHC		CCLS		Dist		Dist-r		HS-Greedy		Naps-1.02-ms		Optiriss6-in		Ramp		SsMonteCarlo		WPM3-2015-in		dsat-wpm3-in-pms		dsat-wpm3-s-in-pms		avdsls	
benchmark family	#Ins.	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score
aes-key-recovery	76	0	0.000	0	0.000	64	0.125	0	0.000	76	0.040	0	0.000	76	0.970	0	0.000	0	0.000	76	0.785	73	0.961	64	0.814	76	0.076
causal-discovery	81	81	0.492	0	0.000	44	0.220	30	0.132	81	0.370	0	0.000	79	0.860	0	0.000	0	0.000	81	0.982	78	0.949	71	0.813	81	0.606
fault-diagnosis	98	95	0.855	0	0.000	98	0.952	8	0.067	95	0.755	0	0.000	92	0.927	0	0.000	0	0.000	98	0.985	97	0.984	83	0.839	98	0.967
frb	25	25	1.000	25	1.000	25	1.000	25	0.999	25	0.994	20	0.800	25	0.997	25	1.000	20	0.793	8	0.319	23	0.917	25	0.994	25	1.000
job-shop	3	3	0.319	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	3	1.000	0	0.000	0	0.000	3	1.000	3	1.000	3	1.000	3	0.235
maxclique-random	96	96	1.000	96	1.000	96	1.000	96	1.000	96	0.995	70	0.729	59	0.615	96	1.000	96	1.000	95	0.989	92	0.958	92	0.958	96	0.998
maxclique-structured	58	58	0.980	58	0.979	58	0.979	58	0.977	58	0.971	19	0.328	19	0.328	58	0.980	52	0.868	31	0.527	46	0.785	58	0.977	58	1.000
maxone-3sat	80	79	0.988	78	0.975	80	1.000	80	1.000	79	0.879	64	0.800	80	0.967	75	0.938	78	0.970	80	1.000	80	1.000	80	1.000	79	0.988
maxone-structured	60	47	0.787	3	0.066	56	0.934	55	0.918	42	0.667	55	0.902	60	0.997	1	0.033	15	0.251	60	1.000	60	1.000	60	0.999	49	0.819
min-enc-kbtree	42	42	1.000	42	1.000	42	1.000	42	1.000	42	0.499	3	0.071	41	0.713	42	0.990	42	0.905	40	0.882	32	0.745	32	0.753	42	1.000
pseudo-miplib	4	4	1.000	4	1.000	4	1.000	4	1.000	4	0.823	4	1.000	4	1.000	4	0.992	4	0.985	4	1.000	4	1.000	4	1.000	4	1.000
reversi	42	12	0.258	2	0.048	19	0.405	9	0.195	20	0.276	5	0.119	42	0.966	11	0.240	1	0.024	42	0.976	33	0.786	32	0.707	15	0.340
scheduling	4	4	0.838	0	0.000	4	0.926	4	0.431	4	0.344	0	0.000	4	0.652	0	0.000	0	0.000	4	0.799	2	0.389	4	0.650	4	0.915
aes	5	1	0.200	2	0.400	4	0.551	2	0.400	2	0.400	0	0.000	5	0.204	1	0.200	1	0.200	5	0.687	2	0.323	4	0.534	5	1.000
atcoss-mesat	16	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	14	0.870	0	0.000	0	0.000	14	0.865	8	0.500	7	0.438	0	0.000
atcoss-sugar	17	11	0.252	0	0.000	5	0.026	5	0.139	5	0.010	0	0.000	16	0.876	0	0.000	0	0.000	16	0.875	11	0.647	9	0.529	6	0.192
bcp-fir	32	28	0.778	29	0.887	32	0.955	30	0.879	30	0.096	0	0.000	32	0.985	31	0.895	10	0.036	32	0.997	31	0.969	22	0.688	32	0.981
bcp-hipp-yRa1-simp	10	10	0.995	0	0.000	10	0.993	10	0.986	10	0.613	0	0.000	10	0.986	2	0.170	6	0.378	10	0.998	10	0.998	9	0.886	10	0.769
bcp-hipp-yRa1-su	37	37	0.952	0	0.000	37	0.985	37	0.936	37	0.501	0	0.000	37	0.961	2	0.024	0	0.000	37	0.988	37	0.997	30	0.806	37	0.898
bcp-msp	33	22	0.636	0	0.000	27	0.789	28	0.820	28	0.798	4	0.118	22	0.564	0	0.000	1	0.029	19	0.546	18	0.514	24	0.669	29	0.847
bcp-mtg	30	30	0.898	0	0.000	29	0.839	27	0.823	30	0.471	8	0.267	30	1.000	0	0.000	1	0.018	30	1.000	30	1.000	28	0.933	29	0.886
bcp-syn	38	35	0.927	31	0.760	37	0.925	37	0.939	36	0.958	2	0.054	33	0.428	35	0.938	37	0.660	36	0.911	31	0.819	32	0.843	37	0.998
circuit-trace-compaction	4	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	4	1.000	0	0.000	0	0.000	4	1.000	4	1.000	4	0.865	0	0.000
close-solutions	50	29	0.423	15	0.300	40	0.744	28	0.452	23	0.119	0	0.000	49	0.927	9	0.180	0	0.000	50	0.991	50	0.998	43	0.833	36	0.713
des	49	4	0.066	0	0.000	12	0.206	0	0.000	9	0.130	0	0.000	49	0.917	1	0.020	0	0.000	49	0.955	47	0.959	37	0.755	5	0.087
haplotype-assembly	6	5	0.389	5	0.131	6	0.325	6	0.380	6	0.088	0	0.000	6	0.904	6	0.124	0	0.000	6	0.919	6	0.930	5	0.743	6	0.992
hs-timetabling	2	0	0.000	0	0.000	1	0.414	0	0.000	2	0.418	0	0.000	2	0.731	0	0.000	0	0.000	2	0.688	1	0.500	1	0.188	1	0.500
mbd	46	33	0.204	0	0.000	46	0.373	46	0.174	46	0.022	0	0.000	46	1.000	0	0.000	0	0.000	46	0.997	45	0.978	37	0.804	46	0.855
packup-pms	40	38	0.935	0	0.000	40	0.991	40	0.981	40	0.689	0	0.000	0	0.000	0	0.000	0	0.000	40	1.000	40	1.000	33	0.824	40	0.999
pbo-mqc-nencdr	25	4	0.044	0	0.000	6	0.084	0	0.000	15	0.152	0	0.000	25	0.908	0	0.000	0	0.000	25	1.000	25	1.000	21	0.801	15	0.225
pbo-mqc-nlogencdr	25	25	0.269	0	0.000	24	0.326	17	0.209	25	0.171	0	0.000	25	1.000	0	0.000	0	0.000	25	1.000	25	1.000	20	0.621	25	0.991
pbo-routing	15	7	0.467	3	0.200	15	0.995	6	0.400	15	0.928	5	0.333	15	1.000	0	0.000	0	0.000	15	1.000	15	1.000	15	1.000	14	0.933
protein-ins	12	12	1.000	12	0.986	12	1.000	12	1.000	12	0.881	3	0.250	12	0.900	12	1.000	1	0.083	12	1.000	12	1.000	11	0.917	12	0.955
tpr-Multiple-path	36	24	0.527	0	0.000	24	0.543	24	0.414	24	0.135	0	0.000	35	0.865	0	0.000	0	0.000	35	0.909	36	0.984	34	0.944	24	0.635
tpr-One-path	25	25	0.653	0	0.000	25	0.920	25	0.638	25	0.257	0	0.000	25	1.000	0	0.000	0	0.000	25	1.000	25	1.000	23	0.920	25	0.963
treewidth-computation	31	31	0.998	16	0.516	31	0.307	31	0.262	31	0.246	0	0.000	31	0.911	22	0.668	0	0.000	31	0.926	23	0.742	22	0.679	31	0.902
36 families	1253	957	0.587	421	0.285	1053	0.634	822	0.515	1073	0.436	262	0.160	1107	0.831	433	0.289	365	0.200	1186	0.903	1155	0.870	1079	0.798	1095	0.730

TABLE 19. Detailed results of AVD-SLS and state-of-the-art SLS and hybrid solvers on PMS_2017 (crafted and industrial) benchmark - 300 seconds.

solver		maxroster		Open-WBO-LSU		MaxHS-inc		SAT4J		CCEHC		LMHS-inc		WPM3-in		Dist		avdsls	
benchmark family	#inst.	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score
aes-key-recovery	2	2	0.728	2	0.800	2	0.499	2	0.590	0	0.000	2	0.730	2	0.427	2	0.515	2	0.513
aes	3	2	0.655	2	0.280	2	0.537	3	0.499	2	0.667	2	0.047	3	0.535	3	0.588	3	0.989
atcoss-mesat	5	5	1.000	5	0.694	5	0.294	3	0.282	0	0.000	5	0.091	5	0.226	0	0.000	0	0.000
atcoss-sugar	5	5	1.000	5	0.902	5	0.284	5	0.542	3	0.267	5	0.101	5	0.203	0	0.000	2	0.133
bcp-hipp-yRa1-su	3	3	1.000	3	1.000	3	0.997	3	0.972	3	0.979	3	0.984	3	0.958	3	0.978	3	0.994
bcp-msp	10	4	0.372	4	0.394	4	0.359	4	0.343	4	0.365	4	0.324	10	0.854	5	0.486	6	0.593
bcp-syn	2	2	0.891	2	0.578	2	0.851	2	0.549	2	0.831	2	0.543	2	0.561	2	0.668	2	1.000
close-solutions	1	1	0.989	1	1.000	1	0.411	1	1.000	0	0.000	1	0.115	1	0.994	0	0.000	0	0.000
des	2	2	0.908	2	0.929	2	0.854	2	0.730	0	0.000	2	0.788	2	0.979	0	0.000	0	0.000
extension-enforcement	12	12	0.685	12	0.540	12	0.868	12	0.412	12	0.444	12	0.917	12	0.346	12	0.424	12	0.976
fault-diagnosis	5	5	0.983	5	0.997	5	0.859	5	0.746	4	0.645	5	0.833	5	0.905	5	0.816	5	0.853
gen-hyper	20	20	0.890	20	0.516	20	0.330	19	0.523	17	0.587	20	0.300	20	0.344	15	0.118	20	0.900
haplotype-assembly	1	1	0.778	1	0.458	1	0.824	1	0.518	1	0.629	1	0.947	1	0.581	1	0.537	1	1.000
hs-timetabling	1	1	1.000	1	0.893	1	0.597	1	0.657	0	0.000	1	0.673	1	0.359	1	0.793	1	0.959
maxclicque-structured	8	8	0.999	8	0.994	8	0.987	8	0.984	8	1.000	8	0.989	8	0.991	8	1.000	8	1.000
maxcut-dimacs-mod	13	13	1.000	13	0.969	13	0.891	13	0.820	13	1.000	13	0.968	13	0.603	13	1.000	13	1.000
maxcut-spinglass	2	2	1.000	2	0.764	2	0.821	2	0.654	2	1.000	2	0.935	2	0.599	2	1.000	2	1.000
mbd	1	1	1.000	1	0.839	1	0.813	1	0.194	1	0.283	1	0.441	1	0.743	1	0.265	1	0.650
min-fill	12	12	0.757	12	0.551	12	0.704	12	0.402	10	0.480	12	0.507	12	0.464	12	0.578	12	0.771
reversi	6	6	1.000	6	1.000	6	0.616	6	0.836	1	0.117	6	0.520	6	0.518	2	0.288	2	0.248
scheduling	2	2	1.000	2	0.867	2	0.500	2	0.495	2	0.489	2	0.611	2	0.538	2	0.530	2	0.510
sean-safarpour	5	3	0.086	1	0.148	5	0.432	3	0.300	5	0.000	1	0.068	4	0.341	2	0.220	5	0.922
set-covering-scpc	7	7	0.971	6	0.555	7	0.840	7	0.367	7	1.000	7	0.757	7	0.142	7	0.931	7	0.984
treewidth-computation	6	6	0.994	6	1.000	6	0.706	6	0.994	6	0.977	6	0.457	6	0.701	6	0.571	6	0.960
24 families	134	125	0.862	122	0.736	127	0.661	123	0.600	103	0.490	123	0.569	133	0.580	104	0.513	115	0.706

TABLE 20. Detailed results of AVD-SLS and state-of-the-art SLS and hybrid solvers on PMS_2018 (crafted and industrial) benchmark - 300 seconds.

solver		SATLike-c		maxroster		LinSBPS		SATLike		Open-WBO-Inc-OBV		Open-WBO-Inc-MCS		Open-WBO-Gluc		Open-WBO-Riss		avdsls	
benchmark family	#inst.	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score
aes-key-recovery	1	1	0.738	1	0.689	1	1.000	1	0.413	1	0.886	1	0.838	1	0.912	1	0.738	1	0.508
aes	3	3	0.862	3	0.915	3	0.488	3	0.862	3	0.511	3	0.488	3	0.488	3	0.216	3	0.989
atcoss-mesat	5	5	0.624	5	1.000	5	0.806	0	0.000	5	0.711	5	0.706	5	0.667	5	0.528	0	0.000
bcp-msp	8	4	0.494	3	0.340	3	0.356	4	0.488	3	0.362	3	0.356	3	0.362	3	0.359	5	0.625
bcp-syn	2	2	0.998	2	0.891	2	0.747	2	1.000	2	0.576	2	0.550	2	0.550	2	0.518	2	1.000
close-solutions	1	1	1.000	1	0.989	1	1.000	0	0.000	1	1.000	1	1.000	1	1.000	1	0.745	0	0.000
des	2	2	1.000	2	0.980	2	0.980	0	0.000	2	0.884	2	0.852	2	0.902	2	0.872	0	0.000
extension-enforcement	7	7	0.941	7	0.703	7	0.746	7	0.943	7	0.682	7	0.658	7	0.591	7	0.563	7	0.957
fault-diagnosis	2	2	0.998	2	1.000	2	0.988	2	0.686	2	0.985	2	0.995	2	0.972	2	0.978	2	0.889
gen-hyper	15	15	0.881	15	0.849	15	0.712	15	0.901	15	0.495	15	0.478	15	0.497	15	0.496	15	0.927
hs-timetabling	1	1	0.109	1	0.170	1	1.000	1	0.018	1	0.094	1	0.101	1	0.100	1	0.114	1	0.128
maxclicque-structured	2	2	0.996	2	0.999	2	0.999	2	0.995	2	0.997	2	0.996	2	0.996	2	0.996	2	1.000
maxcut-dimacs-mod	4	4	1.000	4	1.000	4	0.996	4	1.000	4	0.973	4	0.974	4	0.976	4	0.967	4	1.000
maxcut-spinglass	1	1	1.000	1	1.000	1	0.851	1	1.000	1	0.741	1	0.703	1	0.725	1	0.688	1	1.000
min-fill	4	4	0.875	4	0.820	4	0.812	4	0.679	4	0.798	4	0.931	4	0.813	4	0.836	4	0.846
optic-gen	7	7	0.997	7	0.880	7	0.801	7	0.997	7	0.802	7	0.801	7	0.801	7	0.634	7	0.989
reversi	4	4	1.000	4	1.000	4	1.000	0	0.000	4	1.000	4	1.000	4	1.000	4	1.000	2	0.376
scheduling	2	2	0.472	2	0.974	2	0.968	2	0.495	2	0.909	2	0.929	2	0.886	2	0.891	2	0.509
sean-safarpour	3	3	0.992	1	0.000	3	0.259	3	1.000	3	0.406	3	0.262	3	0.259	3	0.001	3	0.942
set-covering-scpc	7	7	0.987	7	0.984	7	0.731	7	0.987	7	0.782	7	0.569	7	0.659	7	0.699	7	0.996
treewidth-computation	3	3	0.921	3	0.967	3	1.000	3	0.942	3	0.955	3	0.967	3	0.955	3	0.967	3	0.921
uaq-uaq-ppr	5	5	1.000	5	0.973	5	0.967	5	1.000	5	0.922	5	0.916	5	0.922	5	0.924	5	1.000
xai-mindset	11	11	0.806	11	0.942	11	0.937	8	0.310	11	0.862	11	0.796	11	0.698	11	0.720	7	0.443
23 families	100	96	0.856	93	0.829	95	0.832	81	0.640	95	0.754	95	0.733	95	0.727	95	0.672	83	0.698

TABLE 21. Detailed results of AVD-SLS and state-of-the-art SLS and hybrid solvers on PMS_2019 (crafted and industrial) benchmark - 300 seconds.

solver		Loandra		SATLike_c		LinSBPS2018		sls-mcs		sls-mcs-lsu		Open-WBO-g		Open-WBO-ms		avdsls	
benchmark family	#inst.	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score
aes-key-recovery	5	5	0.857	5	0.880	5	0.887	5	0.401	5	0.396	5	0.860	5	0.902	5	0.426
aes	3	3	0.699	3	0.973	3	0.467	3	0.859	3	0.849	3	0.298	3	0.287	3	0.954
atcoss-mesat	5	5	0.853	5	0.816	5	0.965	1	0.037	1	0.037	5	0.728	5	0.779	0	0.000
atcoss-sugar	5	5	0.934	5	0.953	5	0.953	5	0.586	5	0.537	5	0.808	5	0.926	2	0.136
bcp-fir	1	1	1.000	1	0.961	1	0.980	1	0.961	1	0.961	1	0.583	1	0.219	1	0.980
bcp-hipp-yRa1-simp	1	1	1.000	1	1.000	1	1.000	1	1.000	1	1.000	1	1.000	1	1.000	1	0.967
bcp-hipp-yRa1-su	2	2	0.992	2	0.990	2	0.983	2	0.995	2	0.995	2	0.961	2	0.966	2	0.990
bcp-msp	2	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000
bcp-syn	1	1	0.920	1	0.939	1	0.958	1	0.979	1	1.000	1	0.639	1	0.667	1	0.920
causal-discovery	3	3	1.000	3	1.000	3	1.000	3	0.425	3	0.425	3	1.000	3	0.949	3	0.425
close-solutions	14	14	0.929	14	0.909	14	0.791	14	0.996	14	0.996	14	0.798	14	0.581	10	0.618
des	11	11	0.988	11	0.000	11	0.967	0	0.000	0	0.000	11	0.870	11	0.814	1	0.083
extension-enforcement	10	10	0.795	10	0.969	10	0.857	10	0.961	10	0.901	10	0.663	10	0.600	10	0.982
fault-diagnosis	7	7	0.980	7	0.981	7	0.981	7	0.722	7	0.722	7	0.968	7	0.962	7	0.870
gen-hyper	15	15	0.836	15	0.890	15	0.749	13	0.729	13	0.689	15	0.708	14	0.695	15	0.948
hs-timetabling	1	1	0.679	1	0.950	1	1.000	1	0.213	1	0.151	1	0.186	1	0.047	1	0.128
logic-synthesis	1	1	0.848	1	0.812	1	0.805	1	1.000	1	0.990	1	0.511	1	0.505	1	0.990
maxclicque-structured	14	14	0.993	14	0.999	14	0.996	14	0.995	14	0.995	14	0.987	14	0.991	14	1.000
maxcut-bipartite	3	3	0.894	3	1.000	3	0.921	3	0.994	3	0.984	3	0.820	3	0.836	3	1.000
maxcut-dimacs-mod	8	8	0.997	8	1.000	8	0.993	8	0.998	8	0.998	8	0.980	8	0.983	8	0.969
MaximumCommonSub-GraphExtraction	15	14	0.933	15	0.981	15	0.999	15	0.983	15	0.986	14	0.925	12	0.790	15	0.987
MaxSATQueries-in-Interpretable-Classifiers-MLIC	7	7	0.914	7	0.943	7	0.820	7	0.985	7	0.975	7	0.544	7	0.520	7	0.882
MaxSATQueries-in-Interpretable-Classifiers-wcnf	8	8	0.955	8	0.944	8	0.931	8	0.899	8	0.898	8	0.628	8	0.743	8	0.860
mbd	6	6	1.000	6	0.636	6	1.000	6	0.693	6	0.682	6	0.788	6	0.731	6	0.728
min-fill	11	11	0.811	11	0.672	11	0.606	10	0.764	10	0.685	11	0.497	11	0.587	11	0.816
optic-gen	12	12	0.941	12	0.937	12	0.835	12	0.998	12	0.998	12	0.697	12	0.671	12	0.991
pseudoBoolean-garden	1	1	0.850	1	0.941	1	0.639	1	0.997	1	0.997	1	0.639	1	0.650	1	1.000
pseudoBoolean-primes-dimacs	5	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000
railway-transport	3	3	0.943	3	0.547	3	0.652	2	0.648	2	0.620	3	0.301	3	0.254	3	0.685
ramsey	14	14	0.933	14	1.000	14	0.945	14	1.000	14	1.000	14	0.788	14	0.726	14	1.000
reversi	9	9	1.000	9	1.000	9	1.000	9	0.699	9	0.747	9	1.000	9	1.000	2	0.167
scheduling	4	4	0.992	4	0.637	4	0.962	4	0.601	4	0.627	4	0.925	4	0.881	4	0.660
sean-safarpour	11	11	0.399	10	0.493	11	0.254	11	0.718	11	0.718	11	0.235	11	0.254	11	0.869
set-covering-scpc	7	7	0.888	7	0.940	7	0.733	7	0.989	7	0.985	7	0.740	7	0.637	7	1.000
treewidth-computation	6	6	1.000	6	0.934	6	1.000	6	0.967	6	0.956	6	0.983	6	0.977	6	0.944
uaq-uaq-plb	1	1	0.976	1	1.000	1	1.000	1	1.000	1	1.000	1	0.952	1	0.930	1	1.000
uaq-uaq-ppr	12	12	0.971	12	1.000	12	0.959	12	1.000	12	1.000	12	0.916	12	0.881	12	1.000
xai-mindset	9	9	0.932	9	0.926	9	0.903	9	0.861	9	0.747	9	0.791	9	0.903	7	0.557
38 families	253	245	0.859	245	0.830	246	0.829	227	0.754	227	0.743	245	0.703	242	0.680	215	0.724

TABLE 22. Detailed results of AVD-SLS and state-of-the-art SLS and hybrid solvers on WPMS_2014 (crafted and industrial) benchmark - 300 seconds.

solver		CCLS2014		CCMPA		Dist		SAT4J-ms-inc		WPM-2014-in		optimax2-g-i		optimax2w-r-i		avdsls	
benchmark family	#Ins.	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score
CSG	10	4	0.156	10	0.541	10	0.341	10	1.000	10	0.848	10	0.999	10	0.998	10	0.909
auctions-auc-paths	20	20	1.000	20	1.000	20	1.000	0	0.000	20	0.996	19	0.948	20	0.998	20	1.000
auctions-auc-scheduling	20	20	1.000	20	1.000	20	1.000	5	0.250	20	1.000	20	1.000	20	1.000	20	1.000
frb	34	34	1.000	34	1.000	34	1.000	4	0.118	34	0.992	34	0.999	34	0.999	34	0.001
min-enc-planning	18	0	0.000	0	0.000	17	0.940	18	1.000	18	0.970	18	1.000	18	1.000	18	0.995
min-enc-planning-old	26	7	0.268	7	0.269	26	1.000	26	1.000	26	1.000	26	1.000	26	1.000	26	1.000
min-enc-planning	12	0	0.000	0	0.000	12	1.000	12	1.000	12	1.000	12	1.000	12	1.000	12	1.000
min-enc-warehouses	18	18	0.914	18	0.768	18	0.768	1	0.056	18	0.593	18	0.676	18	0.665	18	1.000
pseudo-miplib-normalized-mps	10	3	0.269	4	0.361	5	0.409	3	0.300	8	0.755	8	0.797	8	0.793	6	0.467
ramsey	15	15	0.970	15	0.954	15	0.986	3	0.200	15	0.188	14	0.407	14	0.428	15	0.798
random-net	32	32	0.880	32	0.959	32	0.835	0	0.000	32	0.959	32	0.651	32	0.658	32	0.967
set-covering-scp4x	10	10	0.940	10	0.772	10	0.604	0	0.000	10	0.961	10	0.920	10	0.920	10	1.000
set-covering-scp5x	10	10	0.877	10	0.612	10	0.487	0	0.000	10	0.948	10	0.901	10	0.901	10	1.000
set-covering-scp6x	5	5	0.960	5	0.738	5	0.541	0	0.000	5	0.924	5	0.887	5	0.887	5	1.000
set-covering-scpn	20	1	0.032	20	0.401	0	0.000	0	0.000	17	0.768	20	0.905	20	0.904	20	0.763
wmaxcut-dimacs_mod	38	38	1.000	38	1.000	38	0.998	0	0.000	38	0.909	38	0.849	38	0.858	38	0.819
wmaxcut-spinglass	4	4	1.000	4	1.000	4	1.000	0	0.000	4	0.902	4	0.756	4	0.757	4	0.314
haplotyping-pedigrees	100	24	0.176	18	0.175	51	0.413	4	0.040	100	0.999	98	0.961	91	0.668	100	0.985
hs-timetabling	10	0	0.000	0	0.000	2	0.042	0	0.000	10	0.837	9	0.573	9	0.556	5	0.196
packup-wpms	99	2	0.019	10	0.100	99	0.973	0	0.000	99	0.991	99	0.937	99	0.954	99	0.986
preference_planning	29	7	0.230	7	0.122	25	0.713	27	0.931	29	1.000	29	0.967	29	0.967	26	0.793
timetabling	23	0	0.000	0	0.000	18	0.215	0	0.000	20	0.673	20	0.686	20	0.431	20	0.242
upgradeability-problem	100	0	0.000	0	0.000	0	0.000	29	0.290	100	1.000	100	1.000	100	1.000	100	0.641
wcsp-spot5-dir	21	18	0.844	21	0.997	21	0.977	3	0.143	21	0.996	21	0.976	20	0.927	21	0.986
wcsp-spot5-log	20	20	0.994	20	0.999	20	0.989	3	0.150	20	0.989	19	0.926	18	0.872	20	0.980
25 families	704	292	0.541	323	0.551	512	0.689	148	0.259	696	0.888	693	0.869	685	0.846	689	0.794

We found that some miplib instances can be solved when the time limit is increased.

For the **industrial** class, AVD-SLS shows excellent performance in 16 benchmark families and is the best SLS solver on this class. Moreover, AVD-SLS perform competitively with the PMSAT solvers with regards to the number of best solutions found. AVD-SLS has two hard benchmark families: robot navigation and shift design. We found that the robot navigation instances of large-size with hundred thousands of variables and clauses, whereas the shift design instances have very large size with hundred millions of variables and clauses. As discussed in the PMS results, such hard instances may need more pre-processing techniques or structure extraction methods for them to be solved by SLS solvers.

This evaluation study shows that AVD-SLS is the best solver among SLS solvers (Fig. 5 and Fig. 6) and that it is competitive to a number of well-known hybrid solvers, as shown in Fig.s 13 to 18. For MSE 2015 to MSE 2018, AVD-SLS is among the top three solvers. It is remarkable that for most of solved instances, AVD-SLS is able to find the best solution or near best solutions. As discussed in the PMS section above, AVD-SLS may be improved to solve more hard

TABLE 23. Detailed results of AVD-SLS and state-of-the-art SLS and hybrid solvers on WPMS_2015 (crafted and industrial) benchmark - 300 seconds.

solver		CCEHC		CCLS2015		Dist1		Dist2		ILP-2015-in		WPM3-2015-in		optiriss-sel-i		avdsls	
benchmark family	#Ins.	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score
CSG	10	10	0.785	5	0.187	10	0.345	10	0.310	2	0.200	8	0.800	10	0.531	10	0.940
auctions-auc-paths	20	20	1.000	20	1.000	20	1.000	20	1.000	20	1.000	9	0.450	19	0.875	20	1.000
auctions-auc-scheduling	20	20	1.000	20	1.000	20	1.000	20	1.000	20	1.000	19	0.950	19	0.948	20	1.000
causal-discovery	35	35	0.743	0	0.000	14	0.319	26	0.453	0	0.000	20	0.475	0	0.000	35	0.491
frb	34	34	1.000	34	1.000	34	1.000	34	1.000	19	0.559	21	0.609	31	0.851	34	0.001
min-enc-planning	30	29	0.960	0	0.000	29	0.962	29	0.964	27	0.900	29	0.967	30	0.996	30	0.997
min-enc-warehouses	18	18	0.749	6	0.299	18	0.770	18	0.802	18	1.000	14	0.588	18	0.430	18	0.993
pseudo-miplib-mps	10	4	0.332	3	0.282	5	0.404	5	0.403	3	0.300	8	0.798	8	0.745	6	0.465
ramsey	15	15	0.962	15	0.981	15	0.979	15	0.972	3	0.200	15	0.240	14	0.576	15	0.787
random-net	32	32	0.931	32	0.906	32	0.860	32	0.865	23	0.719	6	0.188	29	0.771	32	0.998
set-covering-scp4x	10	10	0.976	10	0.943	10	0.607	10	0.896	10	1.000	10	0.975	10	0.939	10	1.000
set-covering-scp5x	10	10	0.944	10	0.892	10	0.511	10	0.694	10	1.000	10	0.980	10	0.919	10	1.000
set-covering-scp6x	5	5	0.983	5	0.951	5	0.526	5	0.703	5	1.000	5	0.932	5	0.901	5	1.000
set-covering-scpn	20	8	0.384	4	0.144	20	0.684	20	0.679	9	0.450	20	0.885	20	0.883	20	0.760
wmaxcut-dimacs-mod	38	38	1.000	38	1.000	38	0.998	38	1.000	16	0.421	38	0.964	38	0.842	38	0.819
wmaxcut-spinglass	4	4	1.000	4	1.000	4	0.998	4	1.000	4	1.000	0	0.000	4	0.770	4	0.314
BTBNSL	51	51	0.982	3	0.059	51	0.949	51	0.954	9	0.176	27	0.529	11	0.182	51	0.880
correlation-clustering	124	100	0.274	3	0.004	106	0.211	115	0.195	4	0.032	74	0.388	124	0.800	124	0.575
haplotyping-pedigrees	100	30	0.244	21	0.113	78	0.576	66	0.546	10	0.100	100	1.000	100	0.961	100	0.985
hs-timetabling	10	1	0.003	0	0.000	2	0.051	2	0.050	0	0.000	9	0.802	9	0.788	5	0.213
packup-wpms	99	99	0.974	3	0.027	99	0.973	99	0.957	99	0.992	99	0.992	99	0.978	99	0.986
preference-planning	29	25	0.772	6	0.195	25	0.461	25	0.681	8	0.276	29	1.000	29	0.770	26	0.793
railway-transport	9	4	0.189	0	0.000	5	0.226	5	0.206	0	0.000	9	0.710	9	0.586	7	0.596
timetabling	23	9	0.087	0	0.000	18	0.313	20	0.425	0	0.000	19	0.689	20	0.519	20	0.379
upgradeability-problem	100	51	0.083	0	0.000	100	0.130	100	0.130	100	1.000	100	1.000	0	0.000	100	0.641
wcsp-spot5-dir	21	21	0.966	17	0.796	21	0.977	21	0.982	17	0.810	18	0.851	20	0.933	21	0.986
wcsp-spot5-log	20	20	0.999	20	0.993	20	0.988	20	0.989	6	0.300	17	0.840	17	0.819	20	0.979
27 families	897	703	0.716	279	0.473	809	0.660	820	0.698	442	0.535	733	0.726	703	0.715	880	0.762

TABLE 24. Detailed results of AVD-SLS and state-of-the-art SLS and hybrid solvers on WPMS_2016 (crafted and industrial) benchmark - 300 seconds.

solver		CCEHC		CCLS		Dist-r		Dist		HS-Greedy		Naps-1.02-ms		Optiriss6-in		Ramp		SC2016		SsMonteCarlo		WPM3-2015-in		dsat-wpm3-in-wpms		dsat-wpm3-s-in-wpms		avdsls	
benchmark family	#Inst.	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score
CSG	10	10	0.803	6	0.207	10	0.340	10	0.540	10	0.228	3	0.300	10	0.609	10	0.578	9	0.381	3	0.058	10	0.996	9	0.900	10	0.975	10	0.914
auctions-auc-paths	20	20	1.000	20	1.000	20	1.000	20	1.000	20	0.999	7	0.350	20	0.985	20	1.000	20	1.000	20	0.863	15	0.750	20	1.000	20	0.999	20	1.000
auctions-auc-scheduling	20	20	1.000	20	1.000	20	1.000	20	1.000	20	1.000	20	1.000	20	1.000	20	1.000	20	1.000	17	0.835	19	0.950	19	0.950	20	1.000	20	1.000
causal-discovery	35	35	0.505	0	0.000	14	0.158	17	0.231	35	0.345	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	31	0.595	23	0.518	24	0.424	35	0.339
frb	34	34	1.000	34	1.000	0	0.000	0	0.000	34	0.994	27	0.794	34	0.998	34	1.000	34	1.000	34	0.888	14	0.408	34	0.997	34	0.995	34	0.001
min-enc-planning	30	29	0.956	0	0.000	29	0.957	29	0.964	29	0.750	15	0.500	30	1.000	0	0.000	0	0.000	0	0.000	29	0.967	29	0.967	25	0.833	30	0.997
min-enc-warehouses	18	18	0.914	6	0.302	18	0.924	18	0.811	18	0.411	1	0.056	17	0.420	18	0.807	6	0.314	0	0.000	15	0.669	7	0.321	10	0.431	18	1.000
pseudo-miplib-mps	10	4	0.330	3	0.281	5	0.418	5	0.407	4	0.273	2	0.200	8	0.750	4	0.359	3	0.290	0	0.000	8	0.792	7	0.691	7	0.630	6	0.465
ramsey	15	15	0.959	15	0.955	0	0.000	0	0.000	15	0.441	3	0.200	15	0.607	15	0.927	15	1.000	15	0.360	15	0.251	15	0.872	15	0.865	15	0.774
random-net	32	32	0.935	32	0.907	32	0.849	32	0.864	32	0.948	0	0.000	31	0.933	32	0.991	32	0.921	32	0.050	16	0.428	28	0.863	32	0.964	32	0.998
set-covering-scp4x	10	10	0.982	10	0.937	10	0.825	10	0.887	10	0.648	0	0.000	10	0.863	10	0.758	10	0.973	10	0.011	9	0.875	4	0.397	10	0.964	10	1.000
set-covering-scp5x	10	10	0.967	10	0.892	10	0.642	10	0.687	10	0.718	0	0.000	10	0.936	10	0.602	2	0.184	10	0.003	9	0.877	8	0.789	10	0.976	10	1.000
set-covering-scp6x	5	5	0.996	5	0.955	5	0.751	5	0.704	5	0.723	0	0.000	5	0.728	5	0.736	5	0.978	5	0.003	3	0.562	2	0.380	5	0.938	5	1.000
set-covering-scpn	20	10	0.500	13	0.545	20	0.527	20	0.189	20	0.988	0	0.000	20	0.602	3	0.061	0	0.000	20	0.231	18	0.781	10	0.489	15	0.707	20	0.759
staff-scheduling	8	8	0.599	0	0.000	6	0.349	8	0.801	8	0.265	1	0.125	8	0.417	5	0.354	1	0.075	0	0.000	8	0.460	2	0.164	3	0.236	8	0.962
wmaxcut-dimacs-mod	38	38	0.997	38	1.000	0	0.000	0	0.000	38	0.975	0	0.000	38	0.843	38	1.000	38	1.000	38	0.978	37	0.943	38	1.000	38	1.000	38	0.819
wmaxcut-spinglass	4	4	1.000	4	1.000	0	0.000	0	0.000	4	0.833	1	0.250	4	0.895	4	1.000	4	1.000	4	0.379	2	0.477	4	0.999	3	0.701	4	0.314
BTBNSL	51	51	0.974	3	0.059	50	0.970	51	0.952	51	0.803	6	0.118	11	0.199	3	0.059	3	0.059	0	0.000	43	0.836	34	0.659	44	0.846	51	0.872
abstraction-refinement	11	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	5	0.206	0	0.000	0	0.000	0	0.000	5	0.394	5	0.269	2	0.057	11	1.000
correlation-clustering	124	101	0.259	4	0.005	103	0.146	117	0.548	114	0.162	0	0.000	123	0.919	4	0.006	9	0.010	0	0.000	103	0.412	85	0.352	74	0.268	124	0.575
haplotyping-pedigrees	100	38	0.281	22	0.158	83	0.667	100	0.705	100	0.015	0	0.000	100	1.000	18	0.159	32	0.269	0	0.000	98	0.980	98	0.980	76	0.760	100	0.985
hs-timetabling	10	1	0.001	0	0.000	5	0.163	2	0.066	7	0.291	0	0.000	10	0.890	0	0.000	0	0.000	0	0.000	9	0.714	4	0.214	3	0.128	5	0.218
packup-wpms	99	99	0.974	1	0.007	99	0.969	99	0.976	99	0.959	0	0.000	0	0.000	9	0.090	12	0.114	0	0.000	99	0.992	99	0.992	82	0.820	99	0.986
preference-planning	29	25	0.783	6	0.195	19	0.364	25	0.765	24	0.243	0	0.000	0	0.000	7	0.122	7	0.230	0	0.000	29	0.997	29	0.999	26	0.895	26	0.793
railway-transport	9	3	0.250	0	0.000	5	0.174	5	0.214	5	0.046	0	0.000	8	0.603	0	0.000	0	0.000	0	0.000	8	0.633	7	0.559	7	0.484	7	0.607
relational-inference	9	2	0.202	0	0.000	6	0.664	6	0.664	5	0.555	0	0.000	5	0.338	2	0.222	0	0.000	0	0.000	7	0.776	7	0.757	4	0.437	9	0.904
timetabling	23	7	0.076	0	0.000	8	0.076	20	0.306	20	0.104	0	0.000	20	0.557	0	0.000	0	0.000	0	0.000	19	0.720	16	0.670	17	0.675	20	0.390
upgradeability-problem	100	66	0.120	0	0.000	100	0.136	100	0.124	100	0.156	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	100	1.000	100	1.000	90	0.900	100	0.641
wcsp-spot5-dir	21	21	0.966	17	0.797	21	0.980	21	0.983	21	0.853	15	0.714	20	0.935	21	0.995	19	0.890	5	0.212	21	0.988	19	0.897	19	0.896	21	0.986
wcsp-spot5-log	20	20	0.999	20	0.993	20	0.989	20	0.989	20	0.972	14	0.700	18	0.811	20	0.999	20	0.996	3	0.150	18	0.891	14	0.700	19	0.936	20	0.980
30 families	925	736	0.678	289	0.440	718	0.501	770	0.546	878	0.557	115	0.177	600	0.635	312	0.461	301	0.423	216	0.167	817	0.737	776	0.711	744	0.725	908	0.776

TABLE 25. Detailed results of AVD-SLS and state-of-the-art SLS and hybrid solvers on WPMS_2017 (crafted and industrial) benchmark - 300 seconds.

solver		maxroster		WPM3-in		SAT4J		MaxHS-inc		LMHS-inc		Dist		CCEHC		Open-WBO-LSU		avdsls	
benchmark family	#Inst.	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score
af-synthesis	1	1	1.000	1	0.993	1	0.993	1	0.971	1	0.950	1	0.545	1	0.957	1	1.000	1	0.918
BTBNSL	11	11	0.985	11	0.983	11	0.982	11	0.996	11	0.955	11	0.928	11	0.954	6	0.449	11	0.869
causal-discovery	3	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	3	1.000	3	1.000	0	0.000	3	1.000
correlation-clustering	12	12	0.860	12	0.576	12	0.780	12	0.878	12	0.859	12	0.661	11	0.534	2	0.023	12	0.780
css-refactoring	1	1	0.999	1	1.000	1	0.999	1	0.999	1	1.000	1	0.999	1	0.999	0	0.000	1	0.999
haplotyping-pedigrees	1	1	1.000	1	0.995	1	0.446	1	0.643	1	0.061	1	0.504	1	0.347	1	1.000	1	0.959
hs-timetabling	6	6	0.819	6	0.748	6	0.528	6	0.393	6	0.531	1	0.050	0	0.000	6	0.797	4	0.307
lisbon-wedding	9	9	0.996	9	0.968	9	0.970	9	0.676	9	0.835	8	0.613	5	0.340	9	1.000	8	0.622
wmaxcut-dimacs-mod	9	9	1.000	9	0.975	9	0.901	9	0.897	8	0.861	9	1.000	9	1.000	9	0.918	9	1.000
min-width	18	18	0.897	18	0.878	18	0.792	18	0.920	18	0.891	0	0.000	6	0.318	0	0.000	18	0.991
pseudo-miplib-mps	5	3	0.433	5	0.886	3	0.474	3	0.360	3	0.438	0	0.000	0	0.000	1	0.195	1	0.016
railway-transport	4	3	0.388	3	0.196	4	0.466	3	0.737	2	0.279	2	0.188	2	0.190	3	0.400	3	0.480
rna-alignment	5	5	1.000	5	0.999	5	0.995	5	0.998	5	0.991	5	0.998	5	0.993	5	1.000	5	0.996
shiftdesign-limits	2	2	0.998	1	0.500	2	0.999	1	0.250	0	0.000	0	0.000	0	0.000	2	1.000	2	0.833
wcsp-spot5-log	2	2	1.000	2	0.937	2	0.919	2	0.918	2	0.979	2	0.966	2	0.992	2	0.944	2	0.880
staff-scheduling	6	6	0.390	6	0.346	6	0.650	6	0.445	6	0.357	6	0.709	6	0.479	6	0.781	6	0.881
timetabling	5	3	0.536	5	0.717	2	0.070	3	0.498	3	0.445	3	0.313	2	0.160	3	0.510	3	0.370
17 families	100	92	0.782	95	0.747	92	0.704	91	0.681	88	0.614	65	0.557	65	0.545	56	0.589	90	0.759

instances by means of new pre-processing techniques [74], [75] to reduce the dimensionality of large-size instances or by extracting complex structure [76] or by using both. Almost all complete and hybrid PMSAT solvers incorporate many pre- or co-processing techniques to reduce the dimensionality of large-sized instances.

TABLE 26. Detailed results of AVD-SLS and state-of-the-art SLS and hybrid solvers on WPMS_2018 (crafted and industrial) benchmark - 300 seconds.

solver		LinSBPS		Open-WBO-Inc-BMO		maxroster		Open-WBO-Inc-Cluster		SATLike-c		SATLike		Open-WBO-Gluc		Open-WBO-Riss		avdsls	
benchmark family	#Inst.	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score
abstraction-refinement	2	2	0.999	2	1.000	2	0.996	2	0.924	2	0.900	2	1.000	2	0.900	2	0.120	2	1.000
af-synthesis	11	11	1.000	11	0.992	11	0.998	11	0.898	11	0.701	11	0.708	11	0.991	11	0.982	11	0.722
BTBNSL	9	9	0.992	9	0.996	9	0.978	9	0.881	9	0.848	9	0.847	9	0.837	9	0.842	9	0.896
causal-discovery	11	11	0.000	11	0.000	11	0.000	11	0.000	11	0.000	10	0.000	9	0.000	10	0.000	11	1.000
cluster-expansion	12	12	0.944	12	0.978	12	0.996	12	0.944	12	0.999	12	0.999	12	0.944	12	0.943	12	1.000
correlation-clustering	8	8	0.959	8	0.869	8	0.895	8	0.729	8	0.969	8	0.973	7	0.257	8	0.293	8	0.721
hs-timetabling	9	9	1.000	9	0.321	9	0.241	9	0.239	9	0.161	3	0.050	9	0.268	9	0.321	5	0.078
lisbon-wedding	6	3	0.500	3	0.500	3	0.500	3	0.491	3	0.370	3	0.350	3	0.500	3	0.500	3	0.319
wmaxcut-dimacs-mod	6	6	0.975	6	0.908	6	1.000	6	0.909	6	0.999	6	1.000	6	0.898	6	0.777	6	1.000
min-width	9	9	0.976	9	0.982	9	0.894	9	0.856	9	0.974	9	0.973	7	0.550	8	0.633	9	0.979
pseudo-miplib-mps	3	1	0.164	1	0.333	1	0.312	1	0.222	1	0.164	0	0.000	1	0.164	1	0.150	0	0.000
power-distribution	2	2	1.000	2	0.948	2	0.696	2	1.000	2	1.000	2	1.000	2	1.000	2	1.000	2	1.000
railway-transport	3	3	0.910	3	0.837	3	0.696	3	0.704	3	0.771	2	0.491	3	0.742	3	0.848	2	0.500
relational-inference	2	2	0.009	2	0.155	2	0.000	2	0.153	2	0.024	2	0.025	2	0.041	2	1.000	2	0.034
robot-nagivation	2	2	1.000	2	0.938	1	0.188	2	1.000	2	1.000	0	0.000	2	1.000	2	0.778	0	0.000
wcsp-spot5-log	2	2	1.000	2	0.957	2	1.000	2	0.926	2	0.944	2	0.886	2	0.944	2	0.846	2	0.880
staff-scheduling	6	6	0.873	6	0.783	6	0.472	6	0.731	5	0.743	6	0.914	6	0.724	6	0.700	6	0.784
tcp-tcp-students	5	5	1.000	5	1.000	5	0.974	5	0.972	5	0.986	5	0.990	5	1.000	5	0.997	5	0.990
18 families	108	103	0.795	103	0.750	102	0.657	103	0.699	102	0.697	92	0.623	98	0.653	101	0.652	95	0.661

TABLE 27. Detailed results of AVD-SLS and state-of-the-art SLS and hybrid solvers on WPMS_2019 (crafted and industrial) benchmark - 300 seconds.

solver		TT-Open-WBO-Inc		Loandra		Open-WBO-Inc1		LinSBPS2018		SATLike-c		Open-WBO-Inc2		Open-WBO-g		sls-mcs2		uwrmaxsat-inc		Open-WBO-ms		sls-mcs		avdsls	
benchmark family	#Inst.	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score	#sol.	avg. score
abstraction-refinement	10	9	0.900	10	0.918	10	0.953	8	0.797	10	0.892	10	0.953	9	0.865	7	0.629	4	0.399	10	0.981	7	0.629	10	0.939
af-synthesis	12	12	1.000	12	1.000	12	0.926	12	1.000	12	0.787	12	0.993	12	0.992	12	0.913	12	0.957	12	0.992	12	0.673	12	0.752
BTBNSL	12	12	0.988	12	0.996	12	0.986	12	0.981	12	0.877	12	0.986	12	0.984	12	0.948	12	0.950	12	0.983	12	0.948	12	0.852
causal-discovery	15	15	0.000	15	0.000	15	0.000	15	0.000	15	0.000	15	0.000	15	0.000	9	0.000	15	0.000	15	0.000	9	0.000	15	1.000
correlation-clustering	22	22	0.868	22	0.947	22	0.874	22	0.754	22	0.734	22	0.788	22	0.822	20	0.754	22	0.751	19	0.673	20	0.701	22	0.429
drmx-cryptogen	1	1	0.884	1	0.956	1	0.887	1	0.941	1	0.928	1	0.887	1	0.896	1	0.980	1	1.000	1	0.898	1	0.978	1	0.980
hs-timetabling	7	7	0.688	7	0.842	7	0.392	7	0.873	7	0.604	7	0.385	7	0.290	7	0.369	7	0.443	7	0.352	7	0.166	5	0.099
lisbon-wedding	8	6	0.750	6	0.749	6	0.750	6	0.750	6	0.587	6	0.750	6	0.589	6	0.749	4	0.490	6	0.612	6	0.603	6	0.502
wmaxcut-dimacs-mod	11	11	0.908	11	0.981	11	0.985	11	0.983	11	1.000	11	0.904	11	0.912	11	0.999	11	0.932	11	0.912	11	1.000	11	0.999
MaxSATQueries-in-Interpretable-Classifiers-MLIC	8	8	0.901	8	0.729	8	0.691	8	0.735	8	0.937	8	0.691	8	0.666	8	0.954	7	0.611	8	0.722	8	0.948	8	0.927
MaxSATQueries-in-Interpretable-Classifiers-wcnf	7	7	0.888	7	0.674	7	0.646	7	0.754	7	0.826	7	0.646	7	0.613	7	0.909	4	0.381	7	0.668	7	0.873	7	0.889
metro	2	2	1.000	2	1.000	2	0.983	2	1.000	2	0.796	2	1.000	2	1.000	2	0.918	2	1.000	2	0.761	2	0.752	2	0.825
Minimum-Weight-Dominating-Set-Problem	7	7	0.743	7	0.746	7	0.957	6	0.648	6	0.715	7	0.957	7	0.744	5	0.336	5	0.563	6	0.651	5	0.336	7	0.593
min-width	15	15	0.966	15	0.987	15	0.972	15	0.958	15	0.968	15	0.966	15	0.888	15	0.974	15	0.890	15	0.880	15	0.971	15	0.979
random-net	15	10	0.610	15	0.950	15	0.968	15	0.977	15	0.996	15	0.912	15	0.912	15	0.999	15	0.959	15	0.917	15	0.999	15	1.000
Parametric-RBAC-Maintenance	15	15	0.865	15	0.846	15	0.846	15	0.544	15	0.981	15	0.846	15	0.850	15	0.981	15	0.850	15	0.852	15	0.904	15	0.992
pseudo-miplib-mps	5	3	0.507	3	0.592	3	0.353	3	0.360	3	0.360	3	0.353	3	0.456	3	0.305	3	0.463	3	0.376	3	0.326	1	0.016
power-distribution	10	10	0.958	10	0.878	10	0.942	10	1.000	10	1.000	10	0.942	10	0.963	10	1.000	10	0.863	10	0.958	10	1.000	10	1.000
railway-transport	3	3	0.929	3	0.954	3	0.810	3	0.844	3	0.734	3	0.798	3	0.617	3	0.785	3	0.752	2	0.372	3	0.664	2	0.460
ramsey	12	12	0.713	12	0.611	12	0.710	12	0.650	12	0.982	12	0.643	12	0.671	12	0.949	12	0.684	12	0.683	12	0.975	12	0.978
relational-inference	2	2	0.627	2	0.274	2	0.848	2	0.053	2	0.562	2	0.848	2	0.275	2	0.092	2	0.061	2	0.279	2	0.092	2	0.174
robot-nagivation	2	2	1.000	2	1.000	2	0.938	2	1.000	2	1.000	2	0.938	2	0.969	2	0.556	0	0.000	0	0.000	2	0.375	0	0.000
set-covering-scpn	12	12	0.961	12	0.965	12	0.964	12	0.903	12	0.939	12	0.951	12	0.890	12	0.999	12	0.890	12	0.895	12	0.997	12	0.753
shiftdesign-limits	11	11	0.978	11	0.325	11	0.931	9	0.818	11	0.999	11	0.936	11	0.909	0	0.000	11	0.964	4	0.324	0	0.000	0	0.000
wcsp-spot5-dir	1	1	0.958	1	1.000	1	0.962	1	1.000	1	0.996	1	0.962	1	0.979	1	1.000	1	0.913	1	0.966	1	1.000	1	1.000
wcsp-spot5-log	3	3	0.958	3	0.989	3	0.955	3	1.000	3	0.983	3	0.955	3	0.979	3	0.977	3	0.994	3	0.978	3	0.975	3	0.880
staff-scheduling	7	7	0.961	7	0.928	7	0.727	7	0.870	7	0.805	7	0.727	7	0.731	7	0.849	2	0.222	7	0.768	7	0.801	7	0.719
tcp-tcp-students	11	11	1.000	11	0.988	11	0.978	11	1.000	11	0.981	11	0.999	11	0.995	11	0.992	11	0.985	11	0.992	11	0.981	11	0.985
timetabling	15	11	0.633	11	0.617	11	0.319	11	0.492	11	0.175	11	0.321	11	0.353	11	0.261	11	0.507	11	0.251	11	0.147	11	0.149
29 families	261	247	0.832	253	0.808	253	0.802	248	0.782	252	0.798	253	0.794	252	0.752	229	0.730	232	0.672	239	0.679	229	0.683	235	0.685

VII. CONCLUSION

In this work, we have presented an adaptive variable depth SLS algorithm for the PMSAT problem. In this novel algorithm framework, we have proposed two main components: an adaptive parameter tuner and a VDS algorithm adopted for the PMSAT problem. This work provides a comprehensive evaluation of the AVD-SLS solver implemented based on our proposed algorithm with the use of MSE 2014–2019 benchmarks. AVD-SLS was evaluated on more than 3,600 instances from MSE weighted and unweighted benchmarks.

As expected, the experimental evaluation study in Section V demonstrates that our solver AVD-SLS is a highly competitive SLS solver. AVD-SLS proves that PMSAT SLS solvers have the capability to compete with hybrid PMSAT solvers in both PMS and WPMS. Generally, AVD-SLS improves the quality of solutions when the time limit is increased. This emphasizes the critical role of each SLS method's component.

Furthermore, AVD-SLS outperforms all PMSAT SLS solvers that participated in MSE 2014–2019 on crafted and industrial benchmarks with regards to three evaluation criteria: number of solved instances, number of best solutions, and score measure. AVD-SLS performs remarkably better in WPMS than in PMS. For example, if we compare the rankings of AVD-SLS and SATLike on PMS (Fig. 11) and on WPMS (Fig. 17) in MSE 2018, AVD-SLS ranked fourth and third, respectively, whereas SATLike ranked fifth and seventh, respectively.

Our investigation shows that state-of-the-art PMSAT SLS solvers are built around two important algorithmic components: a variable-pick heuristic and a weighting scheme. However, some hard benchmark families constitute very large and/or complex structures, which are still beyond the capacity of existing SLS solvers. Based on the evaluation results in Section V, we found that the general framework of an SLS method consists of additional algorithmic components that can be exploited to improve state-of-the-art SLS solvers. In this study, we selected two components, namely, parameter tuning and VDS for large neighborhood search. Other algorithmic components include new pre-processing techniques, neighborhood definition and search method, and other diversification techniques such as reset and restarts.

In our proposed algorithm, we designed the adaptive parameter tuner based on our study of the features of an input instance. In this study, we considered the problem size features, variable-clause features, and balance features [77]. More features may be included in the future to improve the adaptive tuner, such as variable-graph features, clause-graph features, and local search probes including the minimum fraction of unsatisfied clauses per run, the number of steps to the best local minimum per run, etc. In addition, more heuristics may be adopted with the VDS algorithm to facilitate its improvement, such as the highest cumulative score heuristic [78].

Several algorithmic components of an SLS algorithm can be exploited to improve its performance. The results of the evaluation study emphasize many of these components, such as preprocessing techniques, parameter tuning, neighborhood definition and search algorithms, diversification techniques, and exploitation of the structure of instances.

Nevertheless, SAT and PMSAT solvers have been shown to be competitive for solving hard constrained combinatorial problems in many different domains with various techniques while speeding up solving, including automated software and hardware engineering problems such as fault test, detection and diagnosis [79], [80], upgradability [81], circuit design diagnosis [82], etc., that encoded as SAT and PMSAT problems and solved by SAT solvers [83] and PMSAT solvers [84]–[86]. However, robustness and correctness are essential criteria, since these solvers are used as core decision engines and optimization methods [87]. Automated software engineering approaches, such as combinatorial testing (CT), can be used as systematic techniques that detect faults and failures in the software under testing (SUT) [83].

Thank you for the team of the Supercomputing Laboratory at King Abdullah University of Science & Technology (KAUST) for their support and kindness. For computer time, this research used SHAHEEN hpc, one of the resources of the Supercomputing Laboratory at KAUST in Thuwal, Saudi Arabia.

APPENDIX A
UNWEIGHTED BENCHMARK USED FOR INITIAL TUNING

See Table 14.

APPENDIX B
WEIGHTED BENCHMARK USED FOR INITIAL TUNING

See Table 15.

APPENDIX C
UNWEIGHTED FAMILY-BASED BENCHMARKS

See Tables 17–21.

APPENDIX D
WEIGHTED FAMILY-BASED BENCHMARKS

See Tables 22–27.

REFERENCES

[1] S. A. Cook, “The complexity of theorem-proving procedures,” in *Proc. 3rd Annu. ACM Symp. Theory Comput. - STOC*, 1971, pp. 151–158.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[3] E. Demirovic and N. Musliu, “Modeling high school timetabling as partial weighted maxSAT,” in *Proc. LaSh 4th Workshop Logic Search (SAT/ICLP) Workshop FLoC*, Jul. 2014, pp. 1–39.

[4] E. Demirović, N. Musliu, and F. Winter, “Modeling and solving staff scheduling with partial weighted maxSAT,” *Ann. Oper. Res.*, vol. 275, no. 1, pp. 79–99, Apr. 2019.

[5] F. Juma, E. I. Hsu, and S. A. McIlraith, “Preference-based planning via MaxSAT,” in *Advances in Artificial Intelligence* (Lecture Notes in Computer Science), L. Kosseim and D. Inkpen, Eds. Berlin, Germany: Springer, 2012, pp. 109–120.

[6] H. Xu, R. A. Rutenbar, and K. Sakallah, “Sub-SAT: A formulation for relaxed Boolean satisfiability with applications in routing,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 6, pp. 814–820, Jun. 2003.

[7] Y. Chen, S. Safarpour, J. Marques-Silva, and A. Veneris, “Automated design debugging with maximum satisfiability,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 11, pp. 1804–1817, Nov. 2010.

[8] D. M. Strickland, E. Barnes, and J. S. Sokol, “Optimal protein structure alignment using maximum cliques,” *Oper. Res.*, vol. 53, no. 3, pp. 389–402, Jun. 2005.

[9] S. Miyazaki, K. Iwama, and Y. Kambayashi, “Database queries as combinatorial optimization problems,” in *Proc. Int. Symp. Cooperat. Database Syst. Adv. Appl.*, 1996, pp. 448–454.

[10] K. L. Sadowski, P. A. N. Bosman, and D. Thierens, “On the usefulness of linkage processing for solving MAX-SAT,” in *Proc. 15th Annu. Conf. Genetic Evol. Comput. Conf. GECCO*, 2013, pp. 853–860.

[11] R. Martins, V. Manquinho, and I. Lynce, “Open-WBO: A modular MaxSAT solver,” in *Theory Applications of Satisfiability Testing—SAT* (Lecture Notes in Computer Science), C. Sinz and U. Egly, Eds. Cham, Switzerland: Springer, 2014, pp. 438–445.

[12] P. Saikko, J. Berg, and M. Jarvisalo, “LMHS: A SAT-IP hybrid MaxSAT solver,” in *Theory and Applications of Satisfiability Testing (SAT)* (Lecture Notes in Computer Science), N. Creignou and D. L. Berre, Eds. Cham, Switzerland: Springer, 2016, pp. 539–546.

[13] C. Ansótegui, J. Gabàs, and J. Levy, “Exploiting subproblem optimization in SAT-based MaxSAT algorithms,” *J. Heuristics*, vol. 22, no. 1, pp. 1–53, Feb. 2016.

[14] C. Ansótegui and J. Gabàs, “WPM3: An (in)complete algorithm for weighted partial MaxSAT,” *Artif. Intell.*, vol. 250, pp. 37–57, Sep. 2017.

[15] F. Bacchus, A. Hyttinen, M. Järvisalo, and P. Saikko, “Reduced cost fixing in MaxSAT,” in *Principles and Practice of Constraint Programming* (Lecture Notes in Computer Science), J. C. Beck, Ed. Cham, Switzerland: Springer, 2017, pp. 641–651.

[16] E. Demirovic and P. Stuckey, “Local-style search in the linear MaxSAT algorithm: A computational study of solution-based phase saving,” in *Proc. Pragmatics SAT Workshop*, 2018.

[17] J. Berg, E. Demirović, and J. Peter Stuckey, “Core-boosted linear search for incomplete MaxSAT,” in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (Lecture Notes in Computer Science), L.-M. Rousseau and K. Stergiou, Ed. Cham, Switzerland: Springer, 2019, pp. 39–56.

[18] A. Morgado, F. Heras, M. Liffiton, J. Planes, and J. Marques-Silva, “Iterative and core-guided MaxSAT solving: A survey and assessment,” *Constraints*, vol. 18, no. 4, pp. 478–534, Oct. 2013.

[19] C. Ansótegui, M. L. Bonet, and J. Levy, “SAT-based MaxSAT algorithms,” *Artif. Intell.*, vol. 196, pp. 77–105, Mar. 2013.

[20] C. Ansotegui, F. Didier, and J. Gabas, “Exploiting the structure of unsatisfiable cores in MaxSAT,” in *IJCAI*, 2015.

[21] A. Ignatiev, A. Morgado, and J. Marques-Silva, “RC2: An efficient MaxSAT solver,” *J. Satisfiability, Boolean Model. Comput.*, vol. 11, no. 1, pp. 53–64, Sep. 2019.

[22] J. Gu, “Efficient local search for very large-scale satisfiability problems,” *ACM SIGART Bull.*, vol. 3, no. 1, pp. 8–12, Jan. 1992.

[23] B. Selman, H. Levesque, and D. Mitchell, “A new method for solving hard satisfiability problems,” in *Proc. 10th Nat. Conf. Artif. Intell. (AAAI)*, 1992, pp. 440–446.

[24] B. Selman and H. Kautz, “Domain-independent extensions to GSAT: Solving large structured satisfiability problems,” in *Proc. 13th Int. Joint Conf. Artifical Intell. (IJCAI)*, Chambery, France: Morgan Kaufmann, Aug. 1993, pp. 290–295.

[25] B. Selman, A. H. Kautz, and B. Cohen, “Noise strategies for improving local search,” in *Proc. 12th Nat. Conf. Artif. Intell. (AAAI)*, Menlo Park, CA, USA: American Association for Artificial Intelligence, vol. 1, 1994, pp. 337–343.

[26] M. El Bachir Menai and T. N. Al-Yahya, "A taxonomy of exact methods for partial max-SAT," *J. Comput. Sci. Technol.*, vol. 28, no. 2, pp. 232–246, Mar. 2013.

[27] P. Hansen and B. Jaumard, "Algorithms for the maximum satisfiability problem," *Computing*, vol. 44, no. 4, pp. 279–303, Dec. 1990.

[28] R. Battiti and M. Protasi, "Approximate algorithms and heuristics for MAX-SAT," in *Handbook of Combinatorial Optimization*, D.-Z. D. Panos and M. Pardalos, Eds. New York, NY, USA: Springer, 1998, pp. 77–148.

[29] Z. Wu and W. Benjamin Wah, "Trap escaping strategies in discrete Lagrangian methods for solving hard satisfiability and maximum satisfiability problems," in *Proc. 16th Nat. Conf. Artif. Intell. Eleventh Innov. Appl. Artif. Intell. Conf. Innov. Appl. Artif. Intell. (AAAI)*. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1999, pp. 673–678.

[30] T. Stutzle, H. Hoos, and A. Roli, "A review of the literature on local search algorithms for MAX-SAT," Allen Inst. AI, Seattle, WA, USA, Tech. Rep., 2001.

[31] H. H. Hoos and T. Stutzle, "Stochastic local search algorithms: An overview," in *Springer Handbook of Computational Intelligence*, J. Kacprzyk and W. Pedrycz, Eds. Berlin, Germany: Springer, 2015, pp. 1085–1105.

[32] M. Sevaux, K. Sorensen, and N. Pillay, "Adaptive and multilevel metaheuristics," in *Handbook Heuristics*, R. Marti, P. Panos, and M. G. C. Resende, Eds. Springer, 2018, pp. 1–19.

[33] H. Kautz, A. Sabharwal, and B. Selman, "Incomplete algorithms," in *Handbook of Satisfiability*, vol. 185. Amsterdam, The Netherlands: IOS Press, 2009, pp. 185–203.

[34] R. Marti, M. P. Pardalos, and G. C. M. Resende, "Local search," in *Handbook Heuristics*. Springer, 2018.

[35] S. Cai, C. Luo, J. Thornton, and K. Su, "Tailoring local search for partial MaxSAT," in *Proc. 28th AAAI Conf. Artif. Intell. (AAAI)*, 2014, pp. 2623–2629.

[36] C. Luo, S. Cai, W. Wu, Z. Jie, and K. Su, "CCLS: An efficient local search algorithm for weighted maximum satisfiability," *IEEE Trans. Comput.*, vol. 64, no. 7, pp. 1830–1843, Jul. 2015.

[37] Z. Lei and S. Cai, "Solving (weighted) partial MaxSAT by dynamic local search for SAT," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, Jul. 2018, pp. 1346–1352.

[38] C. S. Kambhampati and T. Liu, "Phase transition and network structure in realistic SAT problems," in *Proc. 27th AAAI Conf. Artif. Intell. (AAAI)*, Bellevue, WA, USA: AAAI Press, 2013, pp. 1619–1620.

[39] F. Glover, "New ejection chain and alternating path methods for traveling salesman problems," in *Computer Science and Operations Research*, O. Balci, R. Sharda, and S. A. Zenios, Eds. New York, NY, USA: Pergamon, 1992, pp. 491–509.

[40] C. Rego and F. Glover, "Local search and metaheuristics," in *The Traveling Salesman Problem and its Variations, Combinatorial Optimization*, G. G. Abraham and P. Punnen, Eds. New York, NY, USA: Springer, 2007, pp. 309–368.

[41] J. Thornton, D. N. Pham, S. Bain, and V. Ferreira, "Additive versus multiplicative clause weighting for SAT," in *Proc. 19th Nat. Conf. Artif. Intell. (AAAI)*, 2004, pp. 191–196.

[42] B. Cha, K. Iwama, Y. Kambayashi, and S. Miyazaki, "Local search algorithms for partial MAXSAT," in *Proc. AAAI/IAAI*, 1997, pp. 263–268.

[43] D. N. Pham, J. Thornton, C. Gretton, and A. Sattar, "Combining adaptive and dynamic local search for satisfiability," *J. Satisfiability, Boolean Model. Comput.*, vol. 4, nos. 2–4, pp. 149–172, May 2008.

[44] Y. Kilani, M. Bsoul, A. Alsarhan, and I. Obeidat, "Improving PAWS by the island confinement method," in *Artificial Intelligence and Soft Computing* (Lecture Notes in Computer Science), L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, Eds. Berlin, Germany: Springer, 2012, pp. 662–670.

[45] S. Cai and K. Su, "Local search for Boolean satisfiability with configuration checking and subscore," *Artif. Intell.*, vol. 204, pp. 75–98, Nov. 2013.

[46] A. Frohlich, A. Biere, C. Wintersteiger, and Y. Hamadi, "Stochastic local search for satisfiability modulo theories," in *Proc. 29th AAAI Conf. Artif. Intell. (AAAI)*, 2015, pp. 1136–1143.

[47] F. Hutter, A. D. D. Tompkins, and H. H. Hoos, "Scaling and probabilistic smoothing: Efficient dynamic local search for SAT," in *Principles and Practice of Constraint Programming—(CP)* (Lecture Notes in Computer Science). vol. 2470, P. Van Hentenryck, Ed. Berlin, Germany: Springer, 2002, pp. 233–248.

[48] S. Cai, C. Luo, J. Lin, and K. Su, "New local search methods for partial MaxSAT," *Artif. Intell.*, vol. 240, pp. 1–18, Nov. 2016.

[49] S. Cai, "Balance between complexity and quality: Local search for minimum vertex cover in massive graphs," in *Proc. 24th Int. Conf. Artif. Intell. (IJCAI)*, 2015, pp. 747–753.

[50] S. Cai, K. Su, and A. Sattar, "Local search with edge weighting and configuration checking heuristics for minimum vertex cover," *Artif. Intell.*, vol. 175, nos. 9–10, pp. 1672–1696, Jun. 2011.

[51] C. Luo, S. Cai, K. Su, and W. Huang, "CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability," *Artif. Intell.*, vol. 243, pp. 26–44, Feb. 2017.

[52] Y. Chu, C. Luo, W. Huang, H. You, and D. Fan, "Hard neighboring variables based configuration checking in stochastic local search for weighted partial maximum satisfiability," in *Proc. IEEE 29th Int. Conf. Tools Artif. Intell. (ICTAI)*, Nov. 2017, pp. 139–146.

[53] C. Luo, S. Cai, K. Su, and W. Wu, "Clause states based configuration checking in local search for satisfiability," *IEEE Trans. Cybern.*, vol. 45, no. 5, pp. 1028–1041, May 2015.

[54] S. Cai, C. Luo, and H. Zhang, "From decimation to local search and back: A new approach to MaxSAT," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, Aug. 2017, pp. 571–577.

[55] S. Cai and Z. Lei, "Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability," *Artif. Intell.*, vol. 287, Oct. 2020, Art. no. 103354.

[56] *MaxSAT Evaluation 2018 Homepage*, MaxSAT, Carnegie Mellon Univ., Pittsburgh, PA, USA, 2018.

[57] F. Hutter, H. Holger Hoos, and T. Stutzle, "Automatic algorithm configuration based on local search," in *Proc. 22nd Nat. Conf. Artif. Intell.*, vol. 2, 2007, pp. 1152–1157.

[58] E. Montero, M.-C. Riff, and N. Rojas-Morales, "Tuners review: How crucial are set-up values to find effective parameter values," *Eng. Appl. Artif. Intell.*, vol. 76, pp. 108–118, Nov. 2018.

[59] H. H. Hoos, "Automated algorithm configuration and parameter tuning," in *Autonomous Search*, Y. Hamadi, E. Monfroy, and F. Saubion, Eds. Berlin, Germany: Springer, 2012, pp. 37–71.

[60] F. Hutter, H. Holger Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization* (Lecture Notes in Computer Science), C. A. C. Coello, Ed. Berlin, Germany: Springer, 2011, pp. 507–523.

[61] V. A. Tatsis and K. E. Parsopoulos, "Dynamic parameter adaptation in metaheuristics using gradient approximation and line search," *Appl. Soft Comput.*, vol. 74, pp. 368–384, Jan. 2019.

[62] D. McAllester, B. Selman, and H. Kautz, "Evidence for invariants in local search," in *Proc. 14th Nat. Conf. Artif. Intell. 9th Conf. Innov. Appl. Artif. Intell. (AAAI/IAAI)*, 1997, pp. 321–326.

[63] D. J. Patterson and H. Kautz, "Auto-walksat: A self-tuning implementation of walksat," *Electron. Notes Discrete Math.*, vol. 9, pp. 360–368, Jun. 2001.

[64] P. R. Brent, *Algorithms for Minimization Without Derivatives; Dover Books on Mathematics*. Mineola, NY, USA: Dover, Apr. 2013.

[65] H. H. Hoos, "An adaptive noise mechanism for walkSAT," in *Proc. 18th Nat. Conf. Artif. Intell.*, Menlo Park, CA, USA: American Association for Artificial Intelligence, 2002, pp. 655–660.

[66] C. M. Li, W. Wei, and H. Zhang, "Combining adaptive noise and lookahead in local search for SAT," in *Trends in Constraint Programming*. Hoboken, NJ, USA: Wiley, 2010, pp. 261–267.

[67] Z. Lü and J.-K. Hao, "Adaptive memory-based local search for MAX-SAT," *Appl. Soft Comput.*, vol. 12, no. 8, pp. 2063–2071, Aug. 2012.

[68] S. Prestwich, "Tuning local search by average-reward reinforcement learning," in *Learning and Intelligent Optimization* (Lecture Notes in Computer Science), V. Maniezzo, R. Battiti, and J.-P. Watson, Eds. Berlin, Germany: Springer, 2008, pp. 192–205.

[69] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen, "A survey of very large-scale neighborhood search techniques," *Discrete Appl. Math.*, vol. 123, nos. 1–3, pp. 75–102, Nov. 2002.

[70] D. Pisinger and S. Ropke, "Large neighborhood search," in *Handbook Metaheuristics* (International Series in Operations Research & Management Science), M. Gendreau and J.-Y. Potvin, Eds. New York, NY, USA: Springer, 2010, pp. 399–419.

[71] N. Bouhmala, "A variable neighborhood walksat-based algorithm for MAX-SAT problems," *Sci. World J.*, vol. 2014, pp. 1–11, Jan. 2014.

[72] MaxSAT. (2018). *Evaluation—History*. [Online]. Available: https://maxsat-evaluations.github.io/2018/history.html

[73] MaxSAT. (2049). *Evaluation Benchmarks*. [Online]. Available: https://maxsat-evaluations.github.io/2019/benchmarks.html

[74] N. Een and A. Biere, ‘‘Effective preprocessing in SAT through variable and clause elimination,’’ in *Theory Applications of Satisfiability Testing* (Lecture Notes in Computer Science), F. Bacchus and T. Walsh, Eds. Berlin, Germany: Springer, 2005, pp. 61–75.

[75] N. Manthey, ‘‘Coprocessor 2.0—A flexible CNF simplifier,’’ in *Theory Applications of Satisfiability Testing (SAT)* (Lecture Notes in Computer Science), A. Cimatti and R. Sebastiani, Eds. Berlin, Germany: Springer, 2012, pp. 436–441.

[76] D. N. Pham, J. Thornton, and A. Sattar, ‘‘Building structure into local search for SAT,’’ in *Proc. 20th Int. Joint Conf. Artif. Intell. (IJCAI)*, San Francisco, CA, USA: Morgan Kaufmann, Jan. 2007, pp. 2359–2364.

[77] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham, ‘‘Understanding random SAT: Beyond the clauses-to-variables ratio,’’ in *Principles and Practice of Constraint Programming—(CP)* (Lecture Notes in Computer Science), M. Wallace, Ed. Berlin, Germany: Springer, 2004, pp. 438–452.

[78] N. Bouhmala, ‘‘A variable depth search algorithm for binary constraint satisfaction problems,’’ *Math. Problems Eng.*, vol. 2015, pp. 1–10, Jan. 2015.

[79] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, ‘‘Fault diagnosis and logic debugging using Boolean satisfiability,’’ *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 10, pp. 1606–1621, Oct. 2005.

[80] A. Riefert, M. Sauer, S. Reddy, and B. Becker, ‘‘Improving diagnosis resolution of a fault detection test set,’’ in *Proc. IEEE 33rd VLSI Test Symp. (VTS)*, Apr. 2015, pp. 1–6.

[81] J. Argelich, I. Lynce, and J. Marques-Silva, ‘‘On solving Boolean multilevel optimization problems,’’ in *Proc. 21st Int. Jont Conf. Artif. Intell. (IJCAI)*, San Francisco, CA, USA: Morgan Kaufmann, Jul. 2009, pp. 393–398.

[82] J. Marques-Silva, M. Janota, A. Ignatiev, and A. Morgado, ‘‘Efficient model based diagnosis with maximum satisfiability,’’ in *Proc. 24th Int. Conf. Artif. Intell. (IJCAI)*, Buenos Aires, AR, USA: AAAI Press, Jul. 2015, pp. 1966–1972.

[83] H. Wu, N. Changhai, J. Petke, Y. Jia, and M. Harman, ‘‘Comparative analysis of constraint handling techniques for constrained combinatorial testing,’’ *IEEE Trans. Softw. Eng.*, early access, Nov. 26, 2019, doi: 10.1109/TSE.2019.2955687.

[84] C. Ansotegui, I. Izquierdo, F. Manyà, and J. Torres-Jimenez, *A Max-SAT-Based Approach to Constructing Optimal Covering Arrays*. Amsterdam, The Netherlands: IOS Press, Oct. 2013.

[85] X. Si, X. Zhang, R. Grigore, and M. Naik, ‘‘Maximum satisfiability in software analysis: Applications and techniques,’’ in *Computer Aided Verification* (Lecture Notes in Computer Science), R. Majumdar and V. Kunčak, Eds., Cham, Switzerland: Springer, 2017, pp. 68–94.

[86] A. Yamada, T. Kitamura, C. Artho, E.-H. Choi, Y. Oiwa, and A. Biere, ‘‘Optimization of combinatorial testing by incremental SAT solving,’’ in *Proc. IEEE 8th Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2015, pp. 1–10.

[87] R. Brummayer, F. Lonsing, and A. Biere, ‘‘Automated testing and debugging of SAT and QBF solvers,’’ in *Theory Applications of Satisfiability Testing—(SAT)* (Lecture Notes in Computer Science), O. Strichman and S. Szeider, Eds. Berlin, Germany: Springer, 2010, pp. 44–57.

HAIFA HAMAD ALKASEM received the B.Sc. and M.Sc. degrees in computer science from King Saud University, in 2000 and 2009, respectively, where she is currently pursuing the Ph.D. degree. She is also a Lecturer with the Department of Computer Science, Imam Muhammad Ibn Saud University. Her research interests include satisfiability problems, metaheuristics, and e-learning.

MOHAMED EL BACHIR MENAI received the Ph.D. degree in computer science from the Mentouri University of Constantine, Algeria, and the University of Paris VIII, France, in 2005, and the Ph.D. degree Habilitation Universitaire in computer science from the Mentouri University of Constantine, in 2007 (it is the highest academic qualification in Algeria, France, and Germany). He is currently a Professor with the Department of Computer Science, King Saud University, Saudi Arabia. His main research interests include satisfiability problems, evolutionary computing, machine learning, and natural language processing.