Artificial Intelligence 348 (2025) 104405

Contents lists available at ScienceDirect

Artificial Intelligence

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

Local-MIP: Efficient local search for mixed integer programming ☆

Peng Lin [a,b], Shaowei Cai [a,b,*], Mengchuan Zou [a], Jinkun Lin [c]

[a] Key Laboratory of System Software (Chinese Academy of Sciences), Institute of Software, Chinese Academy of Sciences, Beijing, China
[b] School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China
[c] SeedMath Technology Limited, Beijing, China

ARTICLE INFO

Dataset link: Detailed results and source code of Local-MIP (Original data)

Keywords:
Mixed integer programming
Local search
Operator
Scoring function

ABSTRACT

Mixed Integer Programming (MIP) is a fundamental model in operations research with broad industrial applications. Local search is a powerful methodology for solving complex optimization problems; however, the development of local search algorithms for MIP still needs exploration. In this work, we propose *Local-MIP*, an efficient local search algorithm tailored for MIP that integrates novel operators and employs a two-mode architecture to adaptively apply operators based on the current solution's feasibility. For the feasible mode, we propose the lift move operator and a corresponding lift process to improve the objective value while maintaining feasibility. For the infeasible mode, we propose the breakthrough move and mixed tight move operators to respectively optimize the objective function and satisfy constraints. To apply operators intelligently, we develop a dynamic weighting scheme that balances the priorities of the objective function and constraints. Furthermore, we propose a two-level scoring function structure that hierarchically selects operations, guiding the search toward high-quality feasible solutions. Experiments are conducted on public benchmarks to compare *Local-MIP* with state-of-the-art MIP solvers in finding high-quality solutions. The results show that *Local-MIP* significantly outperforms *CPLEX*, *HiGHS*, *SCIP*, and *Feasibility Jump* while remaining competitive with the commercial solver *Gurobi* on challenging problems within short time limits. Moreover, *Local-MIP* establishes 10 new records on MIPLIB open instances.

1. Introduction

Mixed Integer Programming (MIP) is a fundamental model in operations research [1] that optimizes a linear objective function subject to linear constraints, where a subset of the variables is restricted to integer values. Owing to its expressive power, MIP serves as a cornerstone for modeling a diverse range of problems [2,3]. These span from classic combinatorial optimization problems, such as the maximum satisfiability (MaxSAT) [4], the knapsack problem [5], the traveling salesman problem (TSP) [6], the job shop scheduling problem (JSP) [7], and various graph problems [8], to large-scale industrial applications, including production planning [9], crew scheduling [10], and resource allocation [11].

Solving MIPs is computationally challenging, as they are NP-hard [12,13]. The corresponding solving methods can be classified into complete and incomplete [14]. Complete methods seek to compute the exact optimal solution and certify its optimality, whereas

☆ This paper is an invited revision of a paper which first appeared at the 2024 International Conference on Principles and Practice of Constraint Programming (CP-24).

* Corresponding author at: Key Laboratory of System Software (Chinese Academy of Sciences), Institute of Software, Chinese Academy of Sciences, Beijing, China.
E-mail addresses: linpeng@ios.ac.cn (P. Lin), caisw@ios.ac.cn (S. Cai), zoumc@ios.ac.cn (M. Zou), linjk@seedmath.com (J. Lin).

https://doi.org/10.1016/j.artint.2025.104405
Received 3 December 2024; Received in revised form 7 August 2025; Accepted 18 August 2025

Available online 27 August 2025

incomplete methods strive to obtain high-quality solutions within a reasonable time frame. The large scale of many real-world problems, however, renders them challenging for complete methods. Incomplete methods are therefore important for solving MIPs, given that high-quality solutions are often sufficient for practical purposes [15].

The majority of contemporary MIP solvers are built upon the branch-and-bound framework [16,17], a widely used complete method. This approach recursively partitions the feasible region and prunes nodes based on objective function bounds. To further enhance performance, various techniques have been developed and incorporated into this framework for solving MIPs, such as cutting plane [18], domain propagation [19] and primal heuristics [20] (e.g., feasibility pump [21], shift-and-propagate [22], structure-oriented fix-and-propagate [23], and fix-propagate-repair [24]). The hybrid approach, which integrates the techniques mentioned above, is the infrastructure of state-of-the-art solvers [25], including commercial solvers *Gurobi* [26] and *CPLEX* [27], and academic solvers *SCIP* [28] and *HiGHS* [29].

Local search is a powerful incomplete method for addressing challenging problems across computer science and operations research [30], distinguished by its ability to find high-quality solutions efficiently. It has demonstrated significant effectiveness on combinatorial problems such as SAT [31–34] and MaxSAT [35–38]. For MIPs, local search algorithms have been proposed for special cases, such as pseudo-Boolean optimization (PBO; i.e., $0-1$ integer programming) [39–41], over-constrained integer programming [42], and satisfiability of integer programming [43]. However, the development of local search for general MIPs still needs to be explored, particularly within the domain of LP-free algorithms.[1] To our knowledge, efficient and open-source local search algorithms are scarce for general MIPs. The *Feasibility Jump* [44] is the most closely related work to ours, which features a local search algorithm that exclusively targets constraint satisfaction and won 1st place in the MIP 2022 Competition.[2] However, its disregard for the objective function limits its capacity to produce high-quality solutions, thereby not addressing MIPs' original optimization purpose. Building on the *Feasibility Jump*, *ViolationLS* [45] is implemented in the CP-SAT Solver [46] to handle both linear constraints and the objective, but only for pure integer variables.

This work aims to develop an efficient local search algorithm for general MIPs. Achieving this goal presents two primary challenges:

1. Enhancing adaptability: General MIPs consist of general constraints and variables; thus, designing local search algorithms for general MIPs is more challenging than for specific combinatorial problems, as less problem-specific information can be leveraged. Therefore, enhancing the algorithm's adaptability is crucial for achieving high performance.
2. Balance of optimization and satisfaction: Unlike decision problems such as SAT, MIP requires optimizing the objective function and satisfying all constraints. These two factors often conflict, making it crucial to strike an effective balance between them.

To address these challenges, we propose novel search strategies, operators, and scoring functions tailored to the general MIP formulation. An important contribution of our work is the design of three new local search operators, each crafted to serve different search purposes and facilitate effective transitions between solutions. These operators form the foundation of our local search mechanism and are coupled with dedicated scoring functions that evaluate candidate neighbor solutions. The scoring functions are specifically designed to guide the search by balancing two goals: improving the solution's objective value and reducing constraint violations.

Beyond these basic components, we further enhance adaptability by introducing distinct search modes and enabling the algorithm to switch between them dynamically. We design two modes (feasible and infeasible modes) and associate each with specific operators and scoring functions. This design allows the algorithm to apply tailored strategies depending on the search context, improving its capability for finding both feasible and high-quality solutions.

Building upon these components, we implement our algorithm, named *Local-MIP*, and evaluate its performance in finding high-quality solutions on general MIP benchmarks with state-of-the-art solvers.

1.1. Contributions

In this work, we present a novel and efficient local search algorithm for general MIPs, named *Local-MIP*, with its source code made publicly available on GitHub.[3] To our knowledge, this represents the first open-source local search algorithm for general MIPs that demonstrates performance competitive with powerful commercial solvers in quickly finding high-quality solutions.

The core of *Local-MIP* is a two-mode search framework that adapts based on the feasibility of the current solution, comprising a feasible mode and an infeasible mode. This framework is powered by three new operators, each tailored for the general MIP formulation with distinct functionalities suited to different search modes. To support the use of these operators, we introduce a novel dynamic weighting scheme and a hierarchical two-level scoring structure. The main contributions and innovations of this paper are summarized as follows.

Firstly, for the feasible mode, we propose the lift move operator, which aims to improve the objective value by modifying a single variable while strictly maintaining feasibility. To enable maintaining feasibility, we introduce the concept of the local feasible domain: the precise range a variable can take without violating any constraints. Based on the local feasible domain, we develop the lift process, which iteratively applies lift moves to consecutively improve the solution until a local optimum is reached. For computational

[1] By "LP-free" we mean the solver does not rely on solving linear programming relaxation in the algorithm.
[2] https://www.mixedinteger.org/2022/competition/.
[3] https://github.com/shaowei-cai-group/Local-MIP.

efficiency, we design an incremental update approach to compute these domains, thereby boosting the efficiency of the entire lift process.

Secondly, for the infeasible mode, we propose two new operators to navigate the search space: the breakthrough move, designed to surpass the best-known objective value, and the mixed tight move, which focuses on satisfying and tightening constraints.

To balance optimization and feasibility within the infeasible mode, we devise a dynamic weighting scheme that adjusts the priority between the objective and individual constraints. This is complemented by a two-level scoring structure that hierarchically evaluates candidate moves. The first level, the progress score, measures the direct improvement of an operation. The second level, the bonus score, provides a finer-grained evaluation, awarding a breakthrough bonus for improving the global best objective and a robustness bonus for maintaining stable constraint satisfaction.

Our algorithm, which integrates these components, is named *Local-MIP*. We conduct extensive experiments on seven public benchmarks to evaluate its effectiveness in finding high-quality solutions. *Local-MIP* is compared with state-of-the-art solvers, including the commercial solvers *Gurobi* [26] and *CPLEX* [27], the academic solvers *SCIP* [28] and *HiGHS* [29], and the local search algorithm *Feasibility Jump* [44]. The results demonstrate the good performance of *Local-MIP* on challenging instances under short time limits. It significantly outperforms *CPLEX*, *HiGHS*, *SCIP*, and *Feasibility Jump*, and its performance is competitive with *Gurobi*, the leading commercial solver. This represents a new advancement for local search methods in the MIP field. Notably, *Local-MIP* establishes 10 new best-known solutions for open instances in the MIPLIB benchmark. Further ablation studies and sensitivity analyses are also performed to validate the effectiveness of our proposed strategies and assess the algorithm's stability.

This article is an extended version of a conference paper presented at CP 2024 [47], which received the Best Paper Award of CP 2024. Additional information and contributions in this article include the local feasible domain of local search, the lift move operator, the lift process, and the partitioning of two modes adapted to the feasibility of the current solution, all of which can be dated back to our previous unpublished work [48]. We also perform a more comprehensive empirical analysis of the proposed methods.

1.2. Paper organization

The remainder of the paper is organized as follows. Section 2 introduces the basic concepts of the MIP formulation and the local search algorithm. Section 3 proposes the local feasible domain and the lift move in the feasible mode. In Sections 4, we introduce the breakthrough move and the mixed tight move operators in the infeasible mode. Section 5 presents the weighting scheme and the two-level scoring function structure. We detail the *Local-MIP* algorithm in Section 6 by integrating these components. The experimental results on public benchmark instances are reported in Section 7, followed by the conclusions and future work in Section 8.

2. Preliminaries and notation

2.1. Formulation of MIP

Definition 1. Mixed Integer Programming (MIP): Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, vectors $\boldsymbol{b} \in \mathbb{R}^m, \boldsymbol{c}, \boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^n$, and a subset $\mathcal{I} \subseteq \mathcal{N} = \{1, ..., n\}$. Let $\boldsymbol{x} = \{x_1, x_2, ..., x_n\}$ be a set of variables. The mixed integer programming is to solve

$$min\{\boldsymbol{c}^\top \boldsymbol{x} |\ \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u},\ \boldsymbol{x} \in \mathbb{R}^n,\ x_j \in \mathbb{Z} \text{ for all } j \in \mathcal{I}\} \tag{1}$$

In the above definition, we call $\boldsymbol{c}^\top \boldsymbol{x}$ the objective function, $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ the general linear constraints, $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$ the global bounds, and $x_j \in \mathbb{Z}\ for\ all\ j \in \mathcal{I}$ the integrality constraints. A general linear constraint $\boldsymbol{A_i}\boldsymbol{x} \leq b_i$ is denoted as con_i, and contains x_j if $A_{ij} \neq 0$. MIP aims to minimize the objective function while satisfying all the constraints. A maximization problem and other types of linear constraints can be easily converted into this formulation.

A solution $\boldsymbol{s}$ of an MIP instance is a vector of values assigned for each variable, where s_j denotes the value assigned for x_j. A solution is a **feasible solution** if and only if it satisfies all constraints, including general linear constraints, global bounds, and integrality constraints. For feasible solutions, the lower objective value indicates higher quality.

To facilitate clarity, we establish certain symbols here. Let $\boldsymbol{s}^*$ denote the best-found solution in the local search process, and $\boldsymbol{s}^{\boldsymbol{cur}}$ denote the current solution in each step of the local search. $\boldsymbol{s}^*$ is initialized to an empty set and updated whenever a new best feasible solution is found. The objective value of a solution $\boldsymbol{s}$ is denoted as $obj(\boldsymbol{s})$, i.e., $obj(\boldsymbol{s}) = \boldsymbol{c}^\top \boldsymbol{s}$.

2.2. Local search algorithm

When solving combinatorial optimization problems, a local search usually starts with an initial solution. It iteratively modifies the current solution by altering the value of a variable to find feasible solutions with high-quality objective values.

In local search, an **operator** defines how to modify variables to generate candidate solutions. When an operator is instantiated by a specified variable to operate, an **operation** is obtained. For example, for Boolean variables, the standard operator is $flip$, which changes the value of a variable to its opposite, and $flip(x_1)$ is the operation that flips the specified variable x_1. For the current solution, performing an operation generates a new candidate solution.

During each step of the local search process, the scoring functions evaluate different candidate operations and select one for execution to update the current solution. Given an operation op, a scoring function $score(op)$ measures how good op is. An operation op is considered **positive** if $score(op) > 0$, indicating that performing op is deemed beneficial.

3. Lift move in the feasible mode

The search enters the feasible mode once the current solution is feasible. In this mode, the primary objective is to improve the objective value while strictly maintaining feasibility. To this end, we propose the lift move operator, which seeks to find the best possible value by modifying a single variable that maximizes objective improvement without violating any constraints. The execution of this operator relies on the concept of the local feasible domain, which we propose to precisely define the allowable range for such a move. Furthermore, we develop the lift process, an incremental procedure that quickly applies a sequence of lift move operations to iteratively improve the solution.

3.1. Local feasible domain

To maintain the feasibility of a feasible solution, we propose the local feasible domain, which defines a domain within which a variable can be modified without violating any constraints.

Definition 2. Given a feasible solution $\boldsymbol{s}$ and a variable x_j, **the local feasible domain** of x_j in $\boldsymbol{s}$, denoted as $\text{lfd}(x_j, \boldsymbol{s})$, is the domain that when x_j varies in this domain and all other variables remain unchanged, the satisfiability of all constraints is preserved.

To compute a variable's local feasible domain, we first consider the range that it can vary with respect to each constraint.

Definition 3. Given a feasible solution $\boldsymbol{s}$, a variable x_j, and a constraint con_i containing x_j, **the local feasible constraint domain** of x_j in $\boldsymbol{s}$ for con_i, denoted as $\text{lfcd}(x_j, \text{con}_i, \boldsymbol{s})$, is the domain within which x_j can vary while maintaining the satisfiability of con_i, assuming that other variables in $\boldsymbol{s}$ remain unchanged. Specifically, let $\Delta_{ij} = (b_i - \boldsymbol{A_i} \cdot \boldsymbol{s})/A_{ij}$, $\text{lfcd}(x_j, \text{con}_i, \boldsymbol{s})$ is derived according to the sign of A_{ij}:

$$\text{lfcd}(x_j, \text{con}_i, \boldsymbol{s}) = \begin{cases} \left[s_j + \Delta_{ij}, +\infty\right), & \Delta_{ij} < 0, \\ \left(-\infty, s_j + \Delta_{ij}\right], & \text{else}. \end{cases} \tag{2}$$

A variable's local feasible domain can be easily derived by computing the intersection of all associated local feasible constraint domains and the variable's global bounds.

Corollary 1. *Given a feasible solution $\boldsymbol{s}$ and a variable x_j. Let C_{x_j} denote the set of constraints that contain x_j. The local feasible domain of x_j can be computed as follows:*

$$lfd(x_j, \boldsymbol{s}) = \left(\bigcap_{con_i \in C_{x_j}} lfcd(x_j, con_i, \boldsymbol{s}) \right) \cap [l_j, u_j] \tag{3}$$

If x_j is an integer variable (i.e., $j \in \mathcal{I}$), then the local feasible domain of x_j is restricted in the integer domain:

$$lfd(x_j, \boldsymbol{s}) = \left(\bigcap_{con_i \in C_{x_j}} lfcd(x_j, con_i, \boldsymbol{s}) \right) \cap [l_j, u_j] \cap \mathbb{Z} \tag{4}$$

3.2. Lift move operator

Clearly, given a feasible solution, moving a variable within its local feasible domain does not violate feasibility as long as the other variables remain constant. Therefore, once the current solution $\boldsymbol{s^{cur}}$ is feasible, we can select an appropriate value for each variable in the objective function to update the current solution to improve the objective value. To achieve this, we propose the lift move operator based on the local feasible domain.

Definition 4. Given a variable x_j that appears in the objective function (that is, $c_j \neq 0$), and a feasible solution $\boldsymbol{s}$, the **lift move operator**, denoted as $lm(x_j, \boldsymbol{s})$, assigns a variable x_j to the upper or lower bound value of its local feasible domain, improving the objective value as much as possible while maintaining feasibility. Specifically, let c_j denote the coefficient of x_j in $\boldsymbol{c}^\top \boldsymbol{x}$, the lift move operation $lm(x_j, \boldsymbol{s})$ assigns x_j to a new value s_j^{new},

$$s_j^{new} = \begin{cases} \text{the upper bound of } \text{lfd}(x_j, \boldsymbol{s}), & \text{if } c_j < 0, \\ \text{the lower bound of } \text{lfd}(x_j, \boldsymbol{s}), & \text{else}. \end{cases} \tag{5}$$

Given the above definition, for a feasible solution, the lift move operator adaptively modifies a variable in the objective function without violating feasibility.

Fact 1. For a specific variable x_j, the new solution generated by the lift move operation has the best objective value among all feasible solutions obtained by modifying x_j.

3.3. Lift process

Based on the local feasible domain and the lift move operator, we propose the lift process to efficiently apply multiple lift move operations consecutively, optimizing the objective value of the current solution until the search process converges to a local optimum.

Within the feasible mode, a candidate lift move operation can be constructed for each variable involved in the objective function. This presents the algorithm with a set of potential operations, necessitating a mechanism to select the most promising one to execute. Given that all lift move operations maintain the solution's feasibility, we propose the lift score to measure the improvement in the objective function offered by each candidate operation.

Definition 5. Given an operation op, and the current solution $\boldsymbol{s}^{cur}$. Let $\boldsymbol{s}^{op}$ be the new candidate solution generated by performing op on $\boldsymbol{s}^{cur}$. The lift score of op measures the improved quality in the objective function, denoted as $score_{\text{lift}}(op)$,

$$score_{\text{lift}}(op) = obj(\boldsymbol{s}^{cur}) - obj(\boldsymbol{s}^{op}) \tag{6}$$

In each step of the feasible mode, we select the operation with the best $score_{\text{lift}}(op)$ to update the current solution.

A key property of the lift move operator is its locality: modifying a single variable only affects the local feasible domains of other variables that share a constraint with it. Consequently, the domains of unaffected variables remain valid and can be reused. This locality is what enables an efficient, incremental approach for updating these domains.

Let C_{x_j} denote the set of constraints containing variable x_j, $\mathcal{F}$ be the set of variables sharing a constraint with the most recently modified variable, and $\mathcal{O}$ be the set of variables appearing in the objective function. We distinguish between two update scenarios:

1. Full Computation: Upon entering the feasible mode from an infeasible mode, the local feasible domains for all variables in $\mathcal{O}$ are computed from scratch. The complexity of this operation is $\Theta(\sum_{x_j \in \mathcal{O}} |C_{x_j}|)$.
2. Incremental Update: Following a lift move operation (i.e., within the feasible mode), only the domains for the affected variables (those in the set $\mathcal{F} \cap \mathcal{O}$) need to be recomputed. The complexity of this update is merely $\Theta(\sum_{x_j \in \mathcal{F} \cap \mathcal{O}} |C_{x_j}|)$.

In typical models with sparse constraints, the complexity of the incremental update is substantially lower than that of a full computation. Algorithm 1 details the lift process in pseudo-code.

Algorithm 1: The Lift Process.

Input: the current solution $\boldsymbol{s}^{cur}$ which is feasible
$\mathcal{O} \leftarrow$ the set of variables appearing in the objective function
for $x_j \in \mathcal{O}$ **do**
  if x_j *is an integer variable* **then** Calculate $\text{lfd}(x_j, \boldsymbol{s}^{cur})$ by Eq. (4) ;
  else Calculate $\text{lfd}(x_j, \boldsymbol{s}^{cur})$ by Eq. (3) ;
  if $lfd(x_j, \boldsymbol{s}^{cur}) \neq \emptyset$ **then**
    Build the lift move operation by Eq. (5): $op_j \leftarrow lm(x_j, \boldsymbol{s}^{cur})$;
    Calculate the lift score $score_{\text{lift}}(op_j)$ by Eq. (6);
while $\exists$ *positive lift move operation such that* $score_{\text{lift}}(op_j) > 0$ **do**
  /* Select the operation with the greatest lift score */
  $\boldsymbol{op}_{best} \leftarrow \arg\max\{score_{\text{lift}}(op_j)\}$;
  /* Update the current solution */
  $\boldsymbol{s}^{cur} \leftarrow$ the solution generated by performing $\boldsymbol{op}_{best}$ on $\boldsymbol{s}^{cur}$;
  /* Incrementally update the affected variables' data */
  $\mathcal{F} \leftarrow$ the set of variables sharing a constraint with x_{best}
  for $x_j \in \mathcal{F} \cap \mathcal{O}$ **do**
    Update $\text{lfd}(x_j, \boldsymbol{s}^{cur})$, op_j, $score_{\text{lift}}(op_j)$ by Eq. (3)~(6).
return *the current solution* $\boldsymbol{s}^{cur}$ *which converges to local optimal*

4. Operators in the infeasible mode

The search enters the infeasible mode when the current solution violates one or more constraints. This occurs in two distinct situations:

1. Initial Search: Before any feasible solution has been found, the primary goal is to resolve constraint violations to the first feasible solution, which implies the feasibility of the given instance.

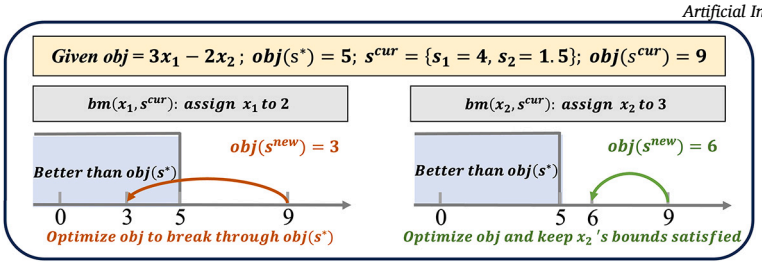

Fig. 1. A graphical explanation of the breakthrough move operator.

2. Optimization Search: After at least one feasible solution is found, the search may enter the infeasible mode (e.g., when escaping a local optimum from the feasible mode). Here, the focus shifts to both satisfying constraints and optimizing the objective function, since the search aims to explore promising regions that may lead to better feasible solutions than the best-found solution.

To navigate these situations, we propose two complementary local search operators for the infeasible mode: the breakthrough move and the mixed tight move. The former focuses on optimizing the objective, while the latter prioritizes satisfying constraints.

As a fallback mechanism, if neither of these operators yields a promising operation, we employ a Boolean flip operator. This strategy is motivated by the pivotal role of Boolean variables in MIP modeling. It serves as a diversification method to escape difficult regions of the search space, complementing our main operators by exploring a different type of neighborhood.

4.1. Breakthrough move

For the first time, we propose the breakthrough move operator for finding solutions with high-quality objective values in the local search process. This operator modifies the current solution to improve the quality of the objective function, aiming to surpass the objective value of the dynamically updated best-found solution.

Definition 6. Given a variable x_j appearing in the objective function (that is, $c_j \neq 0$), and a solution $\boldsymbol{s}$ that $obj(\boldsymbol{s}) \geq obj(\boldsymbol{s}^*)$, the **breakthrough move operator**, denoted as $bm(x_j, \boldsymbol{s})$, assigns a variable x_j to the threshold value making the objective value better than $obj(\boldsymbol{s}^*)$ as possible and keeping the bounds of x_j satisfied. Precisely, let ϵ be a very small positive number (e.g., 10^{-6}) for making the objective value strictly better, $\Delta_j = (obj(\boldsymbol{s}^*) - obj(\boldsymbol{s}) - \epsilon)/c_j$, a breakthrough move operation $bm(x_j, \boldsymbol{s})$ assigns x_j to a new value s_j^{new},

$$s_j^{new} = \begin{cases} min(s_j + \Delta_j, u_j), & \text{if } c_j < 0, \\ max(\ s_j + \Delta_j, l_j), & \text{else.} \end{cases} \tag{7}$$

If x_j is an integer variable, s_j^{new} is rounded to the integer within its global bounds to optimize the objective function, specifically:

$$s_j^{new} = \begin{cases} min(\lceil s_j + \Delta_j \rceil, \lfloor u_j \rfloor), & \text{if } c_j < 0, \\ max(\lfloor s_j + \Delta_j \rfloor, \lceil l_j \rceil), & \text{else.} \end{cases} \tag{8}$$

Given the above definition, the breakthrough move operator adaptively modifies the variable to make the objective value strictly better than the best-found solution while keeping the variable's global bound satisfied. To the best of our knowledge, this is the first time that the idea of an operator breaking through the dynamically updated best-found objective value is proposed in a local search for solving combinatorial optimization problems.

Example 1. Given a MIP instance whose objective function is $obj = 3x_1 - 2x_2$, where x_1 is an integer variable with global bounds $1 \leq x_1 \leq 5$; x_2 is a real variable with $1 \leq x_2 \leq 3$. Suppose that the best-found solution is $\boldsymbol{s}^* = \{s_1 = 3, s_2 = 2\}$, thus $obj(\boldsymbol{s}^*) = 5$; the current solution is $\boldsymbol{s}^{\boldsymbol{cur}} = \{s_1 = 4, s_2 = 1.5\}$, thus $obj(\boldsymbol{s}^{\boldsymbol{cur}}) = 9$. As shown in Fig. 1, the operation $bm(x_1, \boldsymbol{s}^{\boldsymbol{cur}})$ refers to assigning x_1 to the threshold value 2 to make the objective value better than $obj(\boldsymbol{s}^*)$, and the operation $bm(x_2, \boldsymbol{s}^{\boldsymbol{cur}})$ assigns x_2 to its upper bound 3 to optimize the objective function as much as possible while satisfying the variable's bound.

4.2. Mixed tight move

In the infeasible mode, a goal throughout is to resolve violated constraints and achieve feasibility. To address this, we propose the mixed tight move operator, which integrates information from both variables and constraints to simultaneously satisfy and tighten constraints, generating promising candidate values for selection.

Definition 7. Given a variable x_j, a constraint con_i containing x_j (that is, $A_{ij} \neq 0$) and a solution $\boldsymbol{s}$, the **mixed tight move operator**, denoted as $\boldsymbol{mtm}(x_j, con_i, \boldsymbol{s})$, assigns x_j to the threshold value, making the constraint con_i as satisfied and as tight as possible, while

keeping the bounds of x_j' satisfied. Precisely, let $\Delta_{ij} = (b_i - \boldsymbol{A_i} \cdot \boldsymbol{s})/A_{ij}$, a mixed tight move operation $mtm(x_j, con_i, \boldsymbol{s})$ assigns x_j to a new value s_j^{new},

$$s_j^{new} = \begin{cases} min(s_j + \Delta_{ij}, u_j), & \text{if } \Delta_{ij} > 0, \\ max(s_j + \Delta_{ij}, l_j), & \text{if } \Delta_{ij} < 0, \\ s_j, & \text{else.} \end{cases} \tag{9}$$

If x_j is an integer variable, s_j^{new} is rounded to the nearest feasible integer to satisfy and tighten the corresponding constraint, specifically:

$$s_j^{new} = \begin{cases} min(\lceil s_j + \Delta_{ij} \rceil, \lfloor u_j \rfloor), & \text{if } b_i - \boldsymbol{A_i} \cdot \boldsymbol{s} < 0 \text{ and } A_{ij} < 0, \\ max(\lfloor s_j + \Delta_{ij} \rfloor, \lceil l_j \rceil), & \text{if } b_i - \boldsymbol{A_i} \cdot \boldsymbol{s} < 0 \text{ and } A_{ij} > 0. \\ max(\lceil s_j + \Delta_{ij} \rceil, \lceil l_j \rceil), & \text{if } b_i - \boldsymbol{A_i} \cdot \boldsymbol{s} > 0 \text{ and } A_{ij} < 0, \\ min(\lfloor s_j + \Delta_{ij} \rfloor, \lfloor u_j \rfloor), & \text{if } b_i - \boldsymbol{A_i} \cdot \boldsymbol{s} > 0 \text{ and } A_{ij} > 0, \\ s_j, & \text{else.} \end{cases} \tag{10}$$

The mixed tight move operator is designed to handle both violated and satisfied constraints, always respecting global variable bounds. Assuming that there is no global bound for each variable, its behavior adapts based on the status of the target constraint:

- For a violated constraint, the operator applies a minimal modification to a single variable to just satisfy the constraint, thereby minimizing disruption to other related constraints.
- For a satisfied constraint, the operator pushes a variable to its extreme value while ensuring that the corresponding constraint remains satisfied. This action serves as a diversification mechanism, enabling the search to escape local optima by making a maximal excursion that can influence both the objective and other constraints.

The operator's design is inspired by the principle of the Simplex algorithm [49], which systematically tightens constraints to find solutions at the vertices of a feasible polyhedron. It is also related to the critical move from Satisfiability Modulo Theories [50], which moves a variable to a specific threshold to satisfy a violated literal. Our mixed tight move is more general, however, as it not only respects global bounds but can also operate on satisfied constraints, significantly expanding the available neighborhood of operations.

A direct comparison can be made with the jump operator used in *Feasibility Jump* [44]. The jump operator selects a variable and moves it to a unique value that minimizes a sum of violation penalties, which restricts the search to a limited neighborhood. In contrast, our mixed tight move offers two primary advantages. First, it can generate multiple candidate values for a single variable, creating a much broader neighborhood from which our scoring function (Section 5.2) can select a more promising one. Second, and more importantly, this scoring function incorporates the objective value, whereas *Feasibility Jump*'s operator is guided solely by constraint violations. Consequently, the mixed tight move facilitates a more comprehensive search, balancing both feasibility and optimality. Our experimental results in Section 7 will demonstrate the practical benefits of this design.

5. New weighting scheme and scoring functions

In this section, we present the scoring functions developed for the infeasible mode, which are designed to strike an effective balance between optimization and feasibility. As a core component of local search, a scoring function evaluates the potential benefit of each candidate operation, enabling the algorithm to select the most promising one at each step. Dynamic weighting is a common technique used within scoring functions to guide and diversify the search process for combinatorial optimization problems [51,41]. Accordingly, we first propose a new weighting scheme tailored for MIP, and then, based on this scheme, propose a hierarchical, two-level scoring architecture designed to guide the search toward high-quality feasible solutions.

5.1. Weighting scheme for MIP

Weighting schemes adjust the priority of each constraint via dynamic weights [52,53]. Typically, the weights of frequently violated constraints are increased to guide the search toward feasible regions. A key challenge in designing such schemes is managing the balance between the objective function and the constraints [52]. For instance, in MaxSAT, assigning excessive weight to soft clauses can hinder the satisfaction of hard clauses, thereby impeding the search for a feasible solution [35].

We propose a new weighting scheme for MIP inspired by a probabilistic variant of the PAWS scheme [54,34,55]. It employs a smoothing probability, sp, which we set to $sp = 0.0003$ following prior work [55]. Our scheme dynamically balances the influence of the objective and the constraints by preventing excessive weight accumulation while preserving their relative importance. It operates as follows:

(a) The objective function and each constraint con_i are associated with an integer weight, denoted $w(obj)$ and $w(con_i)$, respectively.
(b) Initialization: All weights are initialized to 1, i.e., $w(obj) = 1$ and $w(con_i) = 1$.
(c) When the search becomes trapped in a local optimum (i.e., no positive operation is available), one of two mutually exclusive strategies is chosen probabilistically. The weights are then updated according to the selected strategy:

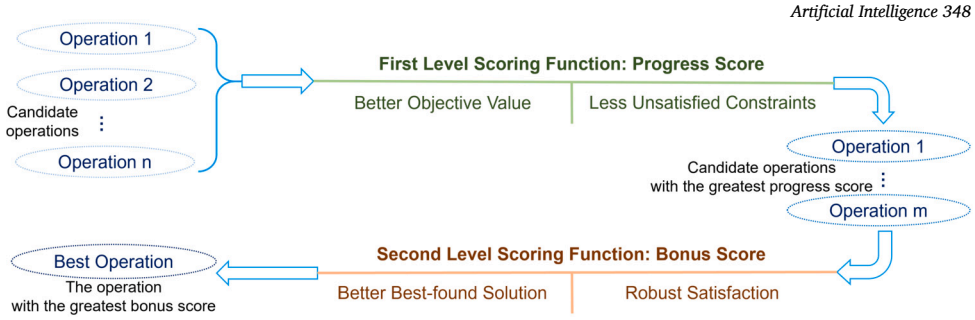

Fig. 2. Two-level Scoring Function Structure.

- Diversification (with probability $1 - sp$): If escaping a feasible local optimum, $w(obj) \leftarrow w(obj) + 1$; otherwise (i.e., if trapped at an infeasible solution), for each currently violated constraint con_i, $w(con_i) \leftarrow w(con_i) + 1$.
- Smoothing (with probability sp): If $obj(s^{cur}) < obj(s^*)$ and $w(obj) > 0$, $w(obj) \leftarrow w(obj) - 1$; for each satisfied constraint con_i whose $w(con_i) > 0$, $w(con_i) \leftarrow w(con_i) - 1$.

This scheme introduces diversity in constraint weights, reflecting their varying priorities in guiding the search. By emphasizing frequently violated constraints in local optima, it facilitates finding feasible solutions. If the search often visits feasible local optima, the objective weight increases to encourage higher-quality solutions. Conversely, if infeasible solutions with better objectives are common, the objective weight decreases to prioritize constraint satisfaction.

5.2. Two-level scoring function structure

Building on the weighting scheme, we propose a two-level scoring structure composed of a first-level progress score and a second-level bonus score, as depicted in Fig. 2. The progress score provides a primary evaluation based on direct improvements to the objective and constraint satisfaction. The bonus score then performs in a finer evaluation that rewards operations achieving strategic goals, including surpassing the best-known objective and enhancing the robustness of constraint satisfaction.

This two-level structure enables a hierarchical decision process. Candidate operations are first ranked by their progress score; the bonus score is then used as a tie-breaker to select the final operation from among the top-ranking candidates. This architecture combines immediate, greedy improvement (via the progress score) with longer-term, strategic guidance (via the bonus score). At each level, the functions assess both objective and feasibility aspects, thereby maintaining the balance between optimization and constraint satisfaction.

5.2.1. First level: progress score as base scoring function

The first level, the progress score, serves as the base evaluation metric. It adopts a purely local perspective, designed to guide the search toward an improved neighboring solution by assessing changes in both the objective function and individual constraint violations.

MIP aims to minimize the objective. Therefore, the progress score rewards the situation if the objective value decreases from the current solution and punishes it if it increases.

Definition 8. Given an operation op, and the current solution $\boldsymbol{s}^{cur}$. Let $\boldsymbol{s}^{op}$ be the new candidate solution generated by performing op on $\boldsymbol{s}^{cur}$.

$$score^{obj}_{\text{progress}}(op) = \begin{cases} w(obj), & \text{if } obj(\boldsymbol{s}^{op}) < obj(\boldsymbol{s}^{cur}), \\ -w(obj), & \text{if } obj(\boldsymbol{s}^{op}) > obj(\boldsymbol{s}^{cur}), \\ 0, & \text{else.} \end{cases} \tag{11}$$

For constraints, MIP's primary goal is to satisfy them; thus, the progress score rewards changes from violated to satisfied and punishes reverse changes. Additionally, for unsatisfied constraints, it is preferred to make them closer to being satisfied. Therefore, the progress score rewards the proximity to satisfy and punishes aggravated violations.

Definition 9. Given an operation op, a constraint con_i, and the current solution $\boldsymbol{s}^{cur}$. Let $\boldsymbol{s}^{op}$ be the new candidate solution generated by performing op on $\boldsymbol{s}^{cur}$. The progress score of op for improving the satisfaction of the constraint con_i, denoted as $score^{con_i}_{\text{progress}}(op)$,

$$score^{con_i}_{\text{progress}}(op) = \begin{cases} w(con_i), & \text{if } \boldsymbol{A_i} \cdot \boldsymbol{s}^{op} \le b_i < \boldsymbol{A_i} \cdot \boldsymbol{s}^{cur}, \\ -w(con_i), & \text{if } \boldsymbol{A_i} \cdot \boldsymbol{s}^{cur} \le b_i < \boldsymbol{A_i} \cdot \boldsymbol{s}^{op}, \\ w(con_i)/2, & \text{if } b_i < \boldsymbol{A_i} \cdot \boldsymbol{s}^{op} < \boldsymbol{A_i} \cdot \boldsymbol{s}^{cur}, \\ -w(con_i)/2, & \text{if } b_i < \boldsymbol{A_i} \cdot \boldsymbol{s}^{cur} < \boldsymbol{A_i} \cdot \boldsymbol{s}^{op}, \\ 0, & \text{else.} \end{cases} \tag{12}$$

Considering the objective function and all the constraints, the progress score is defined below.

Definition 10. Given an operation op, the progress score of op, denoted as $score_{\text{progress}}(op)$,

$$score_{\text{progress}}(op) = score^{obj}_{\text{progress}}(op) + \sum_{i=1}^{m} score^{con_i}_{\text{progress}}(op) \tag{13}$$

According to the above definitions, the progress score measures the benefits of improving objective value and satisfying constraints. Performing the operations with positive progress scores indicates the overall progress made based on the current solution.

5.2.2. Second level: bonus scoring function

The progress score serves as the primary evaluation metric, identifying operations that offer immediate improvement. However, its capacity to distinguish between the best operations is limited. Our preliminary experiments, involving a 10,000-step local search on all test instances, reveal that in 47.5% of the steps, multiple candidate operations yield the same, maximal progress score. This high frequency of ties necessitates a more refined, secondary scoring function for effective tie-breaking. While prior work in local search often uses variable age as a tie-breaker [56], this heuristic is irrelevant to the structure of MIPs, and as our experiments confirm, provides insufficient guidance.

Therefore, to select from a set of operations with identically high progress scores, we shift from the purely local perspective of the progress score. We introduce a secondary evaluation based on the global search trajectory and properties. To this end, we design the bonus scoring function as the second level, denoted as $score_{\text{bonus}}$, which contains the breakthrough bonus for the objective function and the robustness bonus for constraints, allowing for distinctions in operations based on the best progress score.

Breakthrough bonus. For the objective function, we propose the breakthrough bonus to reward situations where the new candidate solution is better than the best-found dynamically updated solution for the objective value, working in conjunction with the proposed breakthrough move operator.

Definition 11. Given an operation op, and the best-found solution $\boldsymbol{s}^*$. Let $\boldsymbol{s}^{op}$ be the new candidate solution generated by performing op on the current solution. The breakthrough bonus of op for breaking through the objective value of $\boldsymbol{s}^*$, denoted as $bonus_{\text{break}}(op)$,

$$bonus_{\text{break}}(op) = \begin{cases} w(obj), & \text{if } obj(\boldsymbol{s}^{op}) < obj(\boldsymbol{s}^*), \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

Robustness bonus. For a constraint $\boldsymbol{A_i} \cdot \boldsymbol{x} \le b_i$, we take a further step by distinguishing satisfied constraints in terms of the equality between the left-hand side $\boldsymbol{A_i} \cdot \boldsymbol{x}$ and the right-hand side b_i. For the current solution $\boldsymbol{s^{cur}}$, a special type of satisfied constraint is $\boldsymbol{A_i} \cdot \boldsymbol{s^{cur}} = b_i$. Although satisfied, these constraints would become violated more easily than other satisfied constraints because they are fragile and sensitive to the operations of variables contained in them. Therefore, we propose the robustness bonus to reward the operations that maintain strict inequality, i.e., the left-hand side is strictly less than the right-hand side.

Definition 12. Given an operation op, a constraint con_i. Let $\boldsymbol{s}^{op}$ be the new candidate solution generated by performing op on the current solution. The robustness bonus of op of the constraint con_i, denoted as $bonus^{con_i}_{\text{robust}}(op)$,

$$bonus^{con_i}_{\text{robust}}(op) = \begin{cases} w(con_i), & \text{if } \boldsymbol{A_i} \cdot \boldsymbol{s}^{op} < b_i, \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

To our knowledge, this is the first time distinctions in satisfied constraints are utilized for scoring functions in local search.

Synthesizing the breakthrough bonus for breaking through the best-found solution and the robustness bonus for robust satisfaction, the bonus scoring function that serves in the second level is defined below.

Definition 13. Given an operation op, the bonus score of op, denoted as $score_{\text{bonus}}(op)$,

$$score_{\text{bonus}}(op) = bonus_{\text{break}}(op) + \sum_{i=1}^{m} bonus^{con_i}_{\text{robust}}(op) \tag{16}$$

Algorithm 2: The Local-MIP Algorithm.

Input: MIP instance Q, time limit *cutoff*
Output: Best-found solution s^* of Q and its objective value $obj(s^*)$

1 $s^{cur} \leftarrow$ all variables are set to the value closest to 0 within their global bounds;
2 $s^* \leftarrow \emptyset$; $obj(s^*) \leftarrow +\infty$;
3 **while** *running time* $<$ *cutoff* **do**
4 **if** s^{cur} *is feasible* **then**
  // Feasible Mode
5 Improve the objective value while maintaining feasibility by the lift process in Algorithm 1;
6 **if** $obj(s^{cur}) < obj(s^*)$ **then**
7 $s^* \leftarrow s^{cur}$; $obj(s^*) \leftarrow obj(s^{cur})$;
  // Infeasible Mode
8 $candOP \leftarrow Get\_Candidate\_Operations(Q, s^{cur})$ in Algorithm 3;
9 $candOP^+ \leftarrow$ operation(s) with the greatest progress score in $candOP$;
10 $op \leftarrow$ an operation with the greatest bonus score in $candOP^+$;
11 $s^{cur} \leftarrow$ a new solution generated by performing op to modify s^{cur} ;
12 **return** s^* and $obj(s^*)$;

In summary, this two-level scoring architecture provides a hierarchical evaluation. The progress score drives local improvement, while the bonus score offers strategic, global guidance. The breakthrough bonus pushes the search toward better global optima, and the robustness bonus promotes solutions that are more deeply inside the feasible region. By combining these local and global perspectives, our framework effectively balances optimization and feasibility, enhancing the overall algorithm's adaptability and performance.

6. The local-MIP algorithm

Based on the ideas proposed in previous sections, we develop a local search algorithm for solving MIP, called *Local-MIP*. The pseudocode of *Local-MIP* is described in Algorithm 2.

We use s^{cur} to denote the current solution that is maintained during the search process, while s^* and $obj(s^*)$ denote the best-found solution and its objective value. Initially, each variable of s^{cur} is initialized as the closest value to 0 within its global bounds (line 1). s^* is initialized as an empty set, and $obj(s^*)$ is initialized as $+\infty$ (line 2). After initialization, *Local-MIP* performs the search process until a given time limit *cutoff* is reached (lines 3–11).

Once the current solution is feasible, the search process enters feasible mode (line 4). The lift process is used to improve the objective value of the current solution by efficiently applying multiple lift move operators consecutively, until it converges to a local optimum (Algorithm 1). The best-found solution and the corresponding objective value are updated once a new better feasible solution is discovered (lines 6-7).

Afterwards, the current solution is either infeasible from initialization or feasible from the local optimum of the lift process, and the search process changes to the infeasible mode. The algorithm generates neighborhood solutions by producing candidate operations (line 8), which are encapsulated by the $Get\_Candidate\_Operations(Q, s^{cur})$ function described in detail in Algorithm 3. Then, *Local-MIP* selects the operations with the greatest progress score, if there is only one such operation, it is applied directly; otherwise, the second level bonus scoring function is applied, and an operation with the greatest bonus score is selected and performed (lines 9-10), to modify the current solution to get a new s^{cur} (line 11).

6.1. Candidate operations in the infeasible mode

The candidate solutions in the infeasible mode are constructed by Algorithm 3, containing the breakthrough move operation and the mixed tight move operation proposed in Section 4. For brevity, we denote the breakthrough move as ***bm***, and denote the mixed tight move as ***mtm***. At the beginning of the search, the top priority is to find the first feasible solution; thus, it first considers the positive ***mtm*** operations in violated constraints (lines 1-3). For violated constraints in an infeasible solution, there are always ***mtm*** operations that improve the satisfaction of these violated constraints. However, there may be no positive operations since their scores are defined on all constraints, as they may reduce the satisfaction of other constraints.

Once any feasible solution has been found, the goal of the infeasible mode is transformed into discovering feasible solutions with higher-quality objective values. Therefore, the primary candidate operations are the union of positive ***bm*** operations and positive ***mtm*** operations in violated constraints (lines 5-6). If there are no such positive operations, it tries to construct the candidate operations set with the positive ***mtm*** operations in satisfied constraints (lines 7-8).

If the algorithm fails to find any positive operations in the previous process, it attempts to identify positive Boolean flip operations as candidate operations (lines 9-11). Once the above exploration fails, it is indicated that the local search has fallen into a local optimum (line 12). It first activates the weighting scheme to update the weights of the objective function and constraints (line 13).

Algorithm 3: Get_Candidate_Operations.

Input: MIP instance Q, current solution s^{cur}
Output: Candidate operations set $candOP$

1 **if** *no feasible solution is found* **then**
2 **if** ∃ *positive* ***mtm*** *operation in violated constraints* **then**
3 $candOP \leftarrow$ positive ***mtm*** operations in violated constraints;
4 **else**
5 **if** ∃ *positive* ***bm*** *operation or* ***mtm*** *operation in violated constraints* **then**
6 $candOP \leftarrow$ positive ***bm*** operations ∪ ***mtm*** operations in violated constraints;
7 **else if** ∃ *positive* ***mtm*** *operation in satisfied constraints* **then**
8 $candOP \leftarrow$ positive ***mtm*** operations in satisfied constraints;
9 **if** $op == \emptyset$ **then**
10 **if** ∃ *positive Boolean flip operation* **then**
11 $candOP \leftarrow$ positive *Boolean flip* operations;
12 **else**
13 Activate the weighting scheme to update the weights;
14 $candOP \leftarrow$ ***bm*** operations ∪ random selected ***mtm*** operations;
15 **return** $candOP$;

Afterward, it randomly selects a violated constraint if one exists and then generates the candidate operations set by the union of ***bm*** operations and ***mtm*** operations in the selected constraint (line 14).

In addition to the main part, as shown in Algorithms 1, 2, and 3, we also introduce a forbidden strategy and BMS sampling to further improve efficiency.

Local search methods may often get stuck in suboptimal regions. To address the cycling phenomenon of revisiting the same regions, we employ a forbidden strategy, the tabu strategy [57,55]. The tabu strategy is applied directly to *Local-MIP*. Once a variable is modified, it forbids modification in the reverse direction for the following tt iterations, where tt is referred to as tabu tenure. We set $tt = 3 + rand(10)$, as mentioned in [55].

We adopt the Best from Multiple Selections (BMS) sampling strategy [58] to sample both constraint and operator selection in Algorithm 3. BMS randomly samples a small and fixed number of constraints or operators and applies the best one among them. Specifically, we use predetermined sample sizes: 12 for violated constraints, 20 for satisfied constraints, and 2000, 3000, 190, 20, and 150 samples for the operations in lines 3, 6, 8, 11, and 15 of Algorithm 3, respectively. These values were selected based on preliminary experiments conducted during the initial design phase to balance convergence speed and solution quality.

7. Experimental evaluations

Here, we introduce the preliminaries and results of our experiments. First, we compare *Local-MIP* with 5 state-of-the-art MIP solvers, including *Gurobi*, *CPLEX*, *SCIP*, *HiGHS*, and *Feasibility Jump*. Moreover, we report new records established by *Local-MIP* for 10 open instances in the MIPLIB dataset. In addition, we analyze the effectiveness of the proposed strategies, perform a sensitivity analysis on various parameter settings, and evaluate the stability of *Local-MIP* with different random seeds.

7.1. Experiment preliminaries

7.1.1. Benchmarks

Our experiments are carried out with four benchmarks from the standard dataset for MIP, i.e., MIPLIB 2017 [59] with hard[4] and open[5] instances. In each instance of MIPLIB, at least one variable is an integer variable. Depending on the types of variables, the MIPLIB instances are classified into four benchmarks.

- **MIPLIB-BP**: the binary programming (66 instances), only contains Boolean variables.
- **MIPLIB-IP**: the integer programming (32 instances), where all variables are integer variables, and at least one variable is not a Boolean variable.
- **MIPLIB-MBP**: the mixed binary programming (195 instances), where all variables are Boolean or real variables, and at least one variable is a real variable.
- **MIPLIB-MIP**: the mixed integer programming (62 instances), where integer variables and real variables both exist, and at least one of the integer variables is not a Boolean variable.

[4] https://miplib.zib.de/downloads/hard-v22.test.
[5] https://miplib.zib.de/downloads/open-v22.test.

In addition, two practical problems are tested: the bin-packing problem and the scheduling problem. These are challenging combinatorial optimization problems with significant applications in the real-world industry. We evaluated the solvers on one standard bin packing benchmark provided by Falkenauer instances [60], and two standard scheduling benchmarks provided by Taillard instances [61].

- **BBP**: the Bin Packing problem. This benchmark consists of 60 instances with 500 and 1000 items to pack, encoded by the modeling method proposed in [62].
- **JSP**: the Job-shop Scheduling problem. This benchmark consists of 80 instances encoded by the modeling method proposed in [7].
- **OSP**: the Open-shop Scheduling problem. This benchmark consists of 60 instances encoded by the modeling method proposed in [63].

7.1.2. State-of-the-art competitors

In Section 7.2, we compare *Local-MIP* with 5 state-of-the-art MIP solvers.

- ***HiGHS*** [29]: an academic solver for large-scale sparse MIP (version 1.6.0).[6]
- ***SCIP*** [28]: one of the fastest academic solvers for MIP (Version 8.1.0, using SoPlex 6.0.4 as internal LP solver).[7]
- ***Gurobi*** [26]: the most powerful commercial MIP solvers (version 11.0.0). We use both its complete and heuristic versions, denoted by $\text{Gurobi}_{\text{comp}}$[8] and $\text{Gurobi}_{\text{heur}}$[9], respectively.
- ***CPLEX*** [27]: a famous commercial MIP solver to solve complex models (version 22.1.0).[10]
- ***Feasibility Jump*** [44]: *FJ* for short, the state-of-the-art local search algorithm MIP without considering the objective function, which won 1st place in MIP 2022 Computational Competition.[11]

All competitors are downloaded from their websites, and the default settings are always used. Note that *HiGHS*, *SCIP*, *Gurobi*, and *CPLEX* try to do much more than find a high-quality solution quickly. They also try to find optimality certificates and guarantee a feasible solution.

7.1.3. Experiment setup

Local-MIP is implemented in C++ and compiled in g++ with the '-O3' option. All experiments are carried out on a server with AMD EPYC 9654 CPU and 2048G RAM under the system Ubuntu 20.04.4. We use two metrics to evaluate the performance of each solver for the ability to find high-quality feasible solutions in a reasonable time:

- ***#Feas***: the number of instances in which a solver can find a feasible solution within given time limits. This evaluates a solver's ability to find feasible solutions.
- ***#Win***: the number of instances in which the solver yields the best solution among all solvers within time limits. This evaluates the ability to find high-quality feasible solutions.

For both *#Feas* and *#Win* on a benchmark, a larger metric value indicates better performance on the corresponding benchmark. For each instance, each solver is executed by one thread, and its time limits are 10, 60, and 300 seconds. For each time limit setting in the table, the best performance for the corresponding benchmark is highlighted in **bold**, and grids with better performance are shaded with deeper colors. Additionally, the number of instances in each benchmark is denoted by *#Inst*. Detailed results and source code are made publicly available on GitHub.[12]

7.2. Comparison with state-of-the-art MIP solvers

The comparison results with 5 state-of-the-art MIP solvers are presented in Tables 1 and 2.

The ability to find feasible solutions (*#Feas*). As shown in Table 1, *Local-MIP* performs best on four benchmarks in the time limits of 10 and 60 seconds and six benchmarks in 300 seconds. For all benchmarks, *Local-MIP* performs best in most benchmarks in the 60 s and 300 s time limits, and the second most in the 10 s.

Regarding the total instances of all benchmarks, *Local-MIP* establishes the best performance for all time limits. Focusing on MIPLIB benchmarks specifically, *Local-MIP* consistently outperforms all academic solvers across all time limits. In general, this result confirms the ability of *Local-MIP* to find feasible solutions within short time limits for challenging problems.

[6] https://github.com/ERGO-Code/HiGHS.
[7] https://www.scipopt.org.
[8] https://www.gurobi.com/solutions/gurobi-optimizer/.
[9] https://www.gurobi.com/documentation/10.0/refman/heuristics.html, setting the parameter *Heuristics* to 1.
[10] https://www.ibm.com/products/ilog-cplex-optimization-studio.
[11] https://github.com/sintef/feasibilityjump.
[12] https://github.com/shaowei-cai-group/Local-MIP/.

Table 1
Empirical comparison of *Local-MIP* with state-of-the-art MIP solvers in terms of ***#Feas*** for each given time limit. The best values within each benchmark are highlighted in bold, and grids with better performance are shaded with deeper colors. (For interpretation of the references to color please refer to the web version of this article.)

Benchmark	*#Inst*	*HiGHS*	*SCIP*	*CPLEX*	$\text{Gurobi}_{\text{comp}}$	$\text{Gurobi}_{\text{heur}}$	*FJ*	*Local-MIP*
Time Limit: 10 Seconds								
MIPLIB-BP	66	6	29	42	**44**	**44**	42	**44**
MIPLIB-IP	32	7	11	**17**	**17**	**17**	11	**17**
MIPLIB-MBP	195	57	80	116	117	**119**	56	103
MIPLIB-MIP	62	9	21	32	**37**	**37**	18	35
BPP	60	9	0	**60**	**60**	**60**	**60**	**60**
JSP	80	22	**70**	31	10	12	0	45
OSP	60	48	**60**	28	47	42	1	**60**
MIPLIB(Total)	355	79	141	207	215	**217**	127	199
Total	555	158	271	326	332	331	188	**364**
Time Limit: 60 Seconds								
MIPLIB-BP	66	14	35	43	46	47	**49**	48
MIPLIB-IP	32	12	14	20	20	20	12	**21**
MIPLIB-MBP	195	96	109	129	**137**	134	62	119
MIPLIB-MIP	62	15	28	36	41	41	20	**43**
BPP	60	40	20	**60**	**60**	**60**	**60**	**60**
JSP	80	41	**70**	52	23	26	1	54
OSP	60	58	**60**	30	53	51	9	**60**
MIPLIB(Total)	355	137	186	228	**244**	242	143	231
Total	555	276	336	370	380	379	213	**405**
Time Limit: 300 Seconds								
MIPLIB-BP	66	22	42	43	47	48	**49**	**49**
MIPLIB-IP	32	14	17	21	21	**22**	12	**22**
MIPLIB-MBP	195	115	122	137	150	**152**	67	123
MIPLIB-MIP	62	24	34	38	44	43	21	**45**
BPP	60	47	40	**60**	**60**	**60**	**60**	**60**
JSP	80	49	**70**	68	36	34	1	**70**
OSP	60	**60**	**60**	33	55	**60**	19	**60**
MIPLIB(Total)	355	175	215	239	262	**265**	149	239
Total	555	331	385	400	413	419	229	**429**

Table 2
Empirical comparison of *Local-MIP* with state-of-the-art MIP solvers in terms of ***#Win*** for each given time limit. The best values within each benchmark are highlighted in bold, and grids with better performance are shaded with deeper colors.

Benchmark	*#Inst*	*HiGHS*	*SCIP*	*CPLEX*	$\text{Gurobi}_{\text{comp}}$	$\text{Gurobi}_{\text{heur}}$	*FJ*	*Local-MIP*
Time Limit: 10 Seconds								
MIPLIB-BP	66	0	2	11	9	7	4	**31**
MIPLIB-IP	32	0	2	4	**9**	8	1	7
MIPLIB-MBP	195	1	7	32	43	**51**	10	35
MIPLIB-MIP	62	2	0	10	11	13	4	**15**
BPP	60	0	0	0	0	0	0	**60**
JSP	80	0	25	0	1	8	0	**36**
OSP	60	22	20	7	27	25	0	**45**
MIPLIB(Total)	355	3	11	57	72	79	19	**88**
Total	555	25	56	64	100	112	19	**229**
Time Limit: 60 Seconds								
MIPLIB-BP	66	0	2	8	10	15	2	**28**
MIPLIB-IP	32	1	1	6	7	**9**	1	6
MIPLIB-MBP	195	6	2	34	49	**65**	9	23
MIPLIB-MIP	62	4	1	7	8	**18**	3	17
BPP	60	0	0	11	13	15	0	**33**
JSP	80	0	15	1	3	13	0	**38**
OSP	60	27	20	10	37	31	0	**42**
MIPLIB(Total)	355	11	6	55	74	**107**	15	74
Total	555	38	41	77	127	166	15	**187**
Time Limit: 300 Seconds								
MIPLIB-BP	66	1	4	6	10	**23**	0	17
MIPLIB-IP	32	2	2	5	10	**14**	1	4
MIPLIB-MBP	195	7	8	23	60	**69**	11	14
MIPLIB-MIP	62	2	1	7	11	**23**	3	16
BPP	60	0	0	**31**	30	13	0	3
JSP	80	0	0	1	10	20	0	**41**
OSP	60	38	24	13	40	43	0	**44**
MIPLIB(Total)	355	12	15	41	91	**129**	15	51
Total	555	50	39	86	171	**205**	15	139

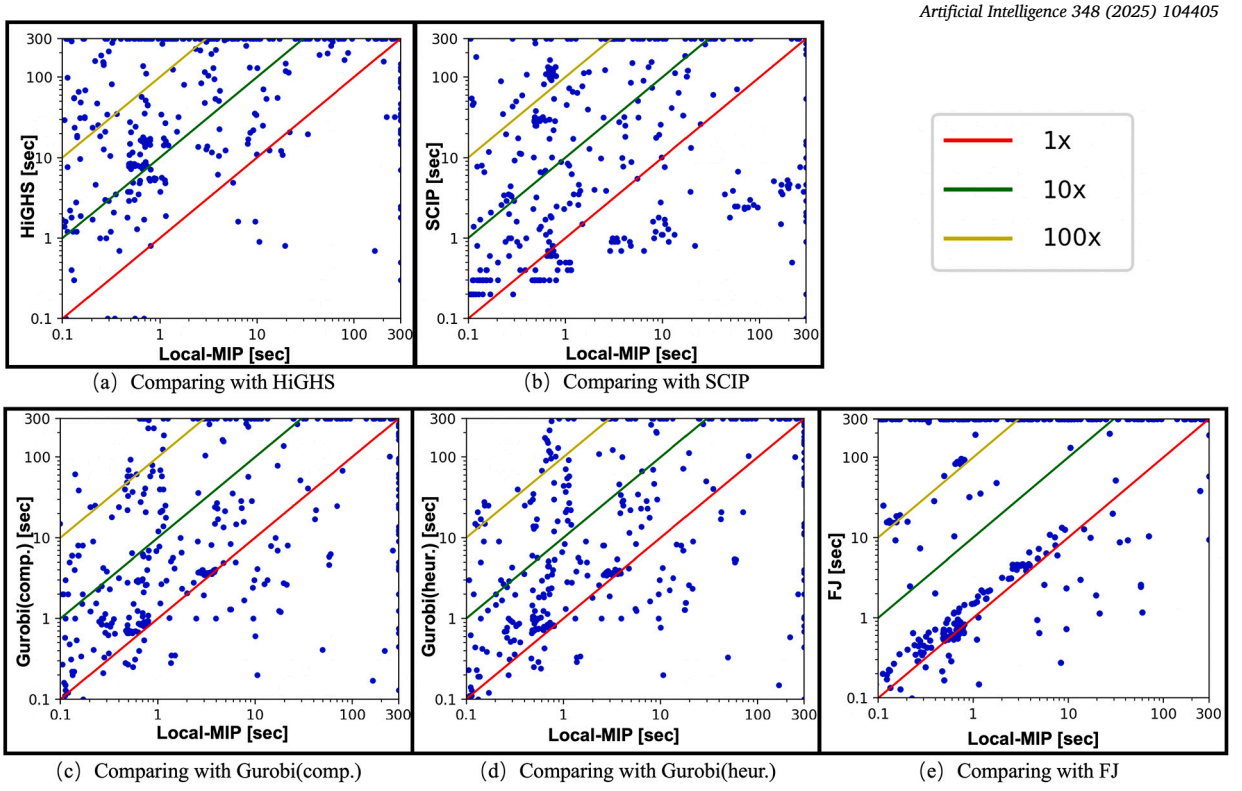

Fig. 3. Run time comparison on each instance for finding the first feasible solution. Note that the comparison with CPLEX is absent, as it does not provide the exact time for finding solutions in its default execution. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Furthermore, *Feasibility Jump* is a local search algorithm focusing only on finding feasible solutions. *Local-MIP* outperforms *Feasibility Jump* in almost all settings, confirming the effectiveness of the mixed tight operator in finding feasible solutions and indicating a performance improvement in the field of local search for MIP.

The ability to find feasible high-quality solutions (*#Win*). As shown in Table 2, *Local-MIP* performs best on five benchmarks in the time limit of 10 s, four benchmarks in 60 s, and two benchmarks in 300 s. For all benchmarks, *Local-MIP* performs best on most benchmarks in the 10 s and 60 s, and the second most in the 300 s. In particular, *Local-MIP* exhibits the best performance for JSP and OSP benchmarks across all time limits, highlighting its advantages in solving scheduling problems.

In terms of total instances, *Local-MIP* consistently establishes the best performance for 10 s and 60 s, but in 300 s, *Gurobi* wins more instances than *Local-MIP*, especially its heuristic version. On the MIPLIB benchmarks, *Local-MIP* wins 88 instances under 10 s, outperforming all competitors. At both 60 s and 300 s, however, Gurobi demonstrates superior optimization capability, with *Local-MIP* placing second. In general, for the ability to find feasible high-quality solutions within short time limits for challenging problems, *Local-MIP* significantly outperforms the two academic MIP solvers *HiGHS* and *SCIP*, and another local search algorithm *Feasibility Jump*. Moreover, *Local-MIP* performs better than the commercial MIP solver *CPLEX* and is competitive with the most powerful solver *Gurobi*.

Note that, to ensure fairness, we apply a numerical tolerance of 1×10^{-6}: a solution is deemed superior only if its objective value exceeds that of the competitor by more than this threshold.

The run time to find the first feasible solution. Furthermore, the runtime comparison for all instances in finding the first feasible solution is presented in Fig. 3. For comparison with each solver, there are obviously more instances above the red line, which confirms the effectiveness of *Local-MIP* in rapidly finding high-quality feasible solutions.

7.3. New records to open instances

In the MIPLIB dataset, the instances labeled as open are those for which the optimal solution has not yet been proved. The current best solutions for each open instance are available on the corresponding page on the MIPLIB website. These open instances are representative of challenging problems.

Table 3
Local-MIP establishes new records to 10 open instances. #var and #cons denote the number of variables and constraints of the corresponding instance, respectively.

Instance name	#var	#cons	Constraint Types	Previous	*Local-MIP*
sorrell7	2048	78848	variable bound	-196	-197
genus-sym-g31-8	3484	32073	knapsack, etc.	-21	-23
supportcase22	7129	260602	mixed binary, etc.	N/A	117
cdc7-4-3-2	11811	14478	set packing	-289	-294
genus-sym-g62-2	12912	78472	set partitioning, etc.	-34	-38
genus-g61-25	14380	94735	cardinality, etc.	-34	-40
ns1828997	27275	81725	precedence, etc.	9	8
neos-4232544-orira	87060	180600	aggregations, etc.	17540506.0	15108527.5
scpm1	500000	5000	set covering	554	544
scpn2	1000000	5000	set covering	501	490

Table 4
Comparison between *Local-MIP* and its modified versions for the feasible mode. *#B* and *#W* denote the number of instances where *Local-MIP* obtains better and worse best-found solution, respectively.

Benchmark	*#Inst*	10 seconds		60 seconds		300 seconds		10 seconds		60 seconds		300 seconds	
		#B	*#W*	*#B*	*#W*	*#B*	*#W*	*#B*	*#W*	*#B*	*#W*	*#B*	*#W*
		Comparison with $V_{\text{no-lift-process}}$						Comparison with $V_{\text{random-lift}}$					
MIPLIB-BP	66	**26**	0	**20**	0	**20**	0	**39**	2	**24**	6	**24**	5
MIPLIB-IP	32	**7**	0	**4**	0	**6**	0	**8**	3	**7**	2	**10**	2
MIPLIB-MBP	195	**42**	0	**40**	0	**41**	0	**67**	4	**53**	23	**53**	27
MIPLIB-MIP	62	**16**	0	**17**	0	**15**	0	**22**	4	**19**	6	**17**	9
BPP	60	**32**	0	**24**	0	**15**	0	**60**	0	**30**	0	**17**	0
JSP	80	**29**	0	**32**	0	**33**	0	**21**	15	**29**	19	**44**	21
OSP	60	**17**	0	**13**	0	**8**	0	17	**23**	**21**	16	**19**	18
Total	555	**169**	0	**150**	0	**138**	0	**234**	51	**183**	72	**184**	82

Interestingly, *Local-MIP* established the new best-known solutions for 10 open instances. The new records have been submitted to MIPLIB 2017 and have been accepted; the links to the website are indicated in the footnotes.[13] [14] [15] [16] [17] [18] [19] [20] [21] [22] As shown in Table 3, each of these 10 instances contains multiple different types of constraints,[23] simultaneously indicating the robust solving ability and its extensive applicability. Moreover, for the instance "supportcase22", *Local-MIP* found the first-ever feasible solution in history.

7.4. Effectiveness analysis on the proposed ideas

To verify the effectiveness of the proposed strategies, we performed comparative experiments on multiple alternative versions of *Local-MIP*.

The ideas in the feasible mode. We first evaluate the effectiveness of the proposed strategies in the feasible mode, which includes the lift process and the lift move operator.

- $V_{\text{no-lift-process}}$: To assess the effectiveness of the lift process and the necessity of the feasible mode which improves the objective value while maintaining feasibility, we modify *Local-MIP* by removing the feasible mode.
- $V_{\text{random-lift}}$: To evaluate the lift move operator, which generates the best objective value among all possible feasible solutions, we modify *Local-MIP* by replacing the lift move operator with a random lift operator, which makes a random move that improves the current solution within the local feasible domain.

As shown in Table 4, for the lift process, *Local-MIP* performs better than $V_{\text{no-lift-process}}$ on all benchmarks and on all time limit settings. Moreover, there is no instance where *Local-MIP* performs worse than $V_{\text{no-lift-process}}$, confirming the deterministic ability of the lift process to improve the objective value. For the lift move operator, *Local-MIP* significantly outperforms $V_{\text{random-lift}}$ in almost all settings, further verifying the effectiveness of the lift move operator in improving the objective value.

[13] https://miplib.zib.de/instance_details_sorrell7.html.
[14] https://miplib.zib.de/instance_details_genus-sym-g31-8.html.
[15] https://miplib.zib.de/instance_details_supportcase22.html.
[16] https://miplib.zib.de/instance_details_cdc7-4-3-2.html.
[17] https://miplib.zib.de/instance_details_genus-sym-g62-2.html.
[18] https://miplib.zib.de/instance_details_genus-g61-25.html.
[19] https://miplib.zib.de/instance_details_ns1828997.html.
[20] https://miplib.zib.de/instance_details_neos-4232544-orira.html.
[21] https://miplib.zib.de/instance_details_scpm1.html.
[22] https://miplib.zib.de/instance_details_scpn2.html.
[23] https://miplib.zib.de/statistics.html.

Table 5
Comparison between *Local-MIP* and its modified versions for the infeasible mode. *#B* and *#W* denote the number of instances where *Local-MIP* obtains better and worse best-found solution, respectively.

Benchmark	*#Inst*	10 seconds		60 seconds		300 seconds		10 seconds		60 seconds		300 seconds	
		#B	*#W*	*#B*	*#W*	*#B*	*#W*	*#B*	*#W*	*#B*	*#W*	*#B*	*#W*
		Comparison with $V_{\text{no-bm}}$						Comparison with $V_{\text{no-weight}}$					
MIPLIB-BP	66	**28**	7	**23**	17	**24**	18	**33**	4	**34**	7	**34**	8
MIPLIB-IP	32	**10**	2	**11**	5	**12**	4	**15**	0	**19**	0	**20**	0
MIPLIB-MBP	195	**61**	20	**68**	26	**63**	35	**95**	5	**112**	5	**116**	5
MIPLIB-MIP	62	**22**	7	**27**	8	**27**	8	**35**	0	**40**	1	**41**	0
BPP	60	**59**	0	**59**	0	**58**	0	**35**	10	**56**	0	**60**	0
JSP	80	**32**	13	**45**	8	**60**	10	**45**	0	**54**	0	**70**	0
OSP	60	**48**	5	**49**	1	**46**	3	**60**	0	**60**	0	**60**	0
Total	555	**260**	54	**282**	65	**290**	78	**318**	19	**375**	13	**401**	13
		Comparison with V_{random}						Comparison with V_{age}					
MIPLIB-BP	66	**26**	10	**27**	15	**26**	15	**27**	10	**23**	16	**26**	13
MIPLIB-IP	32	**10**	4	**13**	3	**15**	4	**11**	3	**15**	1	**15**	1
MIPLIB-MBP	195	**69**	27	**80**	30	**71**	43	**66**	28	**68**	42	**71**	41
MIPLIB-MIP	62	**22**	12	**19**	18	**18**	17	**23**	10	**23**	13	**22**	15
BPP	60	**28**	15	11	**27**	11	**34**	**34**	11	16	**29**	13	**25**
JSP	80	**29**	16	**31**	23	**39**	31	**30**	15	**31**	23	**42**	26
OSP	60	**25**	20	**27**	15	**21**	12	**29**	18	**31**	13	**22**	13
Total	555	**209**	104	**208**	131	**201**	156	**220**	95	**207**	137	**211**	134

The ideas in the infeasible mode. We also evaluate the effectiveness of the proposed strategies in the infeasible mode, including the breakthrough move operator, the weighting scheme, and the bonus score.

- $V_{\text{no-bm}}$: To analyze the effectiveness of the breakthrough move operator, we modify *Local-MIP* by removing all breakthrough move operations in Algorithm 3.
- $V_{\text{no-weight}}$: To evaluate the weighting scheme, we modify *Local-MIP* by disabling the activation of the weighting scheme in Algorithm 3 and setting all weights equal to 1.
- V_{random} and V_{age}: To assess the effectiveness of the bonus score, we modify *Local-MIP* using random selection and the *age* strategy instead of the bonus score to break the ties in Algorithm 2, resulting in versions V_{random} and V_{age}, respectively.

As shown in Table 5, *Local-MIP* significantly outperforms other variations in almost all settings, confirming the effectiveness of the proposed strategies in the infeasible mode.

7.5. Sensitivity analysis on different parameters

Here, we discuss the importance of selecting constants through experiments with different parameter settings. There are three constants in *Local-MIP*:

- sp: the smoothing probability in the weighting scheme, which we set sp $= 0.0003$ as mentioned in [50].
- mf: the mitigation factor in the progress score for the satisfaction of constraints $score^{con_i}_{\text{progress}}(op)$ (Eq. (12)), which we easily set to 2.
- tt: the tabu tenure, which we set tt $= 3 +$ rand (10), as mentioned in [50].

These three parameters, all working in the infeasible mode, primarily impact the performance of finding feasible solutions. Thus, we present the *#Feas* metric for each parameter setting, which is not only an independent metric abstained by self-run, but also excludes the influence of components from the feasible mode.

As presented in Fig. 4, the gap between the maximum *#Feas* and the minimum *#Feas* is correlated. For each benchmark, the maximum gap for sp is 3 in MIPLIB-MBP, 5 for mf in both MIPLIB-MBP and MIPLIB-MIP, and 3 for tt in MIPLIB-MBP. Overall, the results demonstrate that different parameter settings do not significantly affect the performance of *Local-MIP*. Furthermore, these three parameters' settings are highly stable in the BPP, JSP, and OSP benchmarks, with consistent *#Feas* values.

Regarding the total instances, sp is the most stable, with a maximum gap of only 2, and tt is also relatively stable, with a maximum gap of 5.

mf exhibits the maximum gap of 10 for the total instances. However, the worst parameter setting is $mf = 1$, which contradicts the idea of the mitigation strategy and is not an appropriate choice. Excluding the setting of $mf = 1$, the maximum gap is reduced to 7. Moreover, all other settings of mf outperform the setting of $mf = 1$, confirming the effectiveness of the mitigation mechanism in the progress score to satisfy the constraints.

7.6. Stability with repetitive experiments

To examine the stability of *Local-MIP* that involves randomness, we run *Local-MIP* 10 times with seeds ranging from 1 to 10, and measure the coefficient of variation [64,41].

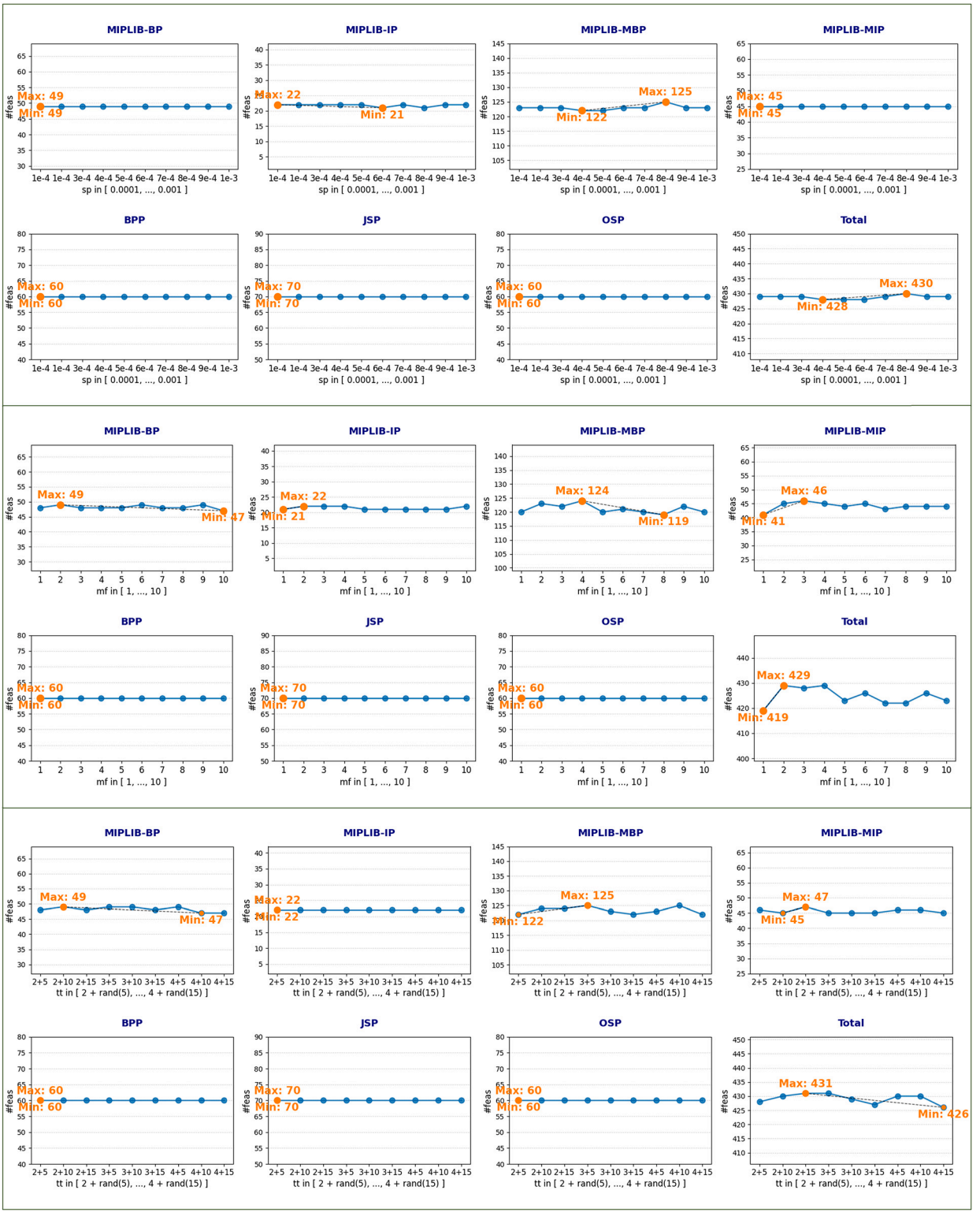

Fig. 4. The performance in terms of *#Feas* of different parameter settings for the smoothing probability sp, the mitigation factor mf, and the tabu tenure tt, respectively.

Table 6
Experimental results of *Local-MIP* with 10 different seeds on each benchmark, where *#CV* denotes the number of instances in each range of the coefficient of variation.

Time limit	#CV			
	$[0, 0.01)$	$[0.01, 0.1)$	$[0.1, 0.5)$	$[0.5, +\infty)$
10 second	**335**	140	56	24
60 second	**322**	150	60	23
300 second	**316**	160	49	30

Table 7
Empirical results on the percentage (%) of different moves of *Local-MIP* with the 300 s time limit.

Benchmark	Instances	lm	bm	mtm-v	mtm-s	flip	mtm-r
MIPLIB-BP	66	3.51	24.71	49.53	4.06	0.95	16.44
MIPLIB-IP	32	0.25	16.01	57.86	0.43	0.45	28.42
MIPLIB-MBP	195	1.89	15.15	60.02	0.81	0.36	13.03
MIPLIB-MIP	62	0.11	4.57	63.36	0.43	0.19	17.84

For each instance, we calculate the average value AVG and the standard deviation STD for the absolute objective values of the best-found solutions from 10 different seeds. The coefficient of variation for each instance is STD/AVG, and the lower the value, the greater the stability. The experimental results are presented in Table 6, where over 85% have a coefficient of variation less than 0.1, indicating that *Local-MIP* exhibits stable performance.

7.7. Analysis of different moves

Here, we analyze different types of moves based on their selection frequency in the MIPLIB benchmarks. As shown in the Table 7, where *lm* denotes the *lift move*, *bm* denotes the *breakthrough move*, *mtm-v* denotes the *mixed tight move* from violated constraints, *mtm-s* denotes the *mixed tight move* from satisfied constraints, *flip* denotes the *Boolean flip*, and *mtm-r* denotes the *mixed tight move* from the randomly selected constraint.

The complexity of the variable types increases from BP to MIP. The proportion of *bm* decreases, while the proportion of *mtm-v* increases across problem types. In particular, the proportion of *lm* is highest in the BP benchmark compared to all others. These observations suggest that, for simpler problems, such as BP, feasible solutions are relatively easy to obtain, and the search focuses more on optimizing the objective function. In contrast, for more complex problems, such as generalized MIPs, the search spends more time satisfying constraints.

7.8. Convergence analysis

We have generated convergence plots for all instances where both *Local-MIP* and all branch-and-bound solvers found feasible solutions; these are publicly available on GitHub.[24] For illustrative purposes, Fig. 5 presents the convergence curves for 20 randomly selected instances. Each plot depicts the best-found objective value (y-axis) as a function of elapsed time (x-axis) over a 300-second horizon, comparing *Local-MIP* against *SCIP*, Gurobi$_{\text{comp}}$, Gurobi$_{\text{heur}}$, and *HiGHS*.

The curves reveal that *Local-MIP*'s search behavior differs markedly from that of classical MIP solvers. Specifically, *Local-MIP* achieves rapid initial improvements but then plateaus, whereas branch-and-bound solvers typically exhibit steadier, incremental progress throughout the run. This pattern confirms that *Local-MIP* excels at quickly generating high-quality solutions but experiences diminishing returns in later stages.

7.9. Discussion of Hexaly and ViolationLS

In addition to *Feasibility Jump*, two other local search-based approaches are worth discussing: *Hexaly*[25] and *ViolationLS* [45].

Hexaly, formerly known as LocalSolver, is a commercial solver designed for general combinatorial optimization based on local search, initially focused on 0-1 programming [65]. According to its official website, *Hexaly* demonstrates strong performance on problems such as FJSP and BPP.[26] However, we were unable to include Hexaly in our empirical evaluation, as an academic license was not available to us.

ViolationLS is a recent development that integrates the *Feasibility Jump* local search strategy within the CP-SAT framework [46]. We compare *Local-MIP* with two configurations of *ViolationLS*:

[24] https://github.com/shaowei-cai-group/Local-MIP/blob/main/result/convergence_plot.zip.
[25] https://www.hexaly.com/.
[26] https://www.hexaly.com/benchmarks.

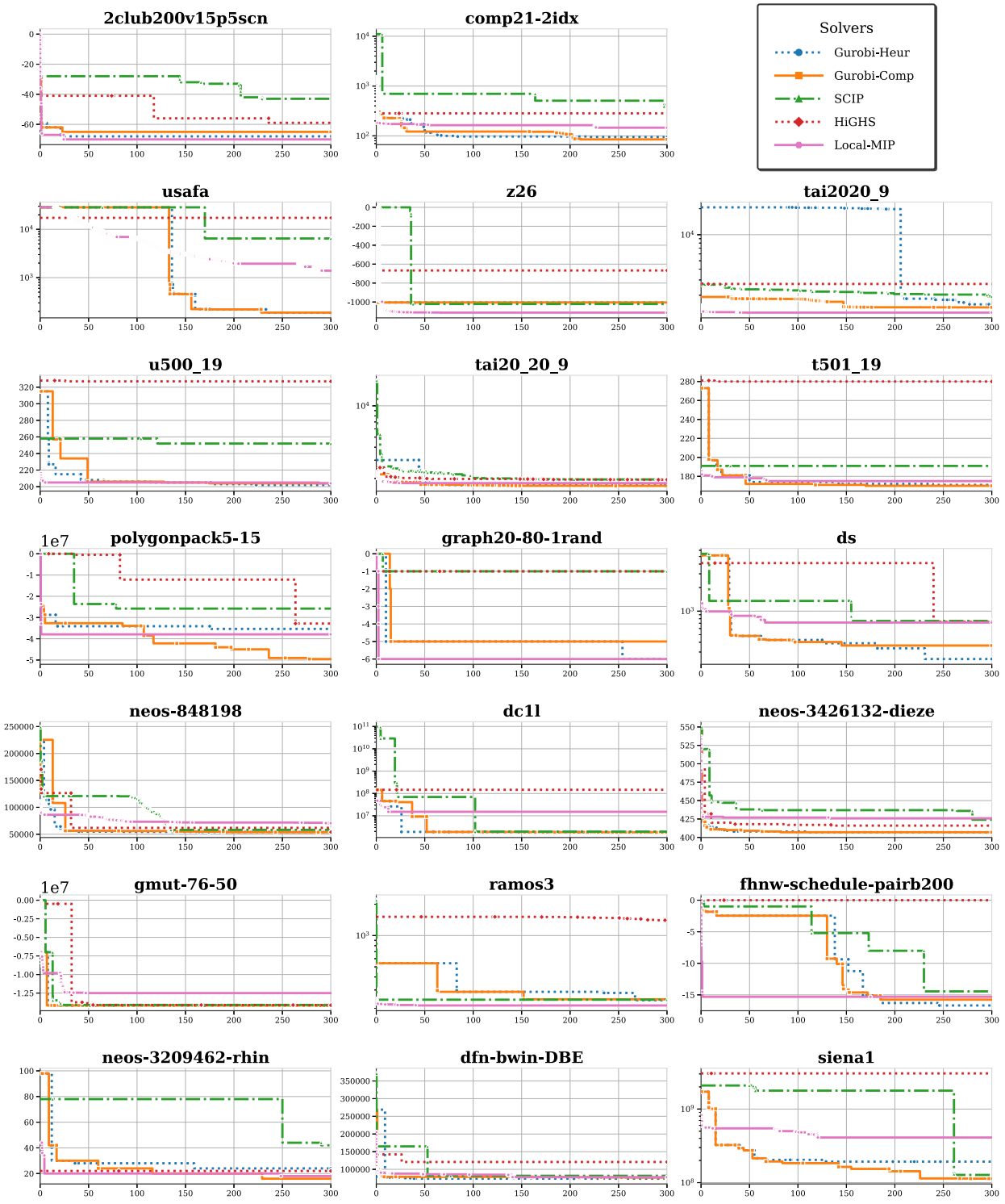

Fig. 5. Convergence curves for *Local-MIP* and branch-and-bound solvers on randomly selected instances over a 300-second limit. The y-axis shows the best-found objective value; the x-axis indicates elapsed time (s).

Table 8
Comparison between *Local-MIP* and *ViolationLS*. *#B* and *#W* denote the number of instances where *Local-MIP* obtains better and worse best-found solution, respectively.

Benchmark	*#Inst*	10 seconds		60 seconds		300 seconds		10 seconds		60 seconds		300 seconds	
		#B	*#W*	*#B*	*#W*	*#B*	*#W*	*#B*	*#W*	*#B*	*#W*	*#B*	*#W*
		Comparison with $V_{\text{CP-SAT+ViolateLS}}$						Comparison with $V_{\text{ViolateLS-pure}}$					
MIPLIB-BP	66	**37**	8	**30**	18	**28**	21	**25**	14	16	**27**	15	**29**
MIPLIB-IP	32	**14**	2	**15**	5	**16**	6	**9**	7	**11**	9	**11**	10
MIPLIB-MBP	195	**85**	23	**80**	48	**77**	53	**86**	18	**97**	23	**86**	35
MIPLIB-MIP	62	**31**	4	**38**	6	**39**	8	**31**	3	**35**	9	**35**	11
BPP	60	**60**	0	**60**	0	**60**	0	**57**	0	**34**	17	20	**40**
JSP	80	**45**	1	**53**	1	**60**	10	**45**	0	**54**	1	**70**	0
OSP	60	**40**	7	**23**	15	**20**	13	**60**	0	**60**	0	**60**	0
Total	555	**312**	45	**299**	93	**300**	111	**313**	42	**307**	86	**297**	125

1. CP-SAT+ViolationLS: by setting parameter `num_violation_ls = 1`
2. Pure ViolationLS: by setting parameters `num_violation_ls = 1` and `use_ls_only = True`

As CP-SAT supports only integer variables and coefficients, we scaled the real-valued variables and coefficients to integers for compatibility. However, this transformation inevitably introduces numerical approximation. The results, summarized in Table 8, show that *Local-MIP* performs strongly across most benchmarks, particularly at shorter time limits. However, the performance gap between *Local-MIP* and *ViolationLS* narrows as the time limit increases, highlighting the complementary strengths of these two approaches.

8. Conclusions and future work

This paper presents an efficient local search algorithm for Mixed Integer Programming (MIP), named *Local-MIP*. The proposed approach divides the search process into two modes based on the feasibility of the current solution. In the feasible mode, the lift move operator and the lift process are proposed to improve the value of the objective function while maintaining feasibility. In the infeasible mode, two novel operators are proposed to adaptively modify variables, thereby optimizing the objective function and satisfying the constraints. Additionally, a new weighting scheme is designed to dynamically balance the priority between the objective function and constraints, and a two-level scoring function structure is proposed to guide the search towards high-quality feasible solutions hierarchically. Experiments are conducted on seven public benchmarks in finding high-quality solutions. The results demonstrate that *Local-MIP* outperforms state-of-the-art MIP solvers such as *CPLEX*, *HiGHS*, *SCIP*, and *Feasibility Jump*, and is competitive with the leading commercial solver *Gurobi* for challenging problems within short time limits. Furthermore, *Local-MIP* sets 10 new records for open instances in MIPLIB.

Although *Local-MIP* excels at finding high-quality solutions quickly, its performance can plateau during later stages of the search. Future work will therefore focus on incorporating adaptive mechanisms or restart-based strategies to mitigate such stagnation, aiming to enhance the algorithm's effectiveness in long-running optimization scenarios.

CRediT authorship contribution statement

Peng Lin: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Formal analysis, Conceptualization. **Shaowei Cai:** Writing – review & editing, Project administration, Funding acquisition, Formal analysis, Conceptualization. **Mengchuan Zou:** Writing – review & editing, Formal analysis, Conceptualization. **Jinkun Lin:** Methodology.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work is supported by National Key R&D Program of China, Grant No. 2023YFA1009500.

Data availability

We have made the code, software, benchmark dataset, and detailed experimental results from this paper publicly available on GitHub: https://github.com/shaowei-cai-group/Local-MIP.

Detailed results and source code of Local-MIP (Original data) (GitHub).

References

[1] T. Achterberg, R. Wunderling, Mixed integer programming: analyzing 12 years of progress, in: Facets of Combinatorial Optimization: Festschrift for Martin Grötschel, Springer, 2013, pp. 449–481.
[2] L.A. Wolsey, Mixed integer programming, in: B.W. Wah (Ed.), Wiley Encyclopedia of Computer Science and Engineering, John Wiley & Sons, Inc., 2008.
[3] J.C. Smith, Z.C. Taskin, A tutorial guide to mixed-integer programming models and solution techniques, in: Optimization in Medicine and Biology, 2008, pp. 521–548.
[4] J. Davies, F. Bacchus, Exploiting the power of mip solvers in maxsat, in: M. Järvisalo, A.V. Gelder (Eds.), Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Proceedings, Helsinki, Finland, July 8–12, 2013, in: Lecture Notes in Computer Science, vol. 7962, Springer, 2013, pp. 166–181.
[5] A. Lodi, M. Monaci, Integer linear programming models for 2-staged two-dimensional knapsack problems, Math. Program. 94 (2003) 257–278, https://doi.org/10.1007/s10107-002-0319-9.
[6] G. Pataki, Teaching integer programming formulations using the traveling salesman problem, SIAM Rev. 45 (2003) 116–123, https://doi.org/10.1137/S00361445023685.
[7] W. Ku, J.C. Beck, Mixed integer programming models for job shop scheduling: a computational analysis, Comput. Oper. Res. 73 (2016) 165–173, https://doi.org/10.1016/j.cor.2016.04.006.
[8] R.A. Melo, C.C. Ribeiro, MIP formulations for induced graph optimization problems: a tutorial, Int. Trans. Oper. Res. 30 (2023) 3159–3200, https://doi.org/10.1111/itor.13299.
[9] Y. Pochet, L.A. Wolsey, Production Planning by Mixed Integer Programming, vol. 149, Springer, 2006.
[10] C.A. Floudas, X. Lin, Mixed integer linear programming in process scheduling: modeling, algorithms, and applications, Ann. Oper. Res. 139 (2005) 131–162, https://doi.org/10.1007/s10479-005-3446-x.
[11] S. Gertphol, V.K. Prasanna, MIP formulation for robust resource allocation in dynamic real-time systems, J. Syst. Softw. 77 (2005) 55–65, https://doi.org/10.1016/j.jss.2003.12.040.
[12] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller, J.W. Thatcher (Eds.), Proceedings of a Symposium on the Complexity of Computer Computations, Held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA, in: The IBM Research Symposia Series, Plenum Press, New York, 1972, pp. 85–103.
[13] R. Kannan, C.L. Monma, On the computational complexity of integer programming problems, in: Optimization and Operations Research: Proceedings of a Workshop Held at the University of Bonn, October 2–8, 1977, Springer, 1978, pp. 161–172.
[14] H.A. Kautz, A. Sabharwal, B. Selman, Incomplete algorithms, in: A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, second edition, in: Frontiers in Artificial Intelligence and Applications, vol. 336, IOS Press, 2021, pp. 213–232.
[15] M. Fischetti, A. Lodi, Heuristics in mixed integer programming, in: Wiley Encyclopedia of Operations Research and Management Science, 2010.
[16] A.H. Land, A.G. Doig, An automatic method for solving discrete programming problems, in: M. Jünger, T.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi, L.A. Wolsey (Eds.), 50 Years of Integer Programming 1958-2008 - from the Early Years to the State-of-the-Art, Springer, 2010, pp. 105–132.
[17] E.L. Lawler, D.E. Wood, Branch-and-bound methods: a survey, Oper. Res. 14 (1966) 699–719, https://doi.org/10.1287/opre.14.4.699.
[18] R.E. Gomory, Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed integer problem, in: M. Jünger, T.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi, L.A. Wolsey (Eds.), 50 Years of Integer Programming 1958-2008 - from the Early Years to the State-of-the-Art, Springer, 2010, pp. 77–103.
[19] M.W.P. Savelsbergh, Preprocessing and probing techniques for mixed integer programming problems, INFORMS J. Comput. 6 (1994) 445–454, https://doi.org/10.1287/ijoc.6.4.445.
[20] T. Berthold, Primal heuristics for mixed integer programs, Ph.D. thesis, Zuse Institute, Berlin (ZIB), 2006.
[21] M. Fischetti, F.W. Glover, A. Lodi, The feasibility pump, Math. Program. 104 (2005) 91–104, https://doi.org/10.1007/s10107-004-0570-3.
[22] T. Berthold, G. Hendel, Shift-and-propagate, J. Heuristics 21 (2015) 73–106.
[23] G. Gamrath, T. Berthold, S. Heinz, M. Winkler, Structure-driven fix-and-propagate heuristics for mixed integer programming, Math. Program. Comput. 11 (2019) 675–702.
[24] D. Salvagnin, R. Roberti, M. Fischetti, A fix-propagate-repair heuristic for mixed integer programming, Math. Program. Comput. (2024) 1–29.
[25] T. Achterberg, SCIP: solving constraint integer programs, Math. Program. Comput. 1 (2009) 1–41, https://doi.org/10.1007/s12532-008-0001-1.
[26] L. Gurobi, Optimization, Gurobi Optimizer Ref. Manual, 2024.
[27] S. Nickel, C. Steinhardt, H. Schlenker, W. Burkart, Ibm ilog cplex optimization studio—a primer, in: Decision Optimization with IBM ILOG CPLEX Optimization Studio: A Hands-On Introduction to Modeling with the Optimization Programming Language (OPL), Springer, 2022, pp. 9–21.
[28] K. Bestuzheva, M. Besançon, W. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A.M. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M.E. Lübbecke, S.J. Maher, F. Matter, E. Mühmer, B. Müller, M.E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, J. Witzig, Enabling research through the SCIP optimization suite 8.0, ACM Trans. Math. Softw. 49 (2023) 22, https://doi.org/10.1145/3585516.
[29] Q. Huangfu, J.A.J. Hall, Parallelizing the dual revised simplex method, Math. Program. Comput. 10 (2018) 119–142, https://doi.org/10.1007/s12532-017-0130-5.
[30] H.H. Hoos, T. Stützle, Stochastic Local Search: Foundations & Applications, Elsevier / Morgan Kaufmann, 2004.
[31] H.H. Hoos, T. Stützle, Local search algorithms for SAT: an empirical evaluation, J. Autom. Reason. 24 (2000) 421–481, https://doi.org/10.1023/A:1006350622830.
[32] A. Balint, U. Schöning, Choosing probability distributions for stochastic local search and the role of make versus break, in: A. Cimatti, R. Sebastiani (Eds.), Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Proceedings, Trento, Italy, June 17–20, 2012, in: Lecture Notes in Computer Science, vol. 7317, Springer, 2012, pp. 16–29.
[33] C.M. Li, Y. Li, Satisfying versus falsifying in local search for satisfiability - (poster presentation), in: A. Cimatti, R. Sebastiani (Eds.), Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Proceedings, Trento, Italy, June 17–20, 2012, in: Lecture Notes in Computer Science, vol. 7317, Springer, 2012, pp. 477–478.
[34] S. Cai, K. Su, Local search for Boolean satisfiability with configuration checking and subscore, Artif. Intell. 204 (2013) 75–98, https://doi.org/10.1016/j.artint.2013.09.001.
[35] B. Cha, K. Iwama, Y. Kambayashi, S. Miyazaki, Local search algorithms for partial MAXSAT, in: B. Kuipers, B.L. Webber (Eds.), Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, IAAI 97, July 27–31, 1997, AAAI Press / The MIT Press, Providence, Rhode Island, USA, 1997, pp. 263–268, http://www.aaai.org/Library/AAAI/1997/aaai97-041.php.
[36] S. Cai, Z. Lei, Old techniques in new ways: clause weighting, unit propagation and hybridization for maximum satisfiability, Artif. Intell. 287 (2020) 103354, https://doi.org/10.1016/j.artint.2020.103354.
[37] J. Zheng, K. He, J. Zhou, Y. Jin, C.-M. Li, F. Manyà, Integrating multi-armed bandit with local search for maxsat, Artif. Intell. 338 (2025) 104242, https://www.sciencedirect.com/science/article/pii/S0004370224001784, https://doi.org/10.1016/j.artint.2024.104242.

[38] Y. Chu, S. Cai, C. Luo, Nuwls: improving local search for (weighted) partial maxsat by new weighting techniques, in: B. Williams, Y. Chen, J. Neville (Eds.), Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7–14, 2023, AAAI Press, 2023, pp. 3915–3923.
[39] V. Beresnev, E. Goncharov, A. Mel'nikov, Local search with a generalized neighborhood in the optimization problem for pseudo-Boolean functions, J. Appl. Ind. Math. 6 (2012) 22–30.
[40] Z. Lei, S. Cai, C. Luo, H.H. Hoos, Efficient local search for pseudo Boolean optimization, in: C. Li, F. Manyà (Eds.), Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Proceedings, Barcelona, Spain, July 5–9, 2021, in: Lecture Notes in Computer Science, vol. 12831, Springer, 2021, pp. 332–348.
[41] Y. Chu, S. Cai, C. Luo, Z. Lei, C. Peng, Towards more efficient local search for pseudo-Boolean optimization, in: R.H.C. Yap (Ed.), 29th International Conference on Principles and Practice of Constraint Programming, CP 2023, August 27–31, 2023, Toronto, Canada, in: LIPIcs, vol. 280, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 12.
[42] J.P. Walser, R. Iyer, N. Venkatasubramanyan, An integer local search method with application to capacitated production planning, in: AAAI/IAAI, 1998, pp. 373–379.
[43] S. Prestwich, S. Verachi, Constructive vs perturbative local search for general integer linear programming, in: Proceedings of the Fifth International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS), 2008.
[44] B. Luteberget, G. Sartor, Feasibility jump: an lp-free Lagrangian MIP heuristic, Math. Program. Comput. 15 (2023) 365–388, https://doi.org/10.1007/s12532-023-00234-8.
[45] T.O. Davies, F. Didier, L. Perron, Violationls: constraint-based local search in CP-SAT, in: B. Dilkina (Ed.), Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 21st International Conference, Proceedings, Part I, CPAIOR 2024, Uppsala, Sweden, May 28–31, 2024, in: Lecture Notes in Computer Science, vol. 14742, Springer, 2024, pp. 243–258.
[46] L. Perron, F. Didier, S. Gay, The CP-SAT-LP solver, in: R.H.C. Yap (Ed.), 29th International Conference on Principles and Practice of Constraint Programming (CP 2023), in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 280, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2023, 3, https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2023.3.
[47] P. Lin, M. Zou, S. Cai, An efficient local search solver for mixed integer programming, in: P. Shaw (Ed.), 30th International Conference on Principles and Practice of Constraint Programming, CP 2024, September 2–6, 2024, Girona, Spain, in: LIPIcs, vol. 307, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 19.
[48] P. Lin, S. Cai, M. Zou, J. Lin, New characterizations and efficient local search for general integer linear programming, arXiv preprint arXiv:2305.00188, 2023, https://arxiv.org/abs/2305.00188.
[49] G.B. Dantzig, A. Orden, P. Wolfe, et al., The generalized simplex method for minimizing a linear form under linear inequality restraints, Pac. J. Math. 5 (1955) 183–195.
[50] S. Cai, B. Li, X. Zhang, Local search for smt on linear integer arithmetic, in: Computer Aided Verification: 34th International Conference, Proceedings, Part II, CAV 2022, Haifa, Israel, August 7–10, 2022, Springer, 2022, pp. 227–248.
[51] Z. Lei, S. Cai, Solving (weighted) partial maxsat by dynamic local search for SAT, in: J. Lang (Ed.), Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13–19, 2018, ijcai.org, Stockholm, Sweden, 2018, pp. 1346–1352.
[52] J. Thornton, A. Sattar, Dynamic constraint weighting for over-constrained problems, in: H. Lee, H. Motoda (Eds.), PRICAI'98, Topics in Artificial Intelligence, 5th Pacific Rim International Conference on Artificial Intelligence, Proceedings, Singapore, November 22–27, 1998, in: Lecture Notes in Computer Science, vol. 1531, Springer, 1998, pp. 377–388.
[53] J. Thornton, Clause weighting local search for SAT, J. Autom. Reason. 35 (2005) 97–142, https://doi.org/10.1007/s10817-005-9010-1.
[54] J. Thornton, D.N. Pham, S. Bain, V.F. Jr., Additive versus multiplicative clause weighting for SAT, in: D.L. McGuinness, G. Ferguson (Eds.), Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, San Jose, California, USA, July 25–29, 2004, AAAI Press / The MIT Press, 2004, pp. 191–196, http://www.aaai.org/Library/AAAI/2004/aaai04-031.php.
[55] S. Cai, B. Li, X. Zhang, Local search for SMT on linear integer arithmetic, in: S. Shoham, Y. Vizel (Eds.), Computer Aided Verification - 34th International Conference, Proceedings, Part II, CAV 2022, Haifa, Israel, August 7–10, 2022, in: Lecture Notes in Computer Science, vol. 13372, Springer, 2022, pp. 227–248.
[56] S. Cai, C. Luo, K. Su, Scoring functions based on second level score for k-sat with long clauses, J. Artif. Intell. Res. 51 (2014) 413–441, https://doi.org/10.1613/jair.4480.
[57] F. Glover, M. Laguna, Tabu Search, Springer, 1998.
[58] S. Cai, Balance between complexity and quality: local search for minimum vertex cover in massive graphs, in: Q. Yang, M.J. Wooldridge (Eds.), Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25–31, 2015, AAAI Press, 2015, pp. 747–753, http://ijcai.org/Abstract/15/111.
[59] A.M. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. Christophel, K. Jarck, T. Koch, J.T. Linderoth, M.E. Lübbecke, H.D. Mittelmann, D.B. Özyurt, T.K. Ralphs, D. Salvagnin, Y. Shinano, MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library, Math. Program. Comput. 13 (2021) 443–490, https://doi.org/10.1007/s12532-020-00194-3.
[60] E. Falkenauer, A hybrid grouping genetic algorithm for bin packing, J. Heuristics 2 (1996) 5–30, https://doi.org/10.1007/BF00226291.
[61] E. Taillard, Benchmarks for basic scheduling problems, Eur. J. Oper. Res. 64 (1993) 278–285.
[62] M. Delorme, M. Iori, S. Martello, Bin packing and cutting stock problems: mathematical models and exact algorithms, Eur. J. Oper. Res. 255 (2016) 1–20, https://doi.org/10.1016/j.ejor.2016.04.030.
[63] B. Naderi, M. Zandieh, Modeling and scheduling no-wait open shop problems, Int. J. Prod. Econ. 158 (2014) 256–266.
[64] H. Abdi, Coefficient of variation, Encycl. Res. Design 1 (2010).
[65] T. Benoist, B. Estellon, F. Gardi, R. Megel, K. Nouioua, Localsolver 1.x: a black-box local-search solver for 0-1 programming, 4OR 9 (2011) 299–316, https://doi.org/10.1007/S10288-011-0165-9.