Artificial Intelligence 338 (2025) 104242

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint

Integrating multi-armed bandit with local search for MaxSAT

Jiongzhi Zheng [a,b], Kun He [a,b,*], Jianrong Zhou [a,b], Yan Jin [a,b], Chu-Min Li [c], Felip Manyà [d]

[a] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
[b] Hopcroft Center on Computing Science, Huazhong University of Science and Technology, Wuhan 430074, China
[c] MIS, University of Picardie Jules Verne, Amiens 80039, France
[d] Artificial Intelligence Research Institute (IIIA), CSIC, Bellaterra 08193, Spain

A R T I C L E I N F O

Keywords:
Maximum satisfiability
Local search
Multi-armed bandit
Hybrid decimation

A B S T R A C T

Partial MaxSAT (PMS) and Weighted PMS (WPMS) are two practical generalizations of the MaxSAT problem. In this paper, we introduce a new local search algorithm for these problems, named BandHS. It applies two multi-armed bandit (MAB) models to guide the search directions when escaping local optima. One MAB model is combined with all the soft clauses to help the algorithm select to satisfy appropriate soft clauses, while the other MAB model is combined with all the literals in hard clauses to help the algorithm select suitable literals to satisfy the hard clauses. These two models enhance the algorithm's search ability in both feasible and infeasible solution spaces. BandHS also incorporates a novel initialization method that prioritizes both unit and binary clauses when generating the initial solutions. Moreover, we apply our MAB approach to the state-of-the-art local search algorithm NuWLS and to the local search component of the incomplete solver NuWLS-c-2023. The extensive experiments conducted demonstrate the excellent performance and generalization capability of the proposed method. Additionally, we provide analyses on the type of problems where our MAB method works well or not, aiming to offer insights and suggestions for its application. Encouragingly, our MAB method has been successfully applied in core local search components in the winner of the WPMS complete track of MaxSAT Evaluation 2023, as well as the runners-up of the incomplete track of MaxSAT Evaluations 2022 and 2023.

1. Introduction

As an optimization extension of the well-known Boolean Satisfiability (SAT) decision problem, the Maximum Satisfiability (MaxSAT) problem aims at finding a complete truth assignment that satisfies as many clauses as possible in a given propositional formula in Conjunctive Normal Form (CNF) [1]. Partial MaxSAT (PMS) is a variant of MaxSAT where the clauses are classified as hard or soft, and its goal is to maximize the number of satisfied soft clauses with the constraint that all the hard clauses must be satisfied. Associating a positive weight to each soft clause in PMS results in Weighted PMS (WPMS), whose goal is to maximize the total weight of satisfied soft clauses while all the hard clauses are satisfied. Both PMS and WPMS, denoted as (W)PMS, have practical applications in diverse fields such as planning [2], combinatorial testing [3], group testing [4], and timetabling [5], etc.

* Corresponding author at: School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China.
E-mail addresses: jzzheng@hust.edu.cn (J. Zheng), brooklet60@hust.edu.cn (K. He).

https://doi.org/10.1016/j.artint.2024.104242
Received 29 November 2022; Received in revised form 31 December 2023; Accepted 20 October 2024

Available online 30 October 2024

Solvers for (W)PMS are categorized as complete or incomplete based on their capacity to provide optimality guarantees. Complete solvers include branch-and-bound [6–10] and SAT-based solvers [11–16]. SAT-based solvers, which solve (W)PMS by iteratively calling a SAT solver, are one of the most popular and best-performing approaches for (W)PMS, especially in industrial instances. Moreover, they can be easily modified to function as incomplete solvers by returning a new solution once a better solution is found [15, 16]. Branch-and-bound solvers incorporate effective inference rules and lower bound computation methods and are particularly efficient on random instances [17,18]. Recently, Li et al. [10,19] showed that branch-and-bound MaxSAT solvers can become highly competitive in industrial when they integrate clause learning.

Incomplete MaxSAT algorithms mostly focus on local search methods [20–26]. Although local search solvers are not as good as SAT-based solvers in solving large industrial instances, they exhibit promising performance in solving random and crafted instances. Besides, the combination of local search methods with complete solvers has demonstrated great potential in solving both SAT [27] and MaxSAT [25,28] problems.

Local search algorithms typically flip the Boolean value of a selected variable at each step when exploring the solution space. Two essential techniques in local search are the clause weighting scheme and the variable selection strategy. Recent high-performing local search (W)PMS solvers incorporate effective clause weighting schemes [29,23,30,25,31]. Nevertheless, their variable selection strategies have inherent limitations, especially when falling into local optima. Note that flipping any single variable in a local minimum does not improve the current solution.

When falling into an infeasible local optimum (i.e., there are falsified hard clauses), these algorithms [29,23,30,25,31] first randomly select a falsified hard clause and then satisfy it by flipping one of its variables. This is reasonable given that all hard clauses must be satisfied. However, when falling into a feasible local optimum, these algorithms still use the random strategy to determine the soft clause to be satisfied at the current step. This may not be an optimal strategy for the following reasons: 1) Not all soft clauses must be satisfied, and 2) the high degree of randomness may lead to a small probability for these algorithms to find a good search direction (satisfying a falsified soft clause corresponds to a search direction). For instance, suppose that, in the current solution, there are p falsified soft clauses and q soft clauses among them are satisfied by an optimal solution. Then, the random strategy leads the search direction far away from the global optimum with a probability of $1-q/p$. Note that the value of p can be close to one million in some cases, and the value of q might be very small, especially when the current solution has higher quality, making the probability $1-q/p$ close to 1. Therefore, simply relying on the random strategy may be a common limitation across various local search MaxSAT algorithms, underscoring the need for developing effective alternative strategies.

To address the mentioned limitation and provide a more effective and generic strategy, we propose employing a multi-armed bandit (MAB) [32,33] approach to help local search algorithms satisfy appropriate soft clauses. In an MAB model, the agent must select which arm to pull (i.e., perform an action) at each decision step (i.e., state), resulting in associated rewards. These rewards serve as feedback to evaluate the efficacy of pulling each arm, and the estimated values guide the agent in deciding which arm to pull in subsequent steps. Fundamentally, MAB, as a reinforcement learning technique, enables the program to learn an optimal policy for selecting the most appropriate item from a set of multiple candidates. In this paper, we introduce a novel MAB model, called *soft* MAB, in which each arm corresponds to a soft clause of the instance. When the search falls into a feasible local optimum, *Soft* MAB replaces the random strategy and determines the soft clause to be satisfied. Additionally, we conduct a detailed analysis of the performance of *soft* MAB across various (W)PMS instances. The results suggest that it is particularly effective in instances where it is more likely to have a large value of $1-q/p$, providing valuable intuitions and insights for our method.

Soft MAB enhances the algorithm's search ability in feasible solution spaces. To achieve a more comprehensive improvement, we propose a *hard* MAB model to further improve the algorithm's search ability in infeasible solution spaces. While satisfying all hard clauses is imperative, a key aspect lies in determining which literal to satisfy in each hard clause. Building on this insight, we propose to combine *soft* MAB with *hard* MAB. This hybrid approach assists the algorithm in selecting appropriate literals to satisfy hard clauses, and in finding feasible solutions faster. The resulting local search algorithm, named BandHS, embodies both *hard* MAB and *soft* MAB.

Moreover, inspired by the studies for SAT and MaxSAT that prioritize both unit and binary clauses (i.e., clauses with exactly one and two literals, respectively) over other clauses [34–36], we propose a novel decimation approach known as hybrid decimation (HyDeci). This approach prefers the satisfaction of both unit and binary clauses during the generation of the initial assignment in BandHS. The decimation method is a category of incomplete approaches that proceed by assigning the Boolean value of some (usually one) variables sequentially and simplifying the formula accordingly [37]. Decimation approaches that focus on unit clauses have been applied in MaxSAT [37,25]. However, it is the first time, to our knowledge, that a MaxSAT decimation method concentrates on both unit and binary clauses.

Several studies have explored the application of MAB techniques in the context of MaxSAT and SAT. For example, Goffinet and Ramanujan [38] proposed a Monte-Carlo tree search algorithm for MaxSAT. In their approach, a two-armed bandit is associated with each variable (node in the search tree) to determine the Boolean value assigned to the branching variable. Lassouaoui et al. [39] utilized an MAB model to select low-level heuristics in a hyper-heuristic framework for MaxSAT, where pulling an arm corresponds to choosing a specific low-level heuristic. Cherif et al. [40] introduced the use of a two-armed bandit to select the branching heuristic, determining the next variable to branch on in a SAT solver. While these methods have achieved notable success in solving MaxSAT and SAT problems, our work proposes two novel MAB models for (W)PMS, representing a significant advancement over existing state-of-the-art local search (W)PMS solvers. Notably, this is the first time where an MAB model is associated with all soft clauses, and another MAB model is linked with all literals in hard clauses within a local search (W)PMS solver.

This paper is an extended and improved version of our conference paper [26], in which we introduced the BandMaxSAT algorithm with the HyDeci initialization algorithm and the *soft* MAB model. In this paper, we introduce the *hard* MAB model in BandHS and conduct a more comprehensive empirical analysis of the proposed methods. Our proposed MAB and HyDeci methods are applicable across various local search MaxSAT algorithms, and BandHS is an improvement of SATLike3.0 [25] incorporating these methods. To assess the generalization capability of the core MAB method, we apply it to the state-of-the-art local search algorithm NuWLS [31] and to the local search component of the incomplete solver NuWLS-c-2023 [28], the winner in all four incomplete tracks of MaxSAT Evaluation (MSE) 2023. Extensive experiments demonstrate that BandHS significantly outperforms SATLike3.0, and our MAB method enhances the performance of both NuWLS and NuWLS-c-2023, underscoring its generalization capability and excellent performance.

The main contributions of this work are as follows:

- We introduce the integration of an MAB model involving all soft clauses for improving the performance of local search (W)PMS algorithms, and another MAB model involving all literals in hard clauses for further improving their performance. The proposed MAB method is applicable to any local search MaxSAT algorithms, offering the ability to select suitable search directions for escaping local optima. Notably, our MAB method has been successfully applied in core components of the winner of the WPMS complete track in MSE2023, WMaxCDCL [41], as well as the runners-up of the incomplete tracks of MSE2022 and MSE2023, DT-HyWalk [42] and NuWLS-c-Band [43].
- We introduce a novel decimation method for (W)PMS, named HyDeci, which prioritizes the satisfaction of both unit and binary clauses during the initial solution construction. HyDeci proves to be effective in generating high-quality initial assignments and has the potential to enhance other local search MaxSAT algorithms.
- We conduct extensive experiments that demonstrate the outstanding performance and generalization capability of our proposed methods. They significantly elevate the state-of-the-art in local search and incomplete (W)PMS solvers, underscoring the promising application of MAB in MaxSAT-based problem solving. Additionally, we provide detailed analyses to highlight instances where our MAB method excels or faces challenges, offering valuable insights and recommendations for its application.

The rest of this paper is organized as follows. Section 2 introduces preliminary concepts. Section 3 details the proposed BandHS algorithm, encompassing the HyDeci initialization method and the *soft* and *hard* MAB models. Section 4 evaluates the performance of the proposed methods. Section 5 concludes the paper with final remarks.

2. Preliminaries

Given a set of Boolean variables $\{x_1, ..., x_n\}$, a literal is either a variable itself x_i or its negation $\neg x_i$; a clause is a disjunction of literals, i.e., $c_j = l_{j1} \vee ... \vee l_{jn_j}$, where n_j is the number of literals in clause c_j. A Conjunctive Normal Form (CNF) formula $\mathcal{F}$ is a conjunction of clauses, i.e., $\mathcal{F} = c_1 \wedge ... \wedge c_m$. A complete assignment A represents a mapping that maps each variable to a value of 1 (true) or 0 (false). A literal x_i (resp. $\neg x_i$) is satisfied if the current assignment maps x_i to 1 (resp. 0). A clause is satisfied by the current assignment if there is at least one satisfied literal in the clause.

Given a CNF formula $\mathcal{F}$, SAT is a decision problem that aims to determine whether there is an assignment that satisfies all the clauses in $\mathcal{F}$, and MaxSAT aims to find an assignment that satisfies as many clauses in $\mathcal{F}$ as possible. Given a CNF formula $\mathcal{F}$ whose clauses are divided into hard and soft clauses, PMS is a variant of MaxSAT that aims to find an assignment that satisfies all the hard clauses meanwhile maximizing the number of satisfied soft clauses in $\mathcal{F}$, and WPMS is a generalization of PMS where each soft clause is associated with a positive weight. The goal of WPMS is to find an assignment that satisfies all the hard clauses while maximizing the total weight of satisfied soft clauses in $\mathcal{F}$. In the local search algorithms for MaxSAT, the flipping operator for a variable is an operator that changes its Boolean value.

Given a (W)PMS instance $\mathcal{F}$, a complete assignment A is feasible if it satisfies all the hard clauses in $\mathcal{F}$. The cost of A, denoted as $cost(A)$, is set to $+\infty$ for convenience if A is infeasible. Otherwise, $cost(A)$ is equal to the number of falsified soft clauses for PMS and equal to the total weight of falsified soft clauses for WPMS.

In addition, the effective clause weighting technique is widely used in recent well-performing (W)PMS local search algorithms [29, 24,25,44]. Algorithms with this technique associate dynamic weights (independent of the original soft clause weights in WPMS instances) to clauses and use the dynamic weights to guide the search direction. BandHS also applies the clause weighting technique and maintains dynamic weights to both hard and soft clauses with the clause weighting strategy used in SATLike3.0 [25].

Given a (W)PMS instance $\mathcal{F}$, the current assignment A, and the dynamic clause weights, the commonly used scoring function for a variable x, denoted as $score(x)$, is defined as the increment of the total dynamic weight of satisfied clauses caused by flipping x in A. Moreover, a local optimum for (W)PMS indicates that there are no variables with positive *score*. A local optimum is feasible if there are no falsified hard clauses and infeasible otherwise.

3. Methodology

BandHS consists of the proposed hybrid decimation (HyDeci) initialization process and the local search process. The proposed *hard* and *soft* MAB models are trained and used during the local search process to guide the search directions when escaping from local optima. This section first presents the main framework of the proposed BandHS algorithm, and subsequently presents its components, including the proposed two MAB models and the HyDeci method.

Algorithm 1: BandHS.

Input: A (W)PMS instance $\mathcal{F}$, cut-off time *cutoff*, BMS parameter k, reward delay steps d, reward discount factor γ, number of sampled arms $ArmNum$, exploration bias parameter λ

Output: A feasible assignment A of $\mathcal{F}$, or *no feasible assignment found*

1 $A := \text{HyDeci}(\mathcal{F})$;
2 initialize $A^* := A$, $A' := A$, $N^s := 0$;
3 initialize $H := +\infty$, $H' := +\infty$, $N^h := 0$;
4 **while** *running time* $<$ *cutoff* **do**
5   **if** *A is feasible & cost(A) < cost(A^*)* **then**
6     $A^* := A$;
7   **if** $D := \{x \mid score(x) > 0\} \neq \emptyset$ **then**
8     $v :=$ a variable in D picked by BMS(k);
9   **else**
10   update_clause_weights();
11   **if** $\exists$ *falsified hard clauses* **then**
12       $H :=$ the number of falsified hard clauses in A;
13       $c :=$ a random falsified hard clause;
14   **if** *A^* is infeasible* **then**
15   update_hard_estimated_value(H, H', d, γ);
16         $N^h := N^h + 1$, $H' := H$;
17         $l :=$ PickHardArm(c, N^h, λ);
18         $t_l^h := t_l^h + 1$;
19         $v :=$ the variable corresponds to literal l;
20   **else**
21         $v :=$ the variable with the highest *score* in c;
22   **else**
23   update_soft_estimated_value(A, A', A^*, d, γ);
24       $N^s := N^s + 1$, $A' := A$;
25       $c :=$ PickSoftArm($ArmNum, N^s, \lambda$);
26       $t_c^s := t_c^s + 1$;
27       $v :=$ the variable with the highest *score* in c;
28   $A := A$ with v flipped;
29 **if** *A^* is feasible* **then return** A^*;
30 **else return** *no feasible assignment found*;

3.1. Main framework of BandHS

Before presenting the main process of BandHS, we introduce some concepts and definitions. In BandHS, each arm in the *hard* MAB model corresponds to a literal in hard clauses, and each arm in the *soft* MAB model corresponds to a soft clause. We associate an estimated value with each arm in the MAB models to evaluate the benefits that may yield upon being pulled, i.e., satisfying its corresponding literal or soft clause. For each arm l (resp. i) in the *hard* (resp. *soft*) MAB model, we define V_l^h (resp. V_i^s) as its estimated value, initialized to 1, and t_l^h (resp. t_i^s) as the number of times it has been pulled. Intuitively, the larger the estimated value of an arm in the *hard* MAB model, satisfying its corresponding literal leads to fewer falsified hard clauses. The larger the estimated value of an arm in the *soft* MAB model, satisfying its corresponding soft clause leads to better feasible solutions. Additionally, we define N^h and N^s as the number of times the algorithm falls into infeasible and feasible local optima, respectively.

The main process of BandHS is depicted in Algorithm 1. BandHS first uses the HyDeci method (refer to Algorithm 4) to generate an initial assignment (line 1) and then iteratively selects a variable to flip until the cut-off time is reached (lines 4-28). When local optima are not reached, BandHS selects a variable to be flipped using the Best from Multiple Selections (BMS) strategy [45]. BMS chooses k random variables (with replacement) and returns one with the highest *score* (lines 7-8). Upon falling into a local optimum, BandHS first updates the dynamic clause weights, using the update_clause_weight() function (line 10), according to the clause weighting scheme in SATLike3.0 [25] and then selects the variable to flip in the current step.

If the local optimum is infeasible, BandHS first randomly selects a falsified hard clause c as the clause to be satisfied (line 13). Then, if no feasible solutions have been found (lines 14-19), *hard* MAB will be called to pick an arm l among all the arms (i.e., literals) in c using function PickHardArm() (refer to Algorithm 2). The variable to be flipped in the current step is the variable that corresponds to literal l. If BandHS finds any feasible solution, it will not call *hard* MAB to select the variable to be flipped but selects it greedily according to the scoring function (line 21).

If the local optimum is feasible (lines 22-27), BandHS first calls the *soft* MAB model to select the soft clause c to be satisfied in the current step using function PickSoftArm() (refer to Algorithm 3), and then selects to flip the variable with the highest *score* in c in the current step (line 27).

Moreover, note that once an arm is picked, the related information, including the estimated values, the number of pulled times, and the number of times to fall into local optima, will be updated (lines 15-18 and 23-26).

In summary, the *hard* and *soft* MAB models are separately trained and used in BandHS. Prior to finding any feasible solution, BandHS focuses on training and using the *hard* MAB model to help it find feasible solutions faster. Once a feasible solution is found, BandHS focuses on training and using the *soft* MAB model to help it find better quality solutions. We stop training *hard* MAB after

Algorithm 2: PickHardArm(c, N^h, λ).

Input: A random falsified hard clause c, number of times to fall into an infeasible local optimum N^h, exploration bias parameter λ
Output: The arm selected to be pulled l

1 initialize $U_*^h := -\infty$;
2 **for** *each literal* j *in* c **do**
3   calculate U_j^h according to Eq. (1);
4   **if** $U_j^h > U_*^h$ **then** $U_*^h := U_j^h$, $l := j$;
5 **return** l;

finding a feasible solution because the transformation of searching between feasible and infeasible solution spaces could mislead the reward function and the estimated values of the arms in *hard* MAB. Consequently, the *hard* and *soft* MAB models enhance the search capabilities of BandHS in infeasible and feasible solution spaces, respectively.

3.2. The hard and soft MAB models

In the previous subsection, we learned about the conditions triggering the invocation of the two MAB models and their roles in determining the literal or soft clause to be satisfied. This subsection introduces details of the MAB models, including the arm selection process and the methodology for updating estimated values.

3.2.1. Arm selection strategy

BandHS adopts the Upper Confidence Bound method [46] to trade-off between exploration and exploitation and determine which arms to pull. Specifically, the upper confidence bound U_l^h on the estimated value V_l^h of arm l in the *hard* MAB model is calculated using the following equation:

$$U_l^h = V_l^h + \lambda \cdot \sqrt{\frac{ln(N^h)}{t_l^h + 1}}, \tag{1}$$

where λ is the exploration bias parameter, N^h is the number of times the algorithm falls into infeasible local optima, and t_l^h is the number of times arm l has been pulled in the *hard* MAB model.

Similarly, the upper confidence bound U_i^s on the estimated value V_i^s of arm i in the *soft* MAB model is calculated using the following equation:

$$U_i^s = V_i^s + \lambda \cdot \sqrt{\frac{ln(N^s)}{t_i^s + 1}}, \tag{2}$$

where N^s is the number of times the algorithm falls into feasible local optima, and t_i^s is the number of times arm i has been pulled in the *soft* MAB.

The procedures for selecting an arm to be pulled in the *hard* or *soft* MAB models, i.e., functions PickHardArm() and PickSoftArm(), are outlined in Algorithms 2 and 3, respectively. As the proposed MAB models encompass a large number of arms (equal to the number of literals in hard clauses or the number of soft clauses), selecting the best among them is inefficient. To address this, we propose using a sampling strategy to reduce the selection scope and enhance the algorithm's efficiency. When selecting an arm to be pulled in the *hard* MAB model, i.e., selecting a falsified literal in hard clauses, the selection of the random falsified hard clause c (where all the literals are falsified) in line 13 of Algorithm 1 can be regarded as a natural and reasonable sampling strategy given that all hard clauses must be satisfied. PickHardArm() actually returns an arm (i.e., literal) with the highest upper confidence bound in the input clause c.

When selecting an arm to be pulled in the *soft* MAB model, the PickSoftArm() function initially samples $ArmNum$ (20 by default) candidate arms and then selects an arm with the highest upper confidence bound among such candidates. Note that the *soft* MAB model aims at selecting a soft clause to be satisfied in the current step. Hence, the arms corresponding to the soft clauses that are satisfied by the current assignment will not be considered as candidates. Similar sampling strategies have been used in MAB problems [47] and certain combinatorial optimization problems [45]. The experimental results also attest that the sampling strategy used in BandHS significantly enhances the algorithm's performance.

3.2.2. Estimated value updating strategy

In general, pulling a high-quality arm in the *hard* MAB model results in fewer falsified hard clauses and can help the algorithm find feasible solutions faster. Similarly, pulling a high-quality arm in the *soft* MAB model contributes to the generation of better feasible solutions. Consequently, we use the change in the number of falsified hard clauses and the change in the cost values (refer to Section 2) to respectively design the reward functions for updating the estimated values of the arms in the *hard* and *soft* MAB models.

Specifically, let H' and H represent the numbers of falsified hard clauses of the previous and current infeasible local optimal solutions, respectively, and let l denote the previous pulled arm in the *hard* MAB model. A simple reward for pulling l can be set to $H' - H$. However, reducing the number of falsified hard clauses from 20 to 10 is much harder and more significant than reducing it

Algorithm 3: PickSoftArm($ArmNum, N^s, \lambda$).

Input: Number of sampled arms $ArmNum$, number of times to fall into a feasible local optimum N^s, exploration bias parameter λ
Output: The arm selected to be pulled c

initialize $U^s_* := -\infty$;
for $i := 1$ *to* $ArmNum$ **do**
    $j :=$ a random falsified soft clause;
    calculate U^s_j according to Eq. (2);
    if $U^s_j > U^s_*$ **then** $U^s_* := U^s_j$, $c := j$;
return c;

Algorithm 4: HyDeci($\mathcal{F}$).

Input: A (W)PMS instance $\mathcal{F}$
Output: A complete assignment A of variables in $\mathcal{F}$

while $\exists$ *unassigned variables* **do**
    if $\exists$ *hard unit clauses* **then**
        $c :=$ a random hard unit clause, satisfy c and SIMPLIFY;
    else if $\exists$ *soft unit clauses* **then**
        $c :=$ a random soft unit clause, satisfy c and SIMPLIFY;
    else if $\exists$ *hard binary clauses* **then**
        $c :=$ a random hard binary clause;
        $l :=$ a greedily selected unassigned literal in c, satisfy l and SIMPLIFY;
    else if $\exists$ *soft binary clauses* **then**
        $c :=$ a random soft binary clause;
        $l :=$ a greedily selected unassigned literal in c, satisfy l and SIMPLIFY;
    else
        $v :=$ a random unassigned variable, assign v a random value and SIMPLIFY;
return the resulting complete assignment A;

from 1000 to 990. Consequently, the rewards of these two cases should not be identical. To address this issue, we define the reward as follows:

$$r^h(H, H') = (H' - H)/H'. \tag{3}$$

Examining Eq. (3), we note that if we assume $H' - H$ to be constant, the smaller the value of H', the more rewards the action of pulling the previous arm in the *hard* MAB model can yield, which is reasonable and intuitive.

Similarly, let A' and A represent the previous and current feasible local optimal solutions, respectively, and let c denote the previous arm pulled in the *soft* MAB. The reward for pulling arm c is defined as follows:

$$r^s(A, A', A^*) = \frac{cost(A') - cost(A)}{cost(A') - cost(A^*) + 1}, \tag{4}$$

where A^* is the best solution found so far. If we consider $cost(A') - cost(A)$ to be constant, the closer $cost(A')$ and $cost(A^*)$, the more rewards the action of pulling the previous arm in the *soft* MAB can yield.

Additionally, as the arms in the *hard* MAB model are connected by the hard clauses, and the arms in the *soft* MAB model are connected by the variables, we assume that the arms in our MAB models are not independent of each other. We also believe that the improvement (or deterioration) of H over H' or A over A' may be attributed not solely to the previous action but also to earlier actions. Hence, we apply the delayed reward method [48] to update the estimated value of the latest d (35 by default) pulled arms once a reward is obtained.

Specifically, let H' and H represent the numbers of falsified hard clauses in the previous and current infeasible local optimal solutions, respectively, and let $l_1, \ldots, l_d$ denote the set of the latest d pulled arms, with l_d being the most recent one, in the *hard* MAB model. The estimated values of these d arms in the *hard* MAB model are updated as follows:

$$V^h_{l_i} = V^h_{l_i} + \gamma^{d-i} \cdot r^h(H, H'), \quad i \in \{1, \ldots, d\}, \tag{5}$$

where γ is the reward discount factor and $r^h(H, H')$ is calculated by Eq. (3).

Similarly, let A' and A denote the previous and current feasible local optimal solutions, respectively; let A^* represent the best solution found so far; and let $a_1, \ldots, a_d$ represent the set of the latest d pulled arms, with a_d being the most recent one, in the *soft* MAB model. The estimated values of these d arms in the *soft* MAB model are updated as follows:

$$V^s_{a_i} = V^s_{a_i} + \gamma^{d-i} \cdot r^s(A, A', A^*), \quad i \in \{1, \ldots, d\}, \tag{6}$$

where $r^s(A, A', A^*)$ is calculated by Eq. (4).

Functions update_hard_estimated_values() and update_soft_estimated_values() in lines 15 and 23 of Algorithm 1 specifically handle the updating of estimated values for the latest pulled d arms based on Eqs. (5) and (5), respectively.

3.3. Hybrid decimation

Finally, we present the proposed HyDeci initialization method, which is an effective decimation approach designed to prioritize the satisfaction of both unit and binary clauses. Given that clauses with shorter lengths are easier to be falsified, a preference for satisfying shorter clauses aims to diminish the number of falsified clauses, thereby yielding high-quality initial assignments. The procedural details of HyDeci are outlined in Algorithm 4, with the term SIMPLIFY denoting the process of simplifying the formula after assigning a value to a variable.

HyDeci systematically generates the initial complete assignment through iterative steps. In each iteration, HyDeci assigns the value of exactly one variable. In the presence of unit clauses, HyDeci randomly selects a unit clause (with hard clauses taking precedence) and proceeds to satisfy it. In the absence of unit clauses but with binary clauses, HyDeci initially samples a random binary clause c (with hard clauses taking precedence) and selects one of the two unassigned literals in c following a greedy strategy: It selects a literal that results in more satisfied soft clauses or a greater total weight of satisfied soft clauses. If there are no unit or binary clauses, HyDeci randomly picks an unassigned variable and assigns a Boolean value to it.

In summary, the principal improvement offered by the HyDeci algorithm, compared to existing decimation approaches [37,25], lies in that HyDeci not only focuses on unit clauses but also on binary clauses.

4. Experiments

Our solver BandHS was implemented on top of SATLike3.0 [25] and integrates the MAB and HyDeci methods proposed in this paper. Additionally, we applied the core MAB approach to enhance the state-of-the-art local search algorithm NuWLS [31] and the local search component of the incomplete solver NuWLS-c-2023 [28], winner of all the four incomplete tracks of MaxSAT Evaluation (MSE) 2023. This enhancement involves invoking the *hard* and *soft* MAB models when falling into local optima to select appropriate search directions. The resulting solvers are named NuWLS-BandHS and NuWLS-c-2023-BandHS, respectively. These are the solvers that we have developed for the empirical investigation.

This section begins by comparing BandHS and NuWLS-BandHS with their respective basic local search algorithms, SATLike3.0 and NuWLS. Then, we use the comparison results between NuWLS-BandHS and NuWLS to analyze the efficacy of the MAB method across various instance types. Subsequently, we conduct a comparative analysis involving NuWLS-c-2023-BandHS and leading incomplete solvers, such as NuWLS-c-2023, SATLike-c [49], Loandra [50], and TT-Open-WBO-Inc [51]. Finally, comprehensive ablation studies are performed to assess the impact of various components and strategies within BandHS, encompassing the HyDeci initialization method, the *hard* and *soft* MAB models, the sampling strategy for arm selection, and the delayed reward method.

4.1. Experimental setup

All the algorithms were implemented in C++ and compiled by g++, and the experiments were conducted on a server equipped with an Intel® Xeon® E5-2650 v3 2.30 GHz 10-core CPU and 256 GB RAM, running Ubuntu 16.04 Linux operation system. We evaluated the algorithms on all the (W)PMS instances from the incomplete track of the six recent MaxSAT Evaluations (MSEs), i.e., MSE2018 to MSE2023. Note that the benchmarks containing all PMS/WPMS instances from the incomplete track of MSE2023 are named PMS_2023/WPMS_2023, and so forth. Each instance was processed once by each algorithm with two time limits: 60 and 300 seconds. This is consistent with the settings of the incomplete track of MSEs.

We adopt two kinds of metrics to compare and evaluate the algorithms. The first one is the number of winning instances, represented by '*#win.*', which indicates the number of instances in which the algorithm yields the best solution among all the algorithms in the table. The metric '*#win.*' has been widely used in comparing local search MaxSAT algorithms [25,31,26,52]. The second one is the scoring function used in the incomplete track of MSEs. The score of a solver for an instance is 0 if the solver cannot find feasible solutions, and $(BKC+1)/(cost(A)+1)$ otherwise, where A is its output feasible solution, and BKC is the best-known cost of the instance. The score of a solver for a benchmark is its average score upon all contained instances, represented by '*#score.*'. The best results appear in bold in the tables.

Parameters related to the MAB method in BandHS include the reward delay steps d, the reward discount factor γ, the number of sampled arms in the *soft* MAB $ArmNum$, and the exploration bias parameter λ. We adopt an automatic configurator called SMAC3 [53] to tune the parameters based on (W)PMS instances in the incomplete track of MSE2017. The tuning domains of the above parameters in BandHS are $d \in \{5, 10, \cdots, 50\}$, $\gamma \in \{0.5, 0.55, \cdots, 0.95, 0.99\}$, $ArmNum \in \{5, 10, \cdots, 50\}$, and $\lambda \in \{0.1, 0.25, 0.5, 1, 2.5, 5, 10, 25, 50, 100\}$. The final settings of these parameters are $d = 35$, $\gamma = 0.5$, $ArmNum = 20$, and $\lambda = 2.5$. Other parameters in BandHS are equal to those in the basic SATLike3.0 algorithm. Parameters related to the MAB method in NuWLS-Band, NuWLS-c-2023-BandHS, and variant algorithms of BandHS in the ablation study are also tuned by SMAC3. Note that SMAC3 is also used for tuning the baseline solvers SATLike3.0, NuWLS, and NuWLS-c-2023. Thus, the baseline solvers in our experiments keep their tuned default parameters. The codes of BandHS and NuWLS-BandHS are available at https://github.com/JHL-HUST/BandHS/.

Table 1
Comparison of BandHS and SATLike3.0 under two time limits of 60 s and 300 s.

Benchmark	*#inst.*	BandHS (60 s)			SATLike3.0 (60 s)			BandHS (300 s)			SATLike3.0 (300 s)		
		#score.	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*
PMS_2018	153	**0.6570**	**104**	15.467	0.5563	56	17.203	**0.7164**	**113**	65.595	0.6072	56	63.735
PMS_2019	299	**0.6736**	**206**	15.227	0.6136	135	12.800	**0.7094**	**215**	56.725	0.6484	140	51.561
PMS_2020	262	**0.6935**	**171**	13.774	0.6247	111	12.112	**0.7157**	**180**	56.543	0.6579	115	53.830
PMS_2021	155	**0.6302**	**105**	15.227	0.5530	60	7.591	**0.6530**	**112**	58.996	0.5889	60	47.335
PMS_2022	179	**0.6794**	**126**	14.311	0.6104	61	9.976	**0.7126**	**133**	60.254	0.6455	61	48.717
PMS_2023	179	**0.5698**	**125**	19.151	0.4947	44	17.938	**0.6174**	**130**	84.841	0.5580	53	116.249
WPMS_2018	172	**0.6970**	**117**	18.046	0.6652	45	11.453	**0.7316**	**122**	108.758	0.6987	50	81.125
WPMS_2019	297	**0.6543**	**196**	21.084	0.6105	93	15.866	**0.7182**	**223**	103.880	0.6780	98	74.991
WPMS_2020	253	**0.6358**	**155**	19.908	0.6181	81	21.785	**0.7105**	**176**	102.467	0.6900	90	80.110
WPMS_2021	151	0.5595	**68**	29.500	**0.5612**	52	26.394	**0.6552**	**90**	118.976	0.6487	55	85.492
WPMS_2022	197	**0.6499**	**110**	23.727	0.6173	51	20.951	**0.7273**	**124**	108.442	0.7079	57	104.751
WPMS_2023	160	**0.5520**	**75**	26.091	0.5387	49	22.946	**0.6424**	**99**	123.500	0.6328	56	114.679

Table 2
Comparison of NuWLS-BandHS and NuWLS under two time limits of 60 s and 300 s.

Benchmark	*#inst.*	NuWLS-BandHS (60 s)			NuWLS (60 s)			NuWLS-BandHS (300 s)			NuWLS (300 s)		
		#score.	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*
PMS_2018	153	**0.7093**	**96**	19.291	0.6864	72	16.613	**0.7540**	**98**	77.786	0.7413	81	83.328
PMS_2019	299	**0.6984**	**193**	15.142	0.6942	164	16.144	**0.7355**	**194**	59.248	0.7326	177	68.053
PMS_2020	262	**0.7272**	**163**	17.334	0.7180	133	16.968	**0.7578**	**176**	72.131	0.7517	143	72.698
PMS_2021	155	**0.6473**	**97**	13.869	0.6406	82	11.944	**0.6950**	**101**	62.144	0.6861	90	56.358
PMS_2022	179	**0.7242**	**113**	17.057	0.7094	90	17.250	**0.7581**	**118**	67.429	0.7546	97	65.379
PMS_2023	179	0.6209	**98**	19.090	**0.6336**	94	21.093	0.6764	**108**	99.992	**0.6792**	98	97.762
WPMS_2018	172	**0.7494**	**103**	22.241	0.7374	84	20.318	**0.7735**	**103**	101.473	0.7702	91	109.023
WPMS_2019	297	**0.7049**	**184**	22.109	0.6701	135	21.164	**0.7654**	**183**	94.073	0.7419	162	100.342
WPMS_2020	253	**0.7096**	**153**	23.925	0.6918	111	22.644	**0.7954**	**147**	114.898	0.7854	139	107.286
WPMS_2021	151	**0.6153**	**87**	24.113	0.5889	51	24.137	**0.7104**	**91**	102.057	0.6889	66	105.771
WPMS_2022	197	**0.6899**	**106**	26.483	0.6799	84	24.122	**0.7619**	**108**	111.168	0.7593	98	100.322
WPMS_2023	160	**0.5855**	**93**	24.598	0.5447	54	17.073	**0.6664**	**93**	100.619	0.6558	71	95.602

4.2. Comparison with baseline local search algorithms

We first compare the performance of BandHS and NuWLS-BandHS against their baseline local search MaxSAT algorithms, SATLike3.0 and NuWLS, across all the tested instances. The results are summarized in Tables 1 and 2, respectively. Column *#inst.* indicates the number of instances in each benchmark. Column *time* represents the average running time (in seconds) to yield the *#win.* instances.

The results presented in Tables 1 and 2 highlight the substantial and consistent improvement of our method over the baseline algorithms across various MSE benchmarks and time limits (60 s or 300 s) for both PMS and WPMS. Specifically, the number of *#win.* instances of BandHS is 31-184% greater than that of SATLike3.0, and the number of *#win.* instances of NuWLS-BandHS is 4-72% greater than that of NuWLS. Regarding the scoring function, BandHS (resp. NuWLS-BandHS) achieves lower performance than SATLike3.0 (resp. NuWLS) on only 1 (resp. 2) out of all 24 benchmarks (12 benchmarks with two time limits). These results indicate that our method not only enhances the local search algorithms on more instances but also improves the overall solution quality.

To obtain a more detailed comparison between the algorithms and evaluate the performance of our method in different instance classes, we collect all the tested instances (duplicated instances are excluded) and compare the algorithms on each instance class. Ties of the algorithms with the same number of *#win.* instances are broken by selecting the one with less running time, aligned with the rules in MSEs. The results of the comparison between BandHS and SATLike3.0 on PMS and WPMS instance classes are presented in Tables 3 and 4, respectively. The comparison between NuWLS-BandHS and NuWLS on PMS and WPMS instance classes is detailed in Tables 5 and 6, respectively. Note that we exclude the instance classes where both algorithms fail to produce feasible solutions.

The results indicate that BandHS (resp. NuWLS-BandHS) consistently outperforms SATLike3.0 (resp. NuWLS) on most classes of both PMS and WPMS instances, irrespective of the 60 s or 300 s time limit, according to the *#win.* and *#score.* metrics. Specifically, for all the 132 classes of (W)PMS instances (66 classes with two time limits), BandHS (resp. NuWLS-BandHS) surpasses SATLike3.0 (resp. NuWLS) on 104 (resp. 85) classes according to the *#win.* metric, and on 109 (resp. 81) classes according to the *#score.* metric. This performance highlights the efficacy and robustness of our method in enhancing the state-of-the-art local search algorithms across diverse instance classes.

4.3. Analysis of the MAB method in different instances

In this subsection, we aim to explore the effectiveness of our proposed MAB method across different instances, offering insights into our method and providing suggestions on its utilization.

Table 3
Comparison of BandHS and SATLike3.0 in each PMS instance class under two time limits of 60 s and 300 s.

PMS instance class	*#inst.*	BandHS (60 s)			SATLike3.0 (60 s)			BandHS (300 s)			SATLike3.0 (300 s)		
		#score.	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*
aes	6	**0.8174**	**4**	8.592	0.8003	1	22.894	**0.9254**	**5**	54.266	0.9122	2	132.759
atcoss	14	**0.0083**	**1**	6.694	0.0000	0	0.000	**0.0083**	**1**	6.694	0.0000	0	0.000
bcp	24	**0.7979**	**16**	19.065	0.7166	9	29.946	**0.8514**	**15**	80.439	0.7676	11	131.226
causal-discovery	3	**0.4245**	3	**2.956**	**0.4245**	3	3.813	**0.4245**	3	**2.956**	**0.4245**	3	3.813
close_solutions	14	0.4799	7	13.535	**0.6228**	**9**	19.667	0.5714	8	82.751	**0.7023**	**10**	53.843
decision-tree	38	**0.2568**	**36**	10.341	0.0411	5	21.401	**0.2787**	**38**	26.235	0.0731	5	65.678
des	13	0.0000	0	0.000	**0.0769**	**1**	28.594	0.0000	0	0.000	**0.1538**	**2**	78.938
extension-enforcement	19	**0.8834**	14	27.047	0.8770	**16**	15.759	**0.9712**	**16**	88.112	0.9427	11	83.823
fault-diagnosis	8	**0.7018**	**8**	13.617	0.0000	0	0.000	**0.7018**	**8**	13.617	0.0000	0	0.000
gen-hyper-tw	37	**0.7599**	**28**	20.058	0.6951	20	19.234	**0.8883**	**31**	105.227	0.8135	27	101.251
hs-timetabling	1	**0.0765**	**1**	1.744	0.0000	0	0.000	**0.0765**	**1**	1.744	0.0000	0	0.000
inconsistency-measurement	25	**0.8603**	**20**	28.155	0.8534	10	33.728	**0.8841**	**21**	140.003	0.8755	7	143.815
judgment-aggregation	24	**0.9683**	**22**	23.025	0.9522	8	19.066	**0.9777**	**23**	64.647	0.9647	10	106.978
large-graph-community	3	**0.7373**	**3**	8.581	0.4955	2	11.025	**0.7373**	**3**	8.581	0.4955	2	11.025
logic-synthesis	1	**0.8739**	**1**	0.750	0.8220	0	0.000	**0.8739**	**1**	0.750	0.8291	0	0.000
maxclique & maxcut	68	0.9872	56	10.351	**0.9883**	**66**	2.128	**0.9884**	64	32.125	0.9883	**65**	2.136
MaximumCommonSub-GraphExtraction	25	0.9628	18	9.018	**0.9722**	**22**	13.943	0.9785	19	47.018	**0.9833**	**23**	54.975
MaxSATQueriesinInterpretableClassifiers	45	**0.9210**	**31**	14.716	0.8002	23	8.100	**0.9507**	**33**	81.017	0.8842	29	71.601
mbd	6	**0.6561**	**4**	16.580	0.6453	2	34.866	0.6826	3	**94.875**	**0.7253**	3	99.238
min-fill	19	**0.6091**	**15**	16.649	0.3756	5	39.139	**0.6943**	**15**	51.625	0.4616	5	58.856
optic	17	**0.9748**	**16**	20.077	0.9178	1	8.250	**0.9761**	**16**	68.543	0.9226	1	150.075
optimizing-BDDs	21	**0.3688**	**15**	22.481	0.2723	3	14.581	**0.5051**	**14**	141.086	0.4734	10	167.078
phylogenetic-trees	14	**0.0339**	**1**	43.800	0.0000	0	0.000	**0.1611**	**4**	195.633	0.0000	0	0.000
pseudoBoolean	11	**0.0887**	**1**	34.106	0.0831	0	0.000	**0.0890**	**1**	242.738	0.0834	0	0.000
railroad_reisch	9	**0.9616**	**9**	7.181	0.9614	6	5.288	**0.9653**	**9**	70.827	0.9650	5	6.113
railway-transport	4	**0.4784**	**2**	37.894	0.3970	1	26.119	**0.5114**	**3**	117.988	0.4823	1	287.213
ramsey	14	**1.0000**	14	0.131	**1.0000**	14	**0.096**	**1.0000**	14	0.131	**1.0000**	14	**0.096**
ran-scp	14	**0.9591**	**14**	15.224	0.9181	2	5.766	**0.9683**	**14**	51.814	0.9281	2	56.991
scheduling	5	**0.5628**	**4**	47.780	0.5452	2	27.797	**0.6214**	**4**	233.250	0.6093	2	107.756
scheduling_xiaojuan	20	**0.9416**	**20**	26.249	0.9165	3	22.269	**0.9606**	**20**	113.079	0.9282	2	187.969
SeanSafarpour	13	**0.5868**	**9**	18.444	0.5537	8	17.531	**0.6277**	**8**	102.490	0.6065	7	116.831
set-covering	9	**0.9720**	**8**	20.292	0.9106	2	31.556	**0.9810**	**9**	73.527	0.9123	1	31.950
setcover-rail_zhendong	4	0.9868	1	1.688	**0.9921**	**4**	2.845	0.9889	2	135.750	**0.9940**	**4**	64.777
treewidth-computation	9	**0.9389**	**8**	20.161	0.9142	4	16.195	**0.9639**	**9**	42.200	0.9404	6	164.869
uaq	20	**1.0000**	**20**	8.128	0.9987	18	11.810	**1.0000**	**20**	8.128	0.9993	19	23.537
uaq_gazzarata	4	**0.9350**	**4**	43.556	0.8775	1	20.756	**0.9725**	**4**	81.727	0.8892	1	20.756
xai-mindset2	19	**0.3148**	**13**	9.506	0.0512	1	47.850	**0.4008**	**14**	68.598	0.1756	1	210.056
Total	600	**0.7207**	**447**	16.203	0.6573	272	12.757	**0.7592**	**473**	66.942	0.7017	291	62.518

As mentioned in Section 1, consider that, in the current feasible local optimal solution, there are p falsified soft clauses in the current solution and, among them, q soft clauses are satisfied by the optimal solution. The random strategy commonly used in local search MaxSAT algorithms leads the search direction far away from the global optimum with a probability of $1-q/p$. The proposed (*soft*) MAB method uses an MAB model to help the algorithm in selecting an appropriate search direction for escaping from local optima. Therefore, we believe that our MAB method is more effective in instances where it is more likely to have a large value of $1-q/p$. To substantiate our hypothesis, we select two characteristics to represent and describe the (W)PMS instances and assess the performance of our method in instances with different characteristics. The first one is the *soft-clause percentage*, which represents the ratio of the number of soft clauses to the total number of clauses. The second one is the *cost percentage*, which represents the ratio of the best-known cost to the total weight of soft clauses. Intuitively, larger values of *soft-clause percentage* and *cost percentage* may result in larger values of p and $1-q/p$. Hence, we hypothesize that our MAB method prefers instances with higher *soft-clause percentage* and *cost percentage*, and we aim to validate this hypothesis through our analysis.

We analyze the comparison results between NuWLS-BandHS and NuWLS under a 300 s time limit because NuWLS is the best-performing local search MaxSAT algorithm, and NuWLS-BandHS solely utilizes the MAB method without the HyDeci method. Among the 1,211 distinct tested instances (total instances in Tables 5 and 6), 92 instances yield no feasible solutions for both algorithms, 450 instances produce identical solutions, NuWLS-BandHS outperforms NuWLS in 388 instances, and NuWLS outperforms NuWLS-BandHS in 281 instances. Additionally, 74 (resp. 51) instances show that NuWLS-BandHS (resp. NuWLS) achieves a score higher than NuWLS (resp. NuWLS-BandHS) by at least 0.05. Furthermore, 11 (resp. 8) instances demonstrate that NuWLS-BandHS (resp. NuWLS) obtains a score higher than NuWLS (resp. NuWLS-BandHS) by at least 0.2. Note that our analysis mainly focuses on the instances that can distinguish the algorithm performance.

Fig. 1 illustrates the distribution on instances based on the *soft-clause percentage* and *cost percentage* characteristics. The difference in scores indicates the score of NuWLS-BandHS minus the score of NuWLS. In Fig. 1(a) (resp. 1(b)), each point with coordinates (x, y) represents a (W)PMS instance with *soft-clause percentage* (resp. *cost percentage*) of x and the difference in scores if y (with $|y| \geq 0.05$ to focus on the distinct instances). In Fig. 1(c), each red (resp. blue) point with coordinates (x, y) represents a (W)PMS instance

Table 4
Comparison of BandHS and SATLike3.0 in each WPMS instance class under two time limits of 60 s and 300 s.

WPMS instance class	#inst.	BandHS (60 s)			SATLike3.0 (60 s)			BandHS (300 s)			SATLike3.0 (300 s)		
		#score.	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*
abstraction-refinement	10	**0.9535**	**10**	171.97	0.9160	0	0.00	0.3113	3	55.32	**0.5069**	3	**49.79**
af-synthesis	33	**0.8981**	**33**	119.05	0.7684	0	0.00	**0.8615**	**33**	16.34	0.7370	0	0.00
BTBNSL-Rounded	26	**0.8423**	**20**	77.60	0.8347	6	42.10	**0.8375**	**18**	9.13	0.8201	8	15.49
causal-discovery	27	**0.4050**	**25**	45.94	0.2716	14	7.66	**0.3724**	**23**	11.38	0.2508	14	5.13
cluster-expansion	21	**0.9964**	**13**	146.87	0.9964	8	94.71	**0.9959**	**13**	6.87	0.9957	8	0.07
correlation-clustering	50	0.7425	**32**	96.89	**0.7616**	18	134.47	0.6901	**34**	31.33	**0.7139**	16	34.51
decision-tree	48	**0.6952**	**30**	118.97	0.6941	27	141.51	**0.6633**	25	29.34	0.6623	**30**	30.58
hs-timetabling	13	0.0312	2	**115.88**	**0.0314**	2	169.73	**0.0264**	**4**	47.91	0.0248	0	0.00
judgment-aggregation	5	**0.9736**	**3**	156.31	0.9666	2	172.78	**0.9513**	**4**	16.34	0.9468	2	38.74
lisbon-wedding	21	**0.4699**	7	209.10	0.4647	**8**	109.09	0.3888	6	30.34	**0.4056**	**8**	31.87
maxcut	29	**0.9985**	**29**	14.83	0.9962	27	1.01	**0.9983**	**29**	1.68	0.9960	27	1.01
max-realizability	13	**0.7692**	**10**	42.19	0.7520	8	45.21	**0.7656**	**10**	13.07	0.7258	4	8.16
MaxSATQueriesinInterpretableClassifiers	40	**0.8770**	**24**	103.26	0.8388	17	87.18	**0.8400**	**22**	11.82	0.8140	18	17.42
metro	2	0.6485	0	0.00	**0.7830**	**2**	233.63	0.6163	0	0.00	**0.6710**	**2**	50.44
MinimumWeightDominatingSetProblem	8	0.6010	6	156.02	**0.6800**	**7**	176.26	**0.0921**	1	23.53	**0.0921**	1	**22.14**
min-width	46	**0.9727**	**44**	155.98	0.9582	2	101.15	**0.9635**	**42**	27.69	0.9467	4	27.57
mpe	22	**0.9984**	**19**	90.41	0.7242	9	79.42	**0.9938**	**16**	18.42	0.7214	10	37.08
railroad_reisch	6	**0.9928**	3	109.78	0.9925	**4**	22.39	**0.9924**	3	30.06	0.9924	**4**	22.39
railway-transport	4	0.4618	**2**	49.89	**0.5371**	1	134.27	**0.3445**	**2**	32.42	0.0775	0	0.00
ramsey	12	**0.9675**	10	62.78	0.9623	10	**14.60**	0.9265	7	20.17	**0.9315**	**11**	9.89
RBAC	91	**0.9572**	**76**	140.59	0.9332	17	98.25	**0.8730**	**68**	25.27	0.8564	26	22.10
relational-inference	3	**0.1750**	**3**	121.18	0.0960	0	0.00	**0.0933**	**1**	46.18	0.0443	0	0.00
scSequencing_Mehrabadi	14	0.3109	2	111.33	**0.5349**	**12**	84.27	0.1674	3	33.98	**0.3473**	**8**	11.53
set-covering	13	**0.9918**	**13**	24.49	0.9376	1	41.63	**0.9883**	**12**	10.05	0.9325	2	36.20
setcover-wt	4	**0.1894**	**1**	58.78	0.1854	0	0.00	**0.1894**	**1**	58.78	0.1815	0	0.00
spot5	9	**0.9937**	**9**	78.24	0.9853	1	21.77	**0.9921**	**8**	34.04	0.9805	1	21.77
staff-scheduling	11	**0.8085**	**10**	156.39	0.7677	1	81.26	**0.7813**	**9**	34.04	0.7418	2	29.32
tcp	13	0.9736	10	103.96	**0.9782**	**12**	181.64	**0.9632**	**8**	8.45	0.9617	7	16.16
timetabling	19	**0.1593**	**13**	193.35	0.1272	2	151.51	**0.1463**	**10**	41.39	0.1186	5	35.54
Total	613	**0.7895**	**459**	109.95	0.7632	218	87.54	**0.7366**	**415**	21.16	0.7125	221	20.06

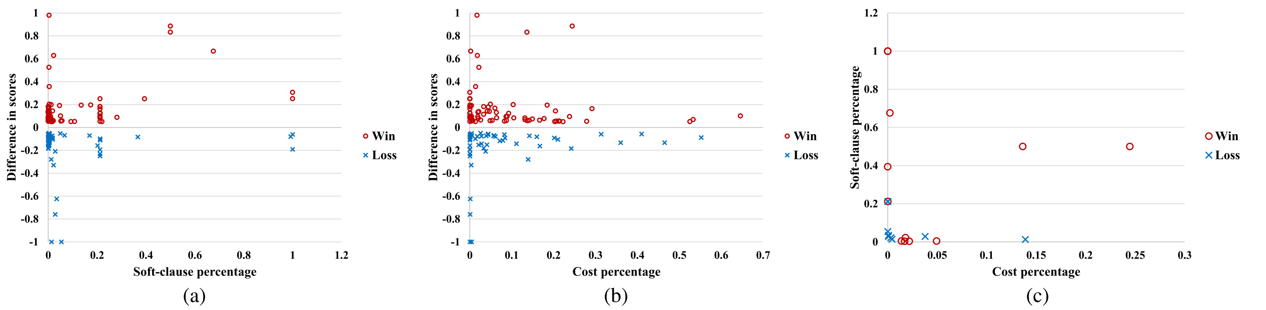

Fig. 1. Distribution on instances based on the *soft-clause percentage* and *cost percentage* characteristics. The difference in scores indicates the score of NuWLS-BandHS minus the score of NuWLS. (a) Distribution on instances based on *soft-clause percentage*. (b) Distribution on instances based on *cost percentage*. (c) Distribution on instances with the absolute value of the difference in scores greater than or equal to 0.2 based on the two characteristics. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

with its difference in scores greater than 0.2 (resp. less than -0.2), and its *cost percentage* and *soft-clause percentage* equal to x and y, respectively.

From the results, we observe that most blue *Loss* instances, especially those with a difference in scores less than -0.2, are very close to the y-axes in Figs. 1(a) and 1(b), and very close to the origin in Fig. 1(c). Specifically, 5 out of 8 instances with a difference in scores less than -0.2 are concentrated in areas with a *cost percentage* less than 0.005 and a *soft-clause percentage* less than 0.06. In contrast, areas with large *soft-clause percentage* and *cost percentage* are dominated by red *Win* instances. The results indicate that our MAB method prefers instances with large *soft-clause percentage* and *cost percentage* and shows poor performance in instances with small *soft-clause percentage* and *cost percentage*, which is consistent with our hypothesis.

In summary, when the random strategy has a high probability of leading the search direction far from the global optimum, such as when solving instances with large *soft-clause percentage* and *cost percentage*, our MAB method proves to be an effective approach for selecting a suitable search direction to escape local optima. Nevertheless, when the *soft-clause percentage* and *cost percentage* are small, the random strategy can still select good search directions with a high probability, but our MAB method may incur additional computational time, potentially reducing the algorithm efficiency.

Table 5
Comparison of NuWLS-BandHS and NuWLS in each PMS instance class under two time limits of 60 s and 300 s.

PMS instance class	*#inst.*	NuWLS-BandHS (60 s)			NuWLS (60 s)			NuWLS-BandHS (300 s)			NuWLS (300 s)		
		#score.	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*
aes	6	**0.6986**	**4**	8.944	0.6785	2	16.097	**0.9338**	4	**94.181**	0.8280	4	127.205
atcoss	14	0.0000	0	0.000	0.0000	0	0.000	0.0108	0	0.000	**0.0123**	**1**	177.900
bcp	24	**0.8073**	**15**	23.744	0.7630	12	16.833	**0.8198**	**16**	96.189	0.8168	14	121.030
causal-discovery	3	0.6551	2	29.934	**0.6697**	2	**5.831**	0.6789	1	273.506	**0.7143**	**3**	167.406
close_solutions	14	0.5303	8	9.012	**0.5885**	**10**	14.490	0.5342	9	27.550	**0.5885**	**10**	14.490
decision-tree	38	**0.3863**	29	11.908	0.3833	**30**	14.138	**0.4503**	**33**	69.748	0.4464	29	58.506
extension-enforcement	19	**0.7494**	**16**	30.012	0.6819	8	23.798	**0.8811**	**14**	126.414	0.8567	9	145.040
fault-diagnosis	8	0.6174	4	**23.939**	**0.7239**	4	32.217	**0.7930**	**6**	121.597	0.7656	2	138.638
gen-hyper-tw	37	0.8458	26	22.267	**0.8478**	**27**	20.491	**0.9054**	**30**	93.900	0.9003	28	64.191
hs-timetabling	1	**0.0915**	**1**	47.981	0.0893	0	0.000	**0.0987**	**1**	249.113	0.0955	0	0.000
inconsistency-measurement	25	0.8693	14	23.964	**0.8725**	**16**	30.600	0.8873	**17**	107.828	**0.8874**	14	129.324
judgment-aggregation	24	**0.9645**	**18**	22.331	0.9627	17	16.788	0.9740	19	**78.889**	**0.9740**	19	78.990
large-graph-community	3	0.8605	1	53.831	**0.8695**	**2**	28.472	**0.9863**	**3**	230.481	0.8976	0	0.000
logic-synthesis	1	**0.9895**	**1**	3.600	0.9307	0	0.000	**1.0000**	**1**	137.681	0.9592	0	0.000
maxclique & maxcut	68	0.9889	66	2.036	**0.9892**	**67**	3.277	0.9892	67	3.650	**0.9892**	67	**3.277**
MaximumCommonSub-GraphExtraction	25	**0.9884**	**24**	11.415	0.9790	19	14.323	0.9940	23	**30.611**	**0.9948**	23	56.448
MaxSATQueriesinInterpretableClassifiers	45	0.9623	**34**	16.065	**0.9629**	31	14.052	0.9841	**36**	70.859	**0.9845**	33	64.976
mbd	6	**0.6564**	**5**	33.214	0.6320	1	55.350	**0.6878**	**4**	119.241	0.6732	2	153.291
min-fill	19	**0.6182**	**14**	20.950	0.6037	7	33.739	**0.6986**	8	98.173	0.6790	**12**	186.717
optic	17	**0.9858**	**17**	20.472	0.9552	4	23.517	**0.9958**	**17**	87.095	0.9862	4	30.591
optimizing-BDDs	21	0.5854	9	26.490	**0.7385**	**14**	23.542	0.7358	10	103.108	**0.9717**	**15**	108.433
phylogenetic-trees	14	0.0000	0	0.000	0.0000	0	0.000	0.0494	0	0.000	**0.0589**	**1**	180.956
pseudoBoolean	11	0.0898	0	0.000	**0.0909**	**1**	14.081	0.0903	0	0.000	**0.0909**	**1**	64.294
railroad_reisch	9	**0.9916**	**7**	20.834	0.9916	4	37.673	**0.9964**	**7**	68.997	0.9963	5	114.308
railway-transport	4	**0.4645**	**3**	36.388	0.3256	0	0.000	**0.4997**	**2**	92.184	0.4783	1	166.800
ramsey	14	**1.0000**	14	**0.008**	**1.0000**	14	0.287	**1.0000**	14	**0.008**	**1.0000**	14	0.287
ran-scp	14	0.9896	12	**9.944**	**0.9908**	12	20.569	0.9951	13	33.903	**0.9971**	**14**	52.990
reversi	11	0.0000	0	0.000	0.0000	0	0.000	**0.0690**	**1**	122.119	0.0541	0	0.000
scheduling	5	**0.5808**	2	21.272	0.5791	**3**	36.600	**0.6370**	**4**	181.467	0.6145	1	37.106
scheduling_xiaojuan	20	**0.8943**	**12**	30.197	0.8904	11	34.565	0.9079	**11**	107.245	**0.9103**	10	104.970
SeanSafarpour	13	**0.6494**	8	25.748	0.6330	**10**	26.464	**0.7024**	8	60.783	0.7014	**12**	136.158
set-covering	9	**0.9978**	7	**10.749**	0.9942	7	13.706	**0.9990**	**9**	44.277	0.9951	6	8.684
setcover-rail_zhendong	4	0.9922	3	13.131	**1.0000**	**4**	21.150	**1.0000**	4	38.555	**1.0000**	4	**21.150**
treewidth-computation	9	0.9231	**7**	12.450	**0.9296**	6	20.606	0.9549	7	92.338	**0.9681**	**9**	150.460
uaq	20	0.9972	17	13.667	**0.9993**	**20**	16.402	0.9993	19	30.025	**1.0000**	**20**	21.223
uaq_gazzarata	4	**0.8894**	3	25.506	0.8807	3	**23.594**	**0.9155**	**4**	78.942	0.9050	3	49.150
xai-mindset2	19	**0.6748**	**12**	26.947	0.6186	3	20.131	**0.6818**	**11**	71.129	0.6451	6	109.263
Total	598	**0.7577**	**415**	16.112	0.7568	371	16.430	0.7934	**433**	65.194	**0.7974**	396	67.027

4.4. Comparison with baseline incomplete solvers

We have used the proposed MAB models to enhance the local search component of NuWLS-c-2023, and we refer to the resulting solver as NuWLS-c-2023-BandHS. In our evaluation, we compare NuWLS-c-2023-BandHS with top-performing incomplete (W)PMS solvers from recent MSEs, including NuWLS-c-2023 [28], SATLike-c [49], TT-Open-WBO-Inc [51], and Loandra [50]. Their detailed introductions are as follows.

- **NuWLS-c-2023**: An improvement of the winner of all the four incomplete tracks of MSE2022, NuWLS-c [44], a hybrid solver that combines an improvement of NuWLS local search algorithm with SAT-based solver TT-Open-WBO-Inc [54], winner of all the four incomplete tracks of MSE2023.
- **SATLike-c**: A hybrid solver that combines the SATLike local search algorithm with TT-Open-WBO-Inc. Its various versions won most incomplete tracks of MSE2018 to MSE2021. We select its newest two versions, SATLike-c and SATLike-ck, which use glucose [55] and kissat [56] SAT solvers to obtain the initial solution, respectively.
- **TT-Open-WBO-Inc**: Basic SAT-based incomplete solver of SATLike-c and NuWLS-c-2023, winner of WPMS incomplete track of MSE2019. We select its newest two versions, TT-Open-WBO-Inc-i and TT-Open-WBO-Inc-g, which use glucose [55] and Intel-SAT [57] as their core SAT solvers.
- **Loandra**: A SAT-based incomplete solver, winner of PMS incomplete track of MSE2019 and WPMS incomplete track of MSE2021 with 300 s of time limit.

The comparison results of the seven solvers —NuWLS-c-2023-BandHS (NuWLS-cB), NuWLS-c-2023, SATLike-c, SATLike-ck, TT-Open-WBO-Inc-i (TT-OWI-i), TT-Open-WBO-Inc-g (TT-OWI-g), and Loandra— under 60 s and 300 s of time limits are displayed in Tables 7 and 8, respectively. We observe that NuWLS-c-2023-BandHS achieves the highest score in 8 out of the 12 benchmarks under both 60 s and 300 s time limits. This makes evident the outstanding performance and generalization capability of our proposed MAB method, which also proves effective in enhancing the local search component of hybrid incomplete (W)PMS solvers.

Table 6
Comparison of NuWLS-BandHS and NuWLS in each WPMS instance class under two time limits of 60 s and 300 s.

WPMS instance class	*#inst.*	NuWLS-BandHS (60 s)			NuWLS (60 s)			NuWLS-BandHS (300 s)			NuWLS (300 s)		
		#score.	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*
abstraction-refinement	10	**0.9190**	**8**	193.29	0.9188	4	205.88	**0.4995**	**5**	49.56	0.0999	0	0.00
af-synthesis	33	**0.9916**	**28**	111.68	0.9893	23	102.52	**0.9798**	**26**	27.58	0.9736	14	23.80
BTBNSL-Rounded	26	**0.9524**	**21**	131.90	0.9503	5	197.45	**0.9465**	**20**	42.13	0.9320	6	23.57
causal-discovery	27	**0.4801**	**19**	45.26	0.4253	15	83.44	**0.4395**	**22**	12.66	0.3718	14	10.71
cluster-expansion	21	**0.9992**	**16**	24.93	0.9992	14	12.02	**0.9992**	15	**4.96**	0.9991	15	5.55
correlation-clustering	50	**0.9434**	**30**	135.76	0.9371	20	172.26	**0.8613**	**32**	35.61	0.8542	18	37.36
decision-tree	48	**0.6457**	**41**	47.08	0.6409	30	38.83	**0.6392**	**41**	13.41	0.6293	31	17.79
hs-timetabling	13	**0.0678**	**4**	161.07	0.0676	3	184.53	**0.0629**	**6**	27.22	0.0621	2	20.41
judgment-aggregation	5	**0.9992**	**5**	75.11	0.9984	2	86.80	**0.9986**	**5**	49.50	0.9936	0	0.00
lisbon-wedding	21	0.5388	5	124.59	**0.5440**	**9**	100.79	0.4778	6	33.37	**0.4906**	**8**	37.54
maxcut	29	0.9992	28	2.40	**1.0000**	**29**	3.70	0.9988	27	0.00	**0.9995**	**29**	1.13
max-realizability	13	**0.7692**	10	**4.14**	**0.7692**	10	7.61	**0.7692**	10	**4.14**	**0.7692**	10	7.61
MaxSATQueriesinInterpretableClassifiers	40	**0.9479**	**29**	106.91	0.9353	20	115.48	**0.9101**	**29**	28.27	0.8709	18	28.84
metro	2	0.7217	0	0.00	**0.8434**	**2**	111.53	0.7129	0	0.00	**0.7658**	**2**	41.73
MinimumWeightDominatingSetProblem	8	**0.6512**	**6**	223.26	0.4362	2	189.95	0.0959	0	0.00	**0.0959**	**1**	46.16
min-width	46	0.9831	20	183.10	**0.9838**	**27**	206.28	0.9765	21	30.21	**0.9765**	**25**	38.86
mpe	22	**0.9999**	**21**	4.93	0.7269	14	7.88	**0.9999**	**21**	4.93	0.7270	14	7.88
railroad_reisch	6	**0.9953**	**4**	145.53	0.9701	3	173.79	**0.9397**	**6**	19.11	0.9368	4	15.78
railway-transport	4	0.5634	1	204.98	**0.5789**	**2**	196.45	**0.3420**	**2**	39.83	0.2734	0	0.00
ramsey	12	**0.9696**	**11**	32.10	0.9537	10	17.24	0.9231	10	6.32	**0.9262**	**11**	15.76
RBAC	91	**0.9690**	**54**	138.70	0.9640	40	171.74	**0.9216**	**68**	40.37	0.8980	24	43.76
relational-inference	3	**0.0707**	1	288.43	0.0681	**2**	67.67	0.0329	0	0.00	**0.0330**	**1**	24.13
scSequencing_Mehrabadi	14	**0.9907**	**9**	185.08	0.9656	6	173.14	**0.7039**	**8**	32.26	0.6590	3	45.21
set-covering	13	**0.9995**	**13**	75.99	0.9986	11	83.65	**0.9957**	**12**	10.98	0.9889	9	20.83
setcover-wt	4	0.2054	0	0.00	**0.2255**	**1**	292.41	**0.1998**	**1**	52.03	0.1902	0	0.00
spot5	9	**1.0000**	7	50.51	1.0000	**8**	91.07	**1.0000**	**8**	14.49	1.0000	7	8.21
staff-scheduling	11	0.8025	**7**	134.22	**0.8117**	5	85.59	0.7638	6	**30.03**	**0.7664**	6	39.87
tcp	13	0.9805	5	109.33	**0.9889**	**11**	131.78	0.9645	7	32.80	**0.9677**	**8**	25.76
timetabling	19	0.2777	**8**	192.40	**0.2902**	7	115.49	0.2142	**9**	41.83	**0.2246**	6	35.50
Total	613	**0.8459**	**411**	96.38	0.8287	335	102.59	**0.7969**	**423**	24.60	0.7686	286	22.59

Table 7
Comparison of NuWLS-c-2023-BandHS (NuWLS-cB) with baseline incomplete solvers under 60 s of time limit.

Benchmark	*#inst.*	NuWLS-cB	NuWLS-c-2023	SATLike-c	SATLike-ck	TT-OWI-i	TT-OWI-g	Loandra
PMS_2018	153	**0.8130**	0.8045	0.7674	0.7728	0.7803	0.7921	0.5703
PMS_2019	299	**0.8629**	0.8579	0.8041	0.7930	0.8301	0.8456	0.6265
PMS_2020	262	**0.8362**	0.8324	0.7859	0.7760	0.8024	0.8153	0.6140
PMS_2021	155	0.7858	**0.7985**	0.7478	0.7279	0.7686	0.7838	0.4879
PMS_2022	179	**0.7745**	0.7626	0.7172	0.6996	0.7573	0.7515	0.5028
PMS_2023	179	0.7438	**0.7501**	0.6828	0.6752	0.7251	0.7345	0.4977
WPMS_2018	172	**0.8711**	0.8676	0.8590	0.8611	0.8485	0.8636	0.7284
WPMS_2019	297	0.8176	**0.8217**	0.7952	0.7954	0.8008	0.8077	0.5949
WPMS_2020	253	**0.8268**	0.8249	0.7836	0.7871	0.7910	0.7969	0.5565
WPMS_2021	151	**0.7596**	0.7529	0.6902	0.6901	0.7154	0.7154	0.3502
WPMS_2022	197	0.7397	**0.7462**	0.6830	0.6799	0.7183	0.7266	0.5339
WPMS_2023	160	**0.7231**	0.7178	0.6697	0.6597	0.6962	0.6973	0.4662

Table 8
Comparison of NuWLS-c-2023-BandHS (NuWLS-cB) with baseline incomplete solvers under 300 s of time limit.

Benchmark	*#inst.*	NuWLS-cB	NuWLS-c-2023	SATLike-c	SATLike-ck	TT-OWI-i	TT-OWI-g	Loandra
PMS_2018	153	**0.8736**	0.8641	0.8297	0.8427	0.8536	0.8517	0.7199
PMS_2019	299	**0.9139**	0.9020	0.8695	0.8560	0.9006	0.8952	0.7583
PMS_2020	262	**0.8882**	0.8792	0.8437	0.8475	0.8756	0.8710	0.7418
PMS_2021	155	0.8732	**0.8775**	0.8427	0.8204	0.8732	0.8651	0.7077
PMS_2022	179	**0.8731**	0.8674	0.8287	0.8100	0.8607	0.8633	0.7393
PMS_2023	179	**0.8401**	0.8372	0.7950	0.7709	0.8205	0.8378	0.7732
WPMS_2018	172	0.9027	0.9026	0.8944	0.9003	0.9046	**0.9087**	0.8759
WPMS_2019	297	**0.9133**	0.9118	0.8818	0.8811	0.8837	0.8747	0.7757
WPMS_2020	253	0.8986	**0.9027**	0.8507	0.8526	0.8680	0.8756	0.7439
WPMS_2021	151	0.8311	**0.8364**	0.7567	0.7527	0.7999	0.8130	0.6989
WPMS_2022	197	**0.8389**	0.8365	0.7752	0.7741	0.7864	0.8288	0.7928
WPMS_2023	160	**0.8584**	0.8547	0.8023	0.7960	0.8278	0.8084	0.8169

Table 9
Comparison of BandHS and BandMaxSAT under two time limits of 60 s and 300 s.

Benchmark	*#inst.*	BandHS (60 s)			BandMaxSAT (60 s)			BandHS (300 s)			BandMaxSAT (300 s)		
		#score.	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*
PMS_2018	153	**0.6570**	84	17.651	0.6564	**86**	15.377	**0.7164**	**96**	59.516	0.6968	92	84.058
PMS_2019	299	**0.6736**	**182**	15.318	0.6626	162	12.919	**0.7094**	**189**	53.744	0.6917	175	65.578
PMS_2020	262	**0.6935**	**154**	14.243	0.6870	149	13.373	**0.7157**	**164**	52.887	0.7107	156	64.984
PMS_2021	155	**0.6302**	89	13.872	0.6228	89	**13.627**	**0.6530**	**98**	53.962	0.6474	93	55.116
PMS_2022	179	0.6794	90	13.805	**0.6841**	**107**	15.988	0.7126	98	60.124	**0.7133**	**115**	70.457
PMS_2023	179	**0.5698**	**97**	18.810	0.5668	86	18.603	0.6174	**110**	81.257	**0.6222**	94	98.357
WPMS_2018	172	**0.6970**	**95**	16.055	0.6867	79	15.464	**0.7316**	**100**	103.161	0.7251	77	69.197
WPMS_2019	297	**0.6543**	**162**	20.940	0.6450	144	16.249	**0.7182**	**180**	98.542	0.7120	155	80.301
WPMS_2020	253	**0.6358**	**139**	20.426	0.6257	108	18.098	**0.7105**	**146**	96.743	0.7034	129	96.581
WPMS_2021	151	**0.5595**	**70**	26.833	0.5571	56	25.686	**0.6552**	**82**	123.804	0.6511	65	112.405
WPMS_2022	197	**0.6499**	**98**	23.372	0.6371	69	22.161	**0.7273**	**108**	110.139	0.7200	82	110.600
WPMS_2023	160	**0.5520**	**72**	24.167	0.5445	58	22.705	**0.6424**	**85**	129.743	0.6413	64	115.913

Table 10
Comparison of BandHS and BandHS-NoSoft under two time limits of 60 s and 300 s.

Benchmark	*#inst.*	BandHS (60 s)			BandHS-NoSoft (60 s)			BandHS (300 s)			BandHS-NoSoft (300 s)		
		#score.	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*
PMS_2018	153	**0.6570**	**93**	15.210	0.6478	70	16.263	**0.7164**	**103**	69.880	0.6894	70	60.443
PMS_2019	299	**0.6736**	**198**	15.021	0.6536	144	13.211	**0.7094**	**203**	60.182	0.6833	145	48.300
PMS_2020	262	**0.6935**	**169**	13.637	0.6743	123	12.610	**0.7157**	**177**	59.634	0.6969	124	41.018
PMS_2021	155	**0.6302**	**94**	14.181	0.6150	76	10.933	**0.6530**	**103**	61.533	0.6343	71	35.334
PMS_2022	179	**0.6794**	**114**	14.005	0.6619	76	13.954	**0.7126**	**117**	59.725	0.6920	75	51.991
PMS_2023	179	**0.5698**	**104**	20.053	0.5544	79	19.488	**0.6174**	**112**	95.209	0.6064	85	80.957
WPMS_2018	172	**0.6970**	**117**	18.009	0.6796	64	13.380	**0.7316**	**119**	105.060	0.7101	59	83.117
WPMS_2019	297	**0.6543**	**208**	21.312	0.6243	101	13.418	**0.7182**	**218**	102.775	0.6990	116	92.831
WPMS_2020	253	**0.6358**	**170**	19.733	0.6206	93	16.079	**0.7105**	**184**	102.849	0.6898	99	70.075
WPMS_2021	151	**0.5595**	**89**	27.489	0.5422	39	16.224	**0.6552**	**94**	110.988	0.6341	52	106.915
WPMS_2022	197	**0.6499**	**122**	23.467	0.6175	50	17.137	**0.7273**	**129**	106.819	0.7024	55	102.282
WPMS_2023	160	**0.5520**	**83**	24.241	0.5224	55	17.006	**0.6424**	**100**	115.377	0.6239	51	115.110

Table 11
Comparison of BandHS and BandHS-NoBinary under two time limits of 60 s and 300 s.

Benchmark	*#inst.*	BandHS (60 s)			BandHS-NoBinary (60 s)			BandHS (300 s)			BandHS-NoBinary (300 s)		
		#score.	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*
PMS_2018	153	**0.6570**	98	17.474	0.6543	**101**	20.637	**0.7164**	**109**	70.545	0.6917	99	66.989
PMS_2019	299	**0.6736**	**205**	15.796	0.6571	193	14.289	**0.7094**	**206**	60.758	0.6935	203	56.262
PMS_2020	262	**0.6935**	**177**	14.108	0.6833	169	14.692	**0.7157**	**179**	60.247	0.7112	178	59.223
PMS_2021	155	**0.6302**	97	14.819	0.6161	97	**14.612**	**0.6530**	**104**	59.285	0.6488	97	41.041
PMS_2022	179	**0.6794**	112	14.157	0.6736	**119**	15.611	**0.7126**	118	60.382	0.7026	**120**	54.815
PMS_2023	179	**0.5698**	**113**	19.349	0.5542	108	18.536	**0.6174**	**119**	92.254	0.6079	117	84.296
WPMS_2018	172	**0.6970**	**108**	14.956	0.6929	78	15.767	**0.7316**	**107**	113.528	0.7246	75	84.349
WPMS_2019	297	**0.6543**	**173**	18.991	0.6465	165	20.427	**0.7182**	**198**	97.311	0.7158	165	82.093
WPMS_2020	253	0.6358	**156**	19.075	**0.6396**	129	20.039	0.7105	**167**	98.621	**0.7125**	139	87.009
WPMS_2021	151	**0.5595**	**78**	28.110	0.5567	70	27.173	**0.6552**	**87**	114.055	0.6481	79	118.332
WPMS_2022	197	**0.6499**	**108**	23.540	0.6353	90	22.118	**0.7273**	**119**	99.626	0.7114	98	119.756
WPMS_2023	160	**0.5520**	**83**	22.732	0.5457	79	20.299	**0.6424**	**97**	123.301	0.6334	74	110.052

4.5. Ablation study

Finally, we perform ablation studies to analyze the efficacy of the components and strategies in BandHS. The components include the *hard* and *soft* MAB models and the HyDeci initialization method. The strategies mainly include the sampling strategy for selecting the candidate arms in the *soft* MAB model and the delayed reward method for updating the estimated values. The results consistently demonstrate the effectiveness and necessity of these elements in enhancing the overall performance of BandHS.

4.5.1. Ablation study on components

To assess the impact of the *hard* MAB model, we conducted a comparison between BandHS and BandMaxSAT [26], the latter exclusively utilizing the *soft* MAB model and excluding the *hard* MAB model. The results, as depicted in Table 9, reveal that BandHS outperforms BandMaxSAT across most benchmarks, measured by both the *#win.* and *#score* metrics. This indicates that the incorporation of the *hard* MAB model contributes to the further enhancement of the BandMaxSAT algorithm.

Table 12
Comparison of BandHS and BandHS-NoSample under two time limits of 60 s and 300 s.

Benchmark	*#inst.*	BandHS (60 s)			BandHS-NoSample (60 s)			BandHS (300 s)			BandHS-NoSample (300 s)		
		#score.	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*
PMS_2018	153	0.6570	**94**	17.524	**0.6596**	78	17.262	**0.7164**	**104**	65.924	0.6995	86	65.215
PMS_2019	299	**0.6736**	**196**	15.329	0.6654	150	14.468	**0.7094**	**202**	56.344	0.6986	162	67.960
PMS_2020	262	**0.6935**	**167**	14.589	0.6802	127	13.634	**0.7157**	**172**	58.190	0.7080	139	61.187
PMS_2021	155	**0.6302**	**103**	15.376	0.6139	75	13.850	**0.6530**	**109**	57.032	0.6430	81	51.359
PMS_2022	179	**0.6794**	**114**	13.894	0.6773	91	15.725	**0.7126**	**115**	54.872	0.7054	104	67.123
PMS_2023	179	**0.5698**	**113**	18.507	0.5589	79	19.077	**0.6174**	**116**	88.421	0.6054	87	87.403
WPMS_2018	172	**0.6970**	**98**	14.525	0.6832	85	15.828	**0.7316**	**99**	108.893	0.7226	84	95.362
WPMS_2019	297	**0.6543**	**177**	20.740	0.6351	144	18.584	**0.7182**	**194**	95.053	0.7109	162	93.473
WPMS_2020	253	**0.6358**	**148**	19.047	0.6297	120	19.477	**0.7105**	**154**	98.726	0.7088	140	107.335
WPMS_2021	151	**0.5595**	**74**	26.924	0.5517	56	26.475	**0.6552**	**84**	109.244	0.6475	68	112.652
WPMS_2022	197	**0.6499**	**98**	22.763	0.6281	83	20.889	**0.7273**	**113**	103.087	0.7210	89	104.461
WPMS_2023	160	**0.5520**	**83**	24.145	0.5191	57	21.819	**0.6424**	**92**	126.125	0.6330	60	131.290

Table 13
Comparison of BandHS and BandHS-NoDelay under two time limits of 60 s and 300 s.

Benchmark	*#inst.*	BandHS (60 s)			BandHS-NoDelay (60 s)			BandHS (300 s)			BandHS-NoDelay (300 s)		
		#score.	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*	*#score.*	*#win.*	*time*
PMS_2018	153	0.6570	**89**	15.247	**0.6732**	78	19.488	**0.7164**	**96**	65.767	0.7148	80	69.071
PMS_2019	299	**0.6736**	**186**	14.561	0.6714	151	14.714	**0.7094**	**196**	58.108	0.6973	160	50.501
PMS_2020	262	**0.6935**	**170**	14.101	0.6799	125	13.359	**0.7157**	**175**	56.842	0.7005	132	52.853
PMS_2021	155	**0.6302**	**90**	14.844	0.6131	80	12.521	**0.6530**	**101**	61.118	0.6371	78	34.316
PMS_2022	179	**0.6794**	**103**	14.001	0.6786	91	13.240	**0.7126**	**109**	61.297	0.7079	89	45.050
PMS_2023	179	**0.5698**	**97**	19.444	0.5603	89	19.635	**0.6174**	**107**	89.183	0.6127	94	87.609
WPMS_2018	172	**0.6970**	**115**	16.849	0.6870	67	12.281	**0.7316**	**112**	106.159	0.7192	65	80.441
WPMS_2019	297	**0.6543**	**189**	20.876	0.6343	125	17.984	**0.7182**	**212**	106.699	0.7065	127	75.013
WPMS_2020	253	**0.6358**	**151**	18.813	0.6297	112	18.108	**0.7105**	**172**	103.493	0.7046	113	92.495
WPMS_2021	151	**0.5595**	**82**	25.923	0.5503	48	23.308	**0.6552**	**90**	119.626	0.6338	59	111.582
WPMS_2022	197	**0.6499**	**112**	22.667	0.6285	60	19.188	**0.7273**	**126**	106.351	0.7048	62	105.887
WPMS_2023	160	**0.5520**	**79**	24.368	0.5222	59	20.283	**0.6424**	**97**	121.702	0.6347	58	120.979

To assess the impact of the *soft* MAB model, we conducted a comparison between BandHS and its variant BandHS-NoSoft. In BandHS-NoSoft, we set the parameter $ArmNum$ to 1 and adjusted other parameters to effectively remove the *soft* MAB model from BandHS. The comparison results, presented in Table 10, clearly demonstrate that BandHS significantly outperforms BandHS-NoSoft. This provides compelling evidence for the valuable contribution of the *soft* MAB model in assisting the local search algorithm to discover better feasible solutions.

To assess the impact of HyDeci, we compared BandHS against a variant, BandHS-NoBinary, which excludes the prioritization of binary clauses in HyDeci (i.e., lines 8-15 in Algorithm 4 are removed). The outcomes, detailed in Table 11, highlight that BandHS outperforms BandHS-NoBinary across the majority of PMS and WPMS benchmarks. This underscores the effectiveness of the proposed HyDeci algorithm, which prioritizes both unit and binary clauses, in enhancing the BandHS algorithm.

4.5.2. Ablation study on strategies

To evaluate the impact of the sampling strategy for selecting candidate arms in the *soft* MAB model, we compare BandHS with its variant BandHS-NoSample, which selects the arm to be pulled in the *soft* MAB model by traversing all the available arms. The results, shown in Table 12, demonstrate that the sampling strategy used in the *soft* MAB model is effective and necessary.

We further compare BandHS with its variant BandHS-NoDelay, which sets the parameter d to 1, abandons the parameter γ, and tunes the others to evaluate the effect of the delayed reward method. The results, shown in Table 13, indicate that the delayed reward method fits well with the problems, and the method can better help BandHS evaluate the quality of the arms.

5. Conclusion

This paper introduces BandHS, a novel local search algorithm for (W)PMS. The algorithm applies multi-armed bandit (MAB) models on hard and soft clauses to guide the search directions when escaping from local optima and find better quality solutions. The *hard* MAB model is combined with all the literals in hard clauses to help BandHS select suitable literals to satisfy the hard clauses. The *soft* MAB model is combined with all the soft clauses to help BandHS select to satisfy appropriate soft clauses. We also propose an effective initialization method, HyDeci, that prioritizes unit and binary clauses when constructing the initial solution.

The extensive experiments reported demonstrate the superior performance and generalization capability of the proposed methods, notably improving the state-of-the-art (W)PMS solvers SATLike3.0, NuWLS, and NuWLS-c-2023. The analyses conducted on the category of instances where our MAB method could perform better reveal that our MAB method prefers instances containing more soft clauses that are difficult to satisfy. In such cases, the widely-used random strategy is more likely to select a poor search direction. Conversely, our MAB method could suggest a more appropriate search direction in these scenarios. Additionally, our ablation studies confirm the efficacy, enhancing the solution quality, of each component or strategy of BandHS, including the *hard* MAB model, *soft* MAB model, HyDeci initialization, sampling strategy in the *soft* MAB, and the delayed reward calculation.

In the future, we plan to further explore the potential of MAB methods in solving MaxSAT and SAT problems.

CRediT authorship contribution statement

Jiongzhi Zheng: Conceptualization, Data curation, Methodology, Software, Writing – original draft, Writing – review & editing. **Kun He:** Conceptualization, Funding acquisition, Methodology, Supervision, Writing – review & editing. **Jianrong Zhou:** Conceptualization, Methodology, Writing – review & editing. **Yan Jin:** Writing – review & editing. **Chu-Min Li:** Supervision, Writing – review & editing. **Felip Manyà:** Funding acquisition, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Kun He reports financial support was provided by National Natural Science Foundation of China (U22B2017). Kun He reports financial support was provided by Microsoft Research Asia (100338928).

Acknowledgement

This work is supported by National Natural Science Foundation of China (U22B2017), Microsoft Research Asia (100338928), and grants PID2019-111544GB-C21, PID2022-139835NB-C21 and TED2021-129319B-I00 funded by MCIN/AEI/10.13039/501100011033.

Data availability

Data will be made available on request.

References

[1] Chu Min Li, Felip Manyà, MaxSAT, hard and soft constraints, in: Handbook of Satisfiability, second edition, IOS Press, 2021, pp. 903–927.
[2] Blai Bonet, Guillem Francès, Hector Geffner, Learning features and abstract actions for computing generalized plans, in: Proceedings of the 33rd AAAI Conference on Artificial Intelligence, 2019, pp. 2703–2710.
[3] Carlos Ansótegui, Felip Manyà, Jesus Ojeda, Josep M. Salvia, Eduard Torres, Incomplete MaxSAT approaches for combinatorial testing, J. Heuristics 28 (4) (2022) 377–431.
[4] Lorenzo Ciampiconi, Bishwamittra Ghosh, Jonathan Scarlett, Kuldeep S. Meel, A MaxSAT-based framework for group testing, in: Proceedings of the 34th AAAI Conference on Artificial Intelligence, 2020, pp. 10144–10152.
[5] Emir Demirovic, Nysret Musliu, MaxSAT-based large neighborhood search for high school timetabling, Comput. Oper. Res. 78 (2017) 172–180.
[6] Chu Min Li, Felip Manyà, Jordi Planes, New inference rules for Max-SAT, J. Artif. Intell. Res. 30 (2007) 321–359.
[7] Federico Heras, Javier Larrosa, Albert Oliveras, MiniMaxSat: a new weighted Max-SAT solver, in: Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing, vol. 4501, 2007, pp. 41–55.
[8] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, Jordi Planes, Resolution-based lower bounds in MaxSAT, Constraints 15 (4) (2010) 456–484.
[9] Mohamed Sami Cherif, Djamal Habet, André Abramé, Understanding the power of Max-SAT resolution through UP-resilience, Artif. Intell. 289 (2020) 103397.
[10] Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, Kun He, Combining clause learning and branch and bound for MaxSAT, in: Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming, vol. 210, 2021, pp. 38:1–38:18.
[11] Fahiem Bacchus, Matti Järvisalo, Martins Ruben, Maximum satisfiability, in: Handbook of Satisfiability, second edition, IOS Press, 2021, pp. 929–991.
[12] Zhaohui Fu, Sharad Malik, On solving the partial MAX-SAT problem, in: Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing, vol. 4121, 2006, pp. 252–265.
[13] Nina Narodytska, Fahiem Bacchus, Maximum satisfiability using core-guided MaxSAT resolution, in: The Twenty-Eighth AAAI Conference on Artificial Intelligence, 2014, pp. 2717–2723.
[14] Carlos Ansótegui, Joel Gabàs, WPM3: an (in)complete algorithm for weighted partial MaxSAT, Artif. Intell. 250 (2017) 37–57.
[15] Jeremias Berg, Emir Demirovic, Peter J. Stuckey, Core-boosted linear search for incomplete MaxSAT, in: Proceedings of the 16th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, vol. 11494, 2019, pp. 39–56.
[16] Alexander Nadel, Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization, in: Proceedings of the 2019 Formal Methods in Computer Aided Design, 2019, pp. 193–202.
[17] Chu Min Li, Felip Manyà, Jordi Planes, Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers, in: Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, 2005, pp. 403–414.
[18] Chu Min Li, Felip Manyà, Jordi Planes, Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT, in: Proceedings of the 21st National Conference on Artificial Intelligence, 2006, pp. 86–91.
[19] Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, Kun He, Boosting branch-and-bound MaxSAT solvers with clause learning, AI Commun. 35 (2) (2022) 131–151.
[20] Bart Selman, Henry A. Kautz, Bram Cohen, Local search strategies for satisfiability testing, in: Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, vol. 26, 1993, pp. 521–531.
[21] Paul Morris, The breakout method for escaping from local minima, in: Proceedings of the 11th National Conference on Artificial Intelligence, 1993, pp. 40–45.
[22] Byungki Cha, Kazuo Iwama, Yahiko Kambayashi, Shuichi Miyazaki, Local search algorithms for partial MAXSAT, in: Proceedings of the 14th National Conference on Artificial Intelligence, 1997, pp. 263–268.
[23] Shaowei Cai, Chuan Luo, Jinkun Lin, Kaile Su, New local search methods for partial MaxSAT, Artif. Intell. 240 (2016) 1–18.

[24] Chuan Luo, Shaowei Cai, Kaile Su, Wenxuan Huang, CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability, Artif. Intell. 243 (2017) 26–44.
[25] Shaowei Cai, Zhendong Lei, Old techniques in new ways: clause weighting, unit propagation and hybridization for maximum satisfiability, Artif. Intell. 287 (2020) 103354.
[26] Jiongzhi Zheng, Kun He, Jianrong Zhou, Yan Jin, Chu-Min Li, Felip Manyà, BandMaxSAT: a local search MaxSAT solver with multi-armed bandit, in: Proceedings of the 31st International Joint Conference on Artificial Intelligence, 2022, pp. 1901–1907.
[27] Shaowei Cai, Xindi Zhang, Mathias Fleury, Armin Biere, Better decision heuristics in CDCL through local search and target phases, J. Artif. Intell. Res. 74 (2022) 1515–1563.
[28] Yi Chu, Shaowei Cai, Chuan Luo, NuWLS-c-2023: solver description, MaxSAT Evaluation 2023 (2023) 23–24.
[29] Shaowei Cai, Chuan Luo, John Thornton, Kaile Su, Tailoring local search for partial MaxSAT, in: Proceedings of the 28th AAAI Conference on Artificial Intelligence, 2014, pp. 2623–2629.
[30] Zhendong Lei, Shaowei Cai, Solving (weighted) partial MaxSAT by dynamic local search for SAT, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence, 2018, pp. 1346–1352.
[31] Yi Chu, Shaowei Cai, Chuan Luo, NuWLS: improving local search for (weighted) partial MaxSAT by new weighting techniques, in: Proceedings of the 37th AAAI Conference on Artificial Intelligence, 2023, pp. 3915–3923.
[32] Aleksandrs Slivkins, Introduction to multi-armed bandits, Found. Trends Mach. Learn. 12 (1–2) (2019) 1–286.
[33] Tor Lattimore, Csaba Szepesvári, Bandit Algorithms, Cambridge University Press, 2020.
[34] Ming-Te Chao, John V. Franco, Probabilistic analysis of a generalization of the unit-clause literal selection heuristics for the k satisfiability problem, Inf. Sci. 51 (3) (1990) 289–314.
[35] Chvátal Vasek, Bruce A. Reed, Mick gets some (the odds are on his side), in: Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, 1992, pp. 620–627.
[36] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, Sharad Malik, Chaff: engineering an efficient SAT solver, in: Proceedings of the 38th Design Automation Conference, 2001, pp. 530–535.
[37] Shaowei Cai, Chuan Luo, Haochen Zhang, From decimation to local search and back: a new approach to MaxSAT, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence, 2017, pp. 571–577.
[38] Jack Goffinet, Raghuram Ramanujan, Monte-Carlo tree search for the maximum satisfiability problem, in: Proceedings of the 32nd International Conference on Principles and Practice of Constraint Programming, vol. 9892, 2016, pp. 251–267.
[39] Mourad Lassouaoui, Dalila Boughaci, Belaid Benhamou, A multilevel synergy Thompson sampling hyper-heuristic for solving Max-SAT, Intelligent Decision Technologies 13 (2) (2019) 193–210.
[40] Mohamed Sami Cherif, Djamal Habet, Cyril Terrioux, Kissat MAB: combining VSIDS and CHB through multi-armed bandit, SAT Competition 2021 (2021) 15.
[41] Jordi Coll, Shuolin Li, Felip Manyà, Chu-Min Li, Djamal Habet, Mohamed Sami Cherif, Kun He, WMaxCDCL in MaxSAT evaluation 2023, MaxSAT Evaluation 2023 (2023) 23–24.
[42] Jiongzhi Zheng, Kun He, Zhuo Chen, Jianrong Chou, Chu-Min Li, Decision tree based hybrid walking strategies, MaxSAT Evluation 2022 (2022) 24–25.
[43] Jiongzhi Zheng, Kun He, Mingming Jin, Zhuo Chen, Jinghui Xu, Combining BandMaxSAT and FPS with NuWLS-c, MaxSAT Evluation 2023 (2023) 25–26.
[44] Yi Chu, Shaowei Cai, Zhendong Lei, Xiang He, NuWLS-c: solver description, MaxSAT Evaluation 2022 (2022) 28.
[45] Shaowei Cai, Balance between complexity and quality: local search for minimum vertex cover in massive graphs, in: Proceedings of the 24th International Joint Conference on Artificial Intelligence, 2015, pp. 747–753.
[46] Yi-Qi Hu, Yang Yu, Jun-Da Liao, Cascaded algorithm-selection and hyper-parameter optimization with extreme-region upper confidence bound bandit, in: Proceedings of the 28th International Joint Conference on Artificial Intelligence, 2019, pp. 2528–2534.
[47] Mingdong Ou, Nan Li, Cheng Yang, Shenghuo Zhu, Rong Jin, Semi-parametric sampling for stochastic bandits with many arms, in: Proceedings of the 33rd AAAI Conference on Artificial Intelligence, 2019, pp. 7933–7940.
[48] Sakshi Arya, Yuhong Yang, Randomized allocation with nonparametric estimation for contextual multi-armed bandits with delayed rewards, Stat. Probab. Lett. 164 (2020) 108818.
[49] Zhendong Lei, Shaowei Cai, Fei Geng, Dongxu Wang, Yongrong Peng, Dongdong Wan, Yiping Deng, Pinyan Lu, SATLike-c: solver description, MaxSAT Evaluation 2021 (2021) 19–20.
[50] Jeremias Berg, Loandra in the 2022 (and 2023) MaxSAT evaluation, MaxSAT Evaluation 2023 (2023) 21–22.
[51] Alexander Nadel, Tt-open-wbo-inc-23: an anytime MaxSAT solver entering MSE'23, MaxSAT Evaluation 2023 (2023) 29.
[52] Jiongzhi Zheng, Kun He, Jianrong Zhou, Farsighted probabilistic sampling: a general strategy for boosting local search MaxSAT solvers, in: Proceedings of the 37th AAAI Conference on Artificial Intelligence, 2023, pp. 4132–4139.
[53] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, Frank Hutter, SMAC3: a versatile Bayesian optimization package for hyperparameter optimization, J. Mach. Learn. Res. 23 (54) (2022) 1–9.
[54] Alexander Nadel, Polarity and variable selection heuristics for SAT-based anytime MaxSAT, J. Satisf. Boolean Model. Comput. 12 (1) (2020) 17–22.
[55] Gilles Audemard, Laurent Simon, Predicting learnt clauses quality in modern SAT solvers, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence, 2009, pp. 399–404.
[56] Armin Biere, Katalin Fazekas, Mathias Fleury, Maximilian Heisinger, CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT competition 2020, SAT Competition 2020 (2020) 51–53.
[57] Alexander Nadel, Introducing Intel(R) SAT solver, in: Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing, vol. 236, 2022, pp. 8:1–8:23.