# An Efficient Local Search Solver for Mixed Integer Programming

**Peng Lin**
Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China

**Mengchuan Zou**
Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

**Shaowei Cai**[1]
Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China

## Abstract

Mixed integer programming (MIP) is a fundamental model in operations research. Local search is a powerful method for solving hard problems, but the development of local search solvers for MIP still needs to be explored. This work develops an efficient local search solver for solving MIP, called *Local-MIP*. We propose two new operators for MIP to adaptively modify variables for optimizing the objective function and satisfying constraints, respectively. Furthermore, we design a new weighting scheme to dynamically balance the priority between the objective function and each constraint, and propose a two-level scoring function structure to hierarchically guide the search for high-quality feasible solutions. Experiments are conducted on seven public benchmarks to compare *Local-MIP* with state-of-the-art MIP solvers, which demonstrate that *Local-MIP* significantly outperforms *CPLEX*, *HiGHS*, *SCIP* and *Feasibility Jump*, and is competitive with the most powerful commercial solver *Gurobi*. Moreover, *Local-MIP* establishes 4 new records for MIPLIB open instances.

**2012 ACM Subject Classification** Mathematics of computing → Integer programming; Theory of computation → Randomized local search; Applied computing → Operations research

**Keywords and phrases** Mixed Integer Programming, Local Search, Operator, Scoring Function

**Digital Object Identifier** 10.4230/LIPIcs.CP.2024.19

**Supplementary Material** *Software (Source Code)*: `https://github.com/shaowei-cai-group/Local-MIP` [31], archived at `swh:1:dir:883191ffb9b4503105cce3e9d3da6d50421956f3`

**Funding** This work is supported by National Key R&D Program of China (2023YFA1009500).

## 1 Introduction

Mixed integer programming (MIP) is a fundamental mathematical model in operations research [3], which represents the problem of optimizing a linear objective function under linear constraints, where some variables are restricted to taking integer values. Due to its powerful expressive ability, MIP is widely used both in academic areas and in industrial sectors to model various problems [46, 41]. In previous research, many combinatorial optimization problems can be described by MIP formulation, such as the Boolean satisfiability

[1] Corresponding author

© Peng Lin, Mengchuan Zou, and Shaowei Cai;
licensed under Creative Commons License CC-BY 4.0

(SAT) and maximum satisfiability (MaxSAT) [13], the knapsack problem [32], the traveling salesman problem (TSP) [37], the job-shop scheduling problem (JSP) [26], and many graph problems [34]. In the industry domain, MIP could model lots of optimization problems in applications, including production planning [38], crew scheduling [16], resource allocation [17], and so on.

Solving MIP is a challenging task as the problem is NP-hard [25, 24]. The solving methods can be divided into two classes: complete methods and incomplete methods. While complete methods aim at computing the exact optimal solution and proving its optimality, incomplete methods aim to obtain high-quality solutions within a reasonable time. In real-world applications, due to the large scale of instances, problems are usually difficult to handle by complete methods. On the other hand, high-quality solutions usually show good usability in practical applications; thus, incomplete methods are of great importance in MIP solving.

Existing MIP solvers, however, rely primarily on complete methods. The most commonly admitted complete method is the branch-and-bound algorithm [27, 28], which iteratively divides the feasible region of solution space and prunes nodes by the bounds of the objective function. Additionally, many techniques were developed, including the cutting plane method [20] and the domain propagation [40], which are often incorporated into the branch-and-bound process in modern MIP solvers [2]. Almost all state-of-the-art MIP solvers are based on the branch-and-bound framework, including the commercial solvers *Gurobi* [21] and *CPLEX* [36], and the academic solvers *SCIP* [5] and *HiGHS* [23].

Local search is a powerful incomplete method for solving challenging problems across various fields in computer science and operations research [22]. It aims to find good solutions quickly and demonstrates significant effectiveness in solving SAT and MaxSAT [6]. Some local search solvers for special cases of MIP have been proposed, such as pseudo-Boolean optimization [4, 11] and integer linear programming [39, 30]. However, the development of local search solvers for MIP is still in its infancy. As far as we know, there is in lack of efficient free local search solvers for MIP that is available to public communities. *Feasibility Jump* [33] is the most related work to us, which proposed a local search algorithm to solve MIP, and won 1st place in the MIP 2022 Competition.[2] However, it only focuses on finding feasible solutions and ignores the objective function, which does not treat MIP's original optimization purpose and does not show a strong capacity for finding high-quality solutions.

We intend to propose an efficient local search solver for MIP. There are two main challenges to achieving this goal: 1. Enhancing adaptability: to handle MIP, local search must accommodate general constraints and variables rather than specific forms. However, solving such general-form problems is more difficult than specific combinatorial problems since less information about problems could be leveraged. Therefore, it is important to enhance the adaptability of the solver to search states. 2. Balance the optimization and the satisfaction: unlike decision problems such as SAT, MIP considers both the optimization of the objective function and the satisfaction of all constraints. But these two factors are sometimes conflicting, thus finding the balance between them is also an important concern.

In this work, we design a novel local search solver for solving general MIP, synthesizing new operators and scoring functions to handle the above challenges. We propose a mixed tight move operator to satisfy general constraints, and a breakthrough operator to surpass the best-found solution by considering the objective value of the dynamically updated best-found solution. Moreover, we propose a two-level scoring function structure to measure the benefit of candidate solutions, which is also related to the dynamically updated best-found objective

[2] `https://www.mixedinteger.org/2022/competition/`

value. Within each level of the structure, we design the scoring function that synthesizes the score for optimizing the objective function and the score for satisfying constraints. Experimental results demonstrate our solver's excellent performance for solving MIP.

### Contributions

We develop an efficient local search solver to find high-quality feasible solutions for MIP.

Firstly, we propose two new operators to adaptively modify variables to optimize the objective function and satisfy constraints, respectively. The breakthrough move operator is proposed to break through the objective value of the dynamically updated best-found solution, and the mixed tight move operator is proposed to satisfy and tighten constraints.

Then, to efficiently guide the search, we first design a new weighting scheme to balance the priority between the objective function and each constraint during the search process. Based on the weighting scheme, we propose a two-level scoring function structure to measure the benefit of each candidate solution. The first level is the progress score, which measures operations by comparing them with the current solution, aiming to make local progress for the current solution. The second level is the bonus score, containing the breakthrough bonus for improving the objective function and the robustness bonus for satisfying constraints stably, being complementary to the progress score. The breakthrough bonus, working together with the breakthrough move operator, enables our solver to select and modify variables according to the dynamically updated best-found solution, improving its adaptability to the global search state. At each level, the scoring function consider both the optimization of the objective function and the satisfaction of constraints.

By putting these together, we develop a new local search solver for MIP called *Local-MIP*. Experiments conducted on seven public benchmarks show the efficiency of *Local-MIP* in finding high-quality feasible solutions for MIP. We compare *Local-MIP* with the state-of-the-art MIP solvers, including the commercial solvers *Gurobi* [21] and *CPLEX* [36], the academic solvers *SCIP* [5] and *HiGHS* [23], and the local search algorithm *Feasibility Jump* [33]. Experimental results show the excellent performance of *Local-MIP*, which significantly outperforms *CPLEX*, *HiGHS*, *SCIP* and *Feasibility Jump*, and is competitive with the most powerful commercial solver *Gurobi*, indicating a significant improvement in the field of local search solver for MIP. Moreover, *Local-MIP* establishes 4 new records for MIPLIB open instances by finding the new best solutions. Additionally, we analyze the effectiveness of proposed strategies and the stability of *Local-MIP* with different random seeds.

## 2 Preliminaries and Notation

### 2.1 Formulations 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* $I \subseteq N = \{1, ..., n\}$*. Let* $\boldsymbol{x} = \{x_1, x_2, ..., x_n\}$ *be a set of variables. The mixed integer programming is to solve*

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

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 $x_j \in \mathbb{Z}\ for\ all\ j \in I$ the integrality constraints. A general linear constraint $\boldsymbol{A_i}\boldsymbol{x} \leq b_i$ is denoted as $con_i$, and it contains $x_j$ if $A_{ij} \neq 0$. MIP aims to minimize the objective function while satisfying all constraints. A maximization problem and other types of linear constraints can be easily converted into this formulation.

A solution $\boldsymbol{s}$ of a MIP instance is a vector of values assigned for each variable, where $s_j$ denotes the value of $x_j$. 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. $\boldsymbol{s^*}$ denotes the best-found solution in the local search process, and $\boldsymbol{s^{cur}}$ denotes the current solution in each step of 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, the local search typically starts from an initial solution and iteratively modifies the current solution by changing the value of a variable, in order to search for 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 specifying variable to operate, an **operation** is obtained. For example, for Boolean variables, the standard operator is $flip$, which turns the value of a variable to its opposite, and flip($x_1$) is an operation to flip the specifying variable $x_1$. For the current solution, performing an operation could generate a new candidate solution.

During each step of the local search process, scoring functions are used to evaluate different candidate operations for picking one to execute to update the current solution. Given an operation $op$, a scoring function $score(op)$ measures how good $op$ is. An operation $op$ is said **positive** if $score(op) > 0$, which indicates that performing $op$ could improve the quality of the current solution.

In the following two sections, we propose tailored operators and scoring functions to guide the local search process. Section 5 provides a detailed description of our local search solver.

# 3 New Operators Tailored for MIP

In this section, we propose two novel local search operators for MIP: the breakthrough move for optimizing the objective function, and the mixed tight move for satisfying constraints.

## 3.1 Breakthrough Move

To find solutions with high-quality objective values in the local search process, for the first time, we propose the new idea of the breakthrough move operator, which modifies the current solution to improve the quality of the objective function, aiming to break through the objective value of the dynamically updated best-found solution.

▶ **Definition 2.** *Given a variable $x_j$ that appears in the objective function (i.e., $c_j \neq 0$), 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 $x_j$'s bounds satisfied. Precisely, let $\epsilon$ 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 $s_j^{new}$,*

$$s_j^{new} = \begin{cases} min(s_j + \Delta_j, u_j), & if\ c_j < 0, \\ max(\ s_j + \Delta_j, l_j), & else. \end{cases} \tag{2}$$

*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:*

$$s_j^{new} = \begin{cases} min(\lceil s_j + \Delta_j \rceil, \lfloor u_j \rfloor), & if\ c_j < 0, \\ max(\lfloor s_j + \Delta_j \rfloor, \lceil l_j \rceil), & else. \end{cases} \tag{3}$$

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 to break through the dynamically updated best-found objective value is proposed in the local search for solving combinatorial optimization problems.

▶ **Example 3.** 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 the best-found solution is $\boldsymbol{s^*} = \{s_1 = 3, s_2 = 2\}$, thus $obj(\boldsymbol{s^*}) = 5$; the current solution is $\boldsymbol{s^{cur}} = \{s_1 = 4, s_2 = 1.5\}$, thus $obj(\boldsymbol{s^{cur}}) = 9$. As shown in Figure 1, the operation $bm(x_1, \boldsymbol{s^{cur}})$ refers to assigning $x_1$ to the threshold value 2 to make objective value better than $obj(\boldsymbol{s^*})$, and the operation $bm(x_2, \boldsymbol{s^{cur}})$ assigns $x_2$ to its upper bound 3 to optimize the objective function as much as possible while satisfying the variable's bound.

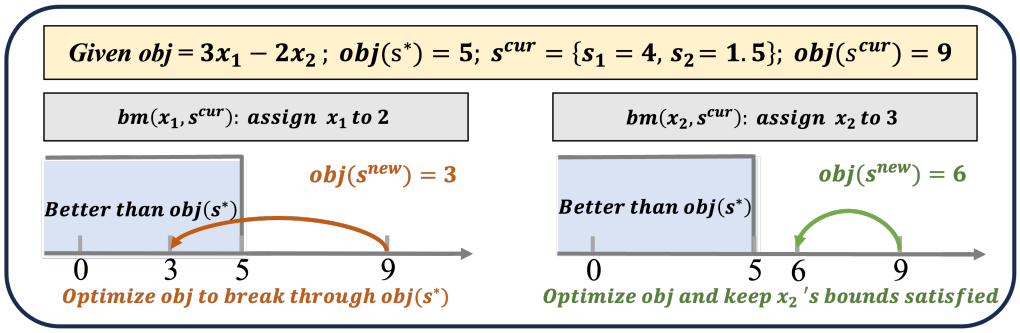



**Figure 1** A graphical explanation of the breakthrough move operator.

## 3.2 Mixed Tight Move

How to adaptively move variables to satisfy constraints is the key technology when applying local search to solve linear systems.

The Simplex algorithm [12] solves linear programming by tightening constraints to locate solutions in extreme points of the polyhedron. Also, tightening constraints show benefits for finding feasible solutions for integer programming [30]. Both linear programming and integer programming are the subclasses of MIP. Here, based on the insight for tightening constraints, we go one step further and propose an operator suitable for MIP, which can handle both real and integer variables, dubbed as the mixed tight move.

▶ **Definition 4.** *Given a variable $x_j$, a constraint $con_i$ containing $x_j$ (i.e., $A_{ij} \neq 0$), 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 $con_i$ satisfied and tight while keeping $x_j$'s bounds 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 $s_j^{new}$,*

$$s_j^{new} = \begin{cases} min(s_j + \Delta_{ij}, u_j), & if\ \Delta_{ij} > 0, \\ max(s_j + \Delta_{ij}, l_j), & if\ \Delta_{ij} < 0, \\ s_j, & else. \end{cases} \tag{4}$$

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

$$s_j^{new} = \begin{cases} min(\lceil s_j + \Delta_{ij} \rceil, \lfloor u_j \rfloor), & if\ b_i - \boldsymbol{A_i} \cdot \boldsymbol{s} < 0\ and\ A_{ij} < 0, \\ max(\lfloor s_j + \Delta_{ij} \rfloor, \lceil l_j \rceil), & if\ b_i - \boldsymbol{A_i} \cdot \boldsymbol{s} < 0\ and\ A_{ij} > 0. \\ max(\lceil s_j + \Delta_{ij} \rceil, \lceil l_j \rceil), & if\ b_i - \boldsymbol{A_i} \cdot \boldsymbol{s} > 0\ and\ A_{ij} < 0, \\ min(\lfloor s_j + \Delta_{ij} \rfloor, \lfloor u_j \rfloor), & if\ b_i - \boldsymbol{A_i} \cdot \boldsymbol{s} > 0\ and\ A_{ij} > 0, \\ s_j, & else. \end{cases} \tag{5}$$

According to the above definition, both violated and satisfied constraints can be handled by the mixed tight move operator. The mixed tight move ensures the global bound of related variables is satisfied. Assuming that there is no global bound for each variable, a mixed tight operation of a violated constraint will satisfy the corresponding constraint by choosing the minimal possible change to a variable, which will have the least impact on other constraints that contain the corresponding variable. For a satisfied constraint, a mixed tight operation assigns a variable to its extreme value while ensuring that the corresponding constraint remains satisfied. This allows the exploration for escaping local optimum, as it takes the maximal change of the corresponding variable to the objective function and other constraints.

## 4 Weighting Scheme and Scoring Functions

As core techniques to guide the local search process, scoring functions measure the benefits of candidate operations for selecting one with the highest score to execute in each step. When solving combinatorial optimization problems, dynamic weighting techniques are commonly used in scoring functions to guide and diversify the search process [29, 11]. In this section, we first design a new weighting scheme for MIP, and then we propose a two-level weighting-based scoring function structure to hierarchically guide search for high-quality feasible solutions.

### 4.1 Weighting Scheme for MIP

Weighting schemes are usually used to adjust the priority of each constraint by diversified weights in the local search process [45, 43]. Weighting schemes typically increase the weights of constraints that are often violated, hence guiding the search process toward satisfying these constraints. When designing a weighting scheme for combinatorial optimization problems, the main challenge is how to balance the weights between the objective function and each constraint in the search process [45]. For example, in solving MaxSAT, excessive weights of soft clauses would make the local search difficult to satisfy all hard clauses, thereby hindering the capability of finding feasible solutions [10].

Here, we design a new weighting scheme for MIP based on the probabilistic version of PAWS scheme [44, 9, 7, 30], which updates weights according to a smoothing probability $sp$, and we set $sp = 0.0003$ as mentioned in [7, 30]. Our weighting scheme aims to dynamically balance the weights of the objective function and each constraint by preventing excessive weight while maintaining the distinction between them. It works as follows:

**(a)** The objective function and each constraint have the attribute of an integral weight, which are denoted as $w(obj)$ and $w(con_i)$, respectively.

**(b)** Initalization: $w(obj) = 1$ and $w(con_i) = 1$.

**(c)** When the search process is trapped in a local optimum (i.e., there is no positive operation to select), the weighting scheme is activated, and weights are updated in one of the ways as follows:

- With probability $1 - sp$, if the current solution is feasible, $w(obj) = w(obj) + 1$; otherwise, for each violated constraint $con_i$, $w(con_i) = w(con_i) + 1$.
- With probability $sp$, if $obj(\boldsymbol{s^{cur}}) < obj(\boldsymbol{s^*})$ and $w(obj) > 0$, $w(obj) = w(obj) - 1$; for each satisfied constraint $con_i$ whose $w(con_i) > 0$, $w(con_i) = w(con_i) - 1$.

The weighting scheme makes the weights of constraints diverse, reflecting different priorities for considering constraints in search directions. By focusing on constraints that are often violated in local optima, the weighting scheme helps the local search process find feasible solutions. Accordingly, if the search process frequently visits feasible solutions in local optima, then the objective function should be prioritized and will be set with a higher weight, which helps to find higher-quality feasible solutions. Besides, the weight of the objective function decreases if the visited solutions are often infeasible but have a better objective value than the best-found solution, which means the satisfaction of constraints should be more addressed.

## 4.2 Two-level Scoring Function Structure

Based on the weighting scheme, we propose a two-level scoring function structure, which contains a base scoring function for the first level and a bonus scoring function for the second level. The base scoring function incorporates basic metrics on improving the quality of the objective function and the number of satisfied constraints. The bonus score evaluates the selected operations from a finer view: it both rewards the breakthrough of the objective function, and the robust satisfaction of constraints.

In evaluating an operation, the base scoring function is applied first to select the best operations under it, and then the bonus scoring function is used to evaluate the operations with the best base scores. The composition of the two-level scoring functions takes into consideration both the basic improvement and the adaptive adjusting with the solving process. Moreover, at each level, the scoring functions both measure the objective value and satisfaction of constraints, which balance the aim of optimization and satisfaction.

### 4.2.1 First Level: Progress Score as the Base Scoring Function

In the first level, we propose the progress score as the base scoring function. The progress score takes a local perspective for improving the quality of the current solution, including the value of the objective function and the satisfaction of each constraint.

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

▶ **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 progress score of op for improving the quality of the objective value, denoted as $score^{obj}_{progress}(op)$,*

$$score^{obj}_{progress}(op) = \begin{cases} w(obj), & if\ obj(\boldsymbol{s}^{op}) < obj(\boldsymbol{s}^{cur}), \\ -w(obj), & if\ obj(\boldsymbol{s}^{op}) > obj(\boldsymbol{s}^{cur}), \\ 0, & else. \end{cases} \tag{6}$$

For constraints, the primary goal of MIP is to make them satisfied, thus the progress score rewards the 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 also rewards the proximity to be satisfied and punishes aggravated violations.

▶ **Definition 6.** *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 $score^{con_i}_{progress}(op)$,*

$$score^{con_i}_{progress}(op) = \begin{cases} w(con_i), & if\ \boldsymbol{A_i} \cdot \boldsymbol{s}^{op} \leq b_i < \boldsymbol{A_i} \cdot \boldsymbol{s}^{cur}, \\ -w(con_i), & if\ \boldsymbol{A_i} \cdot \boldsymbol{s}^{cur} \leq b_i < \boldsymbol{A_i} \cdot \boldsymbol{s}^{op}, \\ w(con_i)/2, & if\ b_i < \boldsymbol{A_i} \cdot \boldsymbol{s}^{op} < \boldsymbol{A_i} \cdot \boldsymbol{s}^{cur}, \\ -w(con_i)/2, & if\ b_i < \boldsymbol{A_i} \cdot \boldsymbol{s}^{cur} < \boldsymbol{A_i} \cdot \boldsymbol{s}^{op}, \\ 0, & else. \end{cases} \tag{7}$$

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

▶ **Definition 7.** *Given an operation op, the progress score of op, denoted as $score_{progress}(op)$,*

$$score_{progress}(op) = score^{obj}_{progress}(op) + \sum_{i=1}^{m} score^{con_i}_{progress}(op) \tag{8}$$

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

### 4.2.2 Second Level: Bonus Scoring Function

The progress score introduced above is employed as the elementary scoring function for selecting the operations with the highest score to be applied. However, the progress score has limited capacity to distinguish better operations, according to our preliminary experiments which execute 10000 steps local search on all testing instances, there are on average %47.5 steps where multiple operations with the same greatest progress score are presented. It is important to further evaluate these operations by designing finer scores, to do tie-breaking and award better operations in some metrics. For a similar purpose, previous local search works often use the *age* information of variables as the secondary scoring function [8], where *age* is defined as the number of steps since the last time it is modified, and the operation of oldest age variable is selected to break tie.

However, the *age* scoring function does not consider the characterizations of MIP and cannot effectively guide the search process for MIP according to the experiment. According to Definition 7, the progress score takes the local perspective to make progress for the current solution. Therefore, to further choose an operation among operations with the same best progress score, we consider global properties for the objective function and each constraint, instead of comparing the candidate solution with the current solution.

We design the bonus scoring function as the second level, denoted as $score_{bonus}$, which contains the breakthrough bonus for the objective function and the robustness bonus for constraints, to make distinctions in operations with the best progress score.

#### Breakthrough Bonus

For the objective function, the bonus scoring function aims to move a variable to break through the global best-found solution, working together with the proposed breakthrough move operator. Thus, we propose the breakthrough bonus to reward the situation that the new candidate solution is better than the dynamically updated best-found solution for the objective value.

▶ **Definition 8.** *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* $bonus_{break}(op)$*,*

$$bonus_{break}(op) = \begin{cases} w(obj), & if\ obj(\boldsymbol{s}^{op}) < obj(\boldsymbol{s}^*), \\ 0, & otherwise. \end{cases} \tag{9}$$

### Robustness Bonus

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

▶ **Definition 9.** *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* $bonus^{con_i}_{robust}(op)$*,*

$$bonus^{con_i}_{robust}(op) = \begin{cases} w(con_i), & if\ \boldsymbol{A_i} \cdot \boldsymbol{s^{op}} < b_i, \\ 0, & otherwise. \end{cases} \tag{10}$$

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

Synthesize 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 10.** *Given an operation op, the bonus score of op, denoted as* $score_{bonus}(op)$*,*

$$score_{bonus}(op) = bonus_{break}(op) + \sum_{i=1}^{m} bonus^{con_i}_{robust}(op) \tag{11}$$

Complementary to the progress score that compares new candidate solutions with the current solution from a local perspective, the breakthrough bonus considers information from a global perspective to compare and try to break through the global best-known solution, thus extending the selection and modification of variables according to the global search performances. Moreover, the robustness bonus score makes distinctions of inequalities, providing a finer evaluation of the satisfaction of constraints.

Consequently, we combine the progress score and the bonus score in a two-level manner to hierarchically evaluate operations in our solver. This design considers both the local improvement and global improvement for operations and balances the objective optimizing and constraints satisfaction, enhancing the adaptability and efficiency of our algorithm.

## 5 The Local-MIP Algorithm

Based on the ideas proposed in previous sections, we develop a local search solver for solving MIP called *Local-MIP*. The pseudo-code of *Local-MIP* is outlined in Algorithm 1.

**Algorithm 1** The Local-MIP Algorithm.

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

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

We use $\boldsymbol{s}^{\boldsymbol{cur}}$ to denote the current solution which is maintained during the search process, while $\boldsymbol{s}^*$ and $obj(\boldsymbol{s}^*)$ denote the best-found solution and its objective value. In the beginning, each variable of $\boldsymbol{s}^{\boldsymbol{cur}}$ is initialized as the value closest to 0 within its global bounds (Line 1). $\boldsymbol{s}^*$ is initialized as an empty set, and $obj(\boldsymbol{s}^*)$ is initialized as $+\infty$ (Line 2).

After initialization, *Local-MIP* conducts the search process until a given time limit *cutoff* is reached (Lines 3–10). In each step of local search, the best-found solution and the corresponding objective value are updated once a new better feasible solution is discovered (Lines 4-7). The lift move process was proposed in [30] to improve the objective value while maintaining feasibility for a feasible solution of integer programming, a subclass of MIP, which can be easily applied for a feasible solution of MIP, and thus we employ the process once a feasible solution is found in the search process (Line 6). The lift move process improves the quality of the objective function via the best local feasible domain derived by the local domain reduction, until reaches the local optimum [30].

The algorithm generates neighborhood solutions through candidate operations by calling the function $Get\_Candidate\_Operations(Q, \boldsymbol{s}^{\boldsymbol{cur}})$ (Line 8), which is detailedly described in Algorithm 2. Afterward, *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 of the bonus score function is applied and an operation with the greatest bonus score is selected and performed (Line 9-10), to modify the current solution to get a new $\boldsymbol{s}^{\boldsymbol{cur}}$ (Line 11).

### Candidate Operations in Each Step

The candidate solutions in each step are constructed by Algorithm 2, containing the breakthrough move operation and the mixed tight move operation proposed in Section 3. 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 under 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, since they perhaps reduce the satisfaction of other constraints.

**Algorithm 2** Get_Candidate_Operations.

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

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

Once any feasible solution has been found, the goal of the search process is transformed into discovering feasible solutions with high-quality objective value. Therefore, the primal candidate operations are the union of positive $\boldsymbol{bm}$ operations and positive $\boldsymbol{mtm}$ 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 $\boldsymbol{mtm}$ operations in satisfied constraints (Lines 7-8).

If the algorithm fails to find any positive operation in the previous process, it tries to search for positive Boolean flip operations as candidate operations (Lines 9-11), which we will describe later. Once the above exploration fails, it is indicated that local search falls into the local optimum (Line 12). It first activates the weighting scheme to update the weights of objective function and constraints (Line 13). Afterward, it randomly selects a violated constraint if one exits, and then generates candidate operations set by the union of $\boldsymbol{bm}$ operations and $\boldsymbol{mtm}$ operations in the selected constraint (Line 14).

Note that we use the Boolean flip operation in Lines 9-11 to flip the values of Boolean variables. The motivation is due to the importance of Boolean variables in MIP modeling (involved in 96.3% of the MIPLIB instances) and the feature of the flip operation that it could generate more operations than the above operators for Boolean candidate variables. Thus we adopt the flip operator as a special treatment for Boolean variables and apply it to complement the proposed operators to generate more operations.

Besides the main part as shown in Algorithm 1 and 2, we also introduce a forbidding strategy to further improve efficiency. Local search methods may often stuck in suboptimal regions. To address the cycling phenomenon of revisiting the same regions, we employ a forbidding strategy, the tabu strategy [19, 7]. The tabu strategy is directly applied to *Local-MIP*. Once a variable is modified, it forbids the modification for the reverse direction in the following $tt$ iterations, where $tt$ is called tabu tenure, and we set $tt = 3 + rand(10)$ as mentioned in [7].

# 6 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 4 open instances in the MIPLIB dataset. Additionally, we analyze the effectiveness of proposed strategies and the stability of *Local-MIP* with different random seeds.

## 6.1 Experiment Preliminaries

### 6.1.1 Benchmarks

Our experiments are carried out with 4 benchmarks of the mainstream dataset for MIP, i.e., MIPLIB [18] with hard[3] and open[4] instances. In each instance of MIPLIB, at least one variable is an integer variable. Depending on the type of variables, the MIPLIB instances are classified into the 4 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 variable and real variable both exist and at least one of integer variables is not a Boolean variable.

Additionally, two practical problems are tested: the bin packing and the scheduling problem. They are challenging combinatorial optimization problems, and also have significant applications in real-world industry. We evaluate solvers on 1 standard bin packing benchmark for [15], and 2 standard scheduling benchmarks provided by Taillard's instances [42].

- **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 [14].
- **JSP**: the Job-shop Scheduling problem. This benchmark consists of 80 instances encoded by the modeling method proposed in [26].
- **OSP**: the Open-shop Scheduling problem. This benchmark consists of 60 instances encoded by the modeling method proposed in [35].

### 6.1.2 State-of-the-art Competitors

In Section 6.2, we compare *Local-MIP* with 5 state-of-the-art MIP solvers.

- ***HiGHS*** [23]: an academic solver for large-scale sparse MIP (version 1.6.0).[5]
- ***SCIP*** [5]: one of the fastest academic solvers for MIP (Version 8.1.0, using SoPlex 6.0.4).[6]
- ***Gurobi*** [21]: the most powerful commercial MIP solvers (version 11.0.0). We use both its complete and heuristic versions, denoted by $\text{Gurobi}_{\text{comp}}$ and $\text{Gurobi}_{\text{heur}}$, respectively.[7]
- ***CPLEX*** [36]: a famous commercial MIP solver to solve complex models (version 22.1.0).[8]
- ***Feasibility Jump*** [33]: *FJ* for short, the state-of-the-art local search MIP algorithm, which won 1st place in MIP 2022 Computational Competition.[9]

[3] https://miplib.zib.de/downloads/hard-v22.test
[4] https://miplib.zib.de/downloads/open-v22.test
[5] https://github.com/ERGO-Code/HiGHS
[6] https://www.scipopt.org
[7] https://www.gurobi.com/solutions/gurobi-optimizer/
[8] https://www.ibm.com/products/ilog-cplex-optimization-studio
[9] https://github.com/sintef/feasibilityjump

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

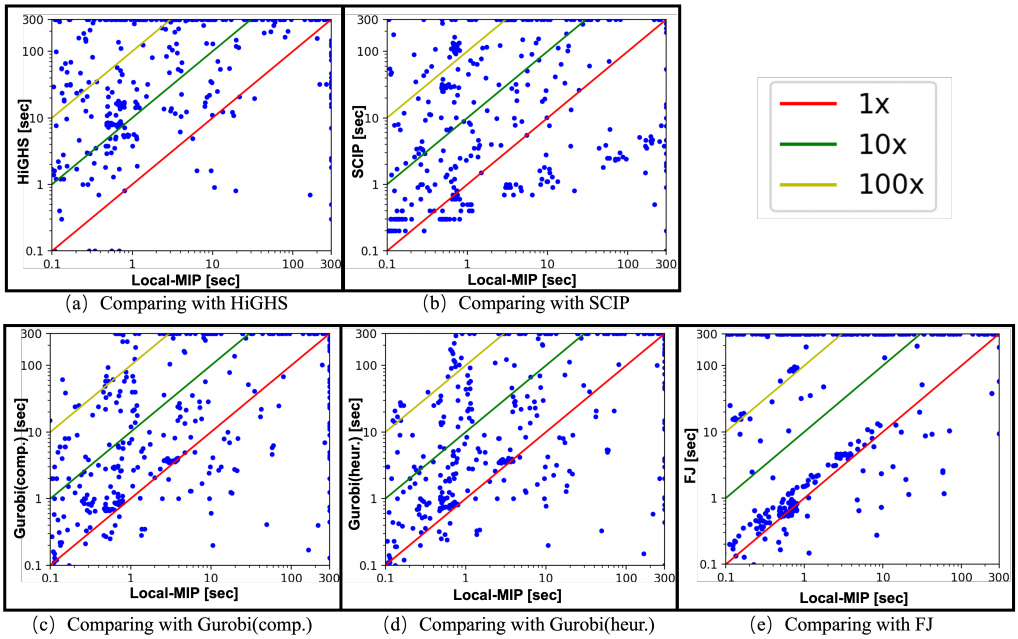
### 6.1.3 Experiment Setup

*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 where 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*, a larger metric value on a benchmark indicates better performance on the corresponding benchmark. For each instance, each solver is executed by one thread with time limits of 10, 60, and 300 seconds as mentioned in [30]. For each time limit setting in the table, the best performance for the corresponding metric is highlighted in **bold**. Additionally, the number of instances in each benchmark is denoted by *#Inst*. Detailed results and the sourced code are made publicly available on GitHub.[10]

## 6.2 Comparison with State-of-the-art MIP Solvers




(a) Comparing with HiGHS (b) Comparing with SCIP

(c) Comparing with Gurobi(comp.) (d) Comparing with Gurobi(heur.) (e) Comparing with FJ

**Figure 2** Run time comparison on each instance for finding the first feasible solution. Note that the comparison with CPLEX is absent, as it do not provide the exact time for finding solutions.

[10] https://github.com/shaowei-cai-group/Local-MIP/

**Table 1** Empirical results on comparing *Local-MIP* with the state-of-the-art MIP solvers in terms of *#Feas* and *#Win* within each given time limit.

| Benchmark | *#Inst* | *HiGHS* | | *SCIP* | | *CPLEX* | | $\text{Gurobi}_{\text{comp}}$ | | $\text{Gurobi}_{\text{heur}}$ | | *FJ* | | *Local-MIP* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *#Feas* | *#Win* | *#Feas* | *#Win* | *#Feas* | *#Win* | *#Feas* | *#Win* | *#Feas* | *#Win* | *#Feas* | *#Win* | *#Feas* | *#Win* |
| time limit 10 seconds | | | | | | | | | | | | | | | |
| MIPLIB-BP | 66 | 6 | 0 | 29 | 2 | 42 | 11 | **44** | 9 | **44** | 7 | 42 | 4 | **44** | **31** |
| MIPLIB-IP | 32 | 7 | 0 | 11 | 2 | **17** | 4 | **17** | **8** | **17** | **8** | 11 | 1 | **17** | 7 |
| MIPLIB-MBP | 195 | 57 | 1 | 80 | 7 | 116 | 31 | 117 | 43 | **119** | **51** | 56 | 10 | 103 | 35 |
| MIPLIB-MIP | 62 | 9 | 2 | 21 | 0 | 32 | 10 | **37** | 11 | **37** | 13 | 18 | 4 | 35 | **15** |
| BPP | 60 | 9 | 0 | 0 | 0 | **60** | 0 | **60** | 0 | **60** | 0 | **60** | 0 | **60** | **60** |
| JSP | 80 | 22 | 0 | **70** | 25 | 31 | 0 | 10 | 1 | 12 | 8 | 0 | 0 | 45 | **36** |
| OSP | 60 | 48 | 22 | **60** | 20 | 28 | 7 | 47 | 27 | 42 | 25 | 1 | 0 | **60** | **45** |
| Total | 555 | 158 | 25 | 271 | 56 | 326 | 63 | 332 | 99 | 331 | 112 | 188 | 19 | **364** | **229** |
| time limit 60 seconds | | | | | | | | | | | | | | | |
| MIPLIB-BP | 66 | 14 | 0 | 35 | 2 | 43 | 8 | 46 | 10 | 47 | 15 | **49** | 2 | 48 | **28** |
| MIPLIB-IP | 32 | 12 | 1 | 14 | 1 | 20 | 6 | 20 | 7 | 20 | **9** | 12 | 1 | **21** | 6 |
| MIPLIB-MBP | 195 | 96 | 6 | 109 | 2 | 129 | 32 | **137** | 49 | 134 | **65** | 62 | 9 | 119 | 23 |
| MIPLIB-MIP | 62 | 15 | 3 | 28 | 1 | 36 | 7 | 41 | 8 | 41 | **18** | 20 | 3 | **43** | 17 |
| BPP | 60 | 40 | 0 | 20 | 0 | **60** | 11 | **60** | 13 | **60** | 15 | **60** | 0 | **60** | **33** |
| JSP | 80 | 41 | 0 | **70** | 15 | 52 | 1 | 23 | 3 | 26 | 13 | 1 | 0 | 54 | **38** |
| OSP | 60 | 58 | 27 | **60** | 20 | 30 | 10 | 53 | 37 | 51 | 31 | 9 | 0 | **60** | **42** |
| Total | 555 | 276 | 37 | 336 | 41 | 370 | 75 | 380 | 127 | 379 | 166 | 213 | 15 | **405** | **187** |
| time limit 300 seconds | | | | | | | | | | | | | | | |
| MIPLIB-BP | 66 | 22 | 1 | 42 | 4 | 43 | 6 | 47 | 10 | 48 | **23** | **49** | 0 | **49** | 17 |
| MIPLIB-IP | 32 | 14 | 2 | 17 | 2 | 21 | 5 | 21 | 10 | **22** | **14** | 12 | 1 | **22** | 4 |
| MIPLIB-MBP | 195 | 115 | 7 | 122 | 7 | 137 | 22 | 150 | 59 | **152** | **69** | 67 | 11 | 123 | 14 |
| MIPLIB-MIP | 62 | 24 | 1 | 34 | 1 | 38 | 7 | 44 | 10 | 43 | **23** | 21 | 3 | **45** | 16 |
| BPP | 60 | 47 | 0 | 40 | 0 | **60** | **31** | **60** | 30 | **60** | 13 | **60** | 0 | **60** | 3 |
| JSP | 80 | 49 | 0 | **70** | 0 | 68 | 1 | 36 | 10 | 34 | 20 | 1 | 0 | **70** | **41** |
| OSP | 60 | **60** | 38 | **60** | 24 | 33 | 13 | 55 | 40 | **60** | 43 | 19 | 0 | **60** | **44** |
| Total | 555 | 331 | 49 | 385 | 38 | 400 | 85 | 413 | 169 | 419 | **205** | 229 | 15 | **429** | 139 |

The results of comparison with 4 state-of-the-art MIP solvers are shown in Tables 1.

**The ability to find feasible solutions (*#Feas*).** *Local-MIP* performs best on 4 benchmarks in the 10s and 60s time limits, and 5 benchmarks in the 300s. For all benchmarks, *Local-MIP* performs best on most benchmarks in the 60s and 300s time limits, and the second most in the 10s. In terms of the total instances of all benchmarks, *Local-MIP* establishes the best performance for all the time limits. In general, this result confirms the powerful ability of *Local-MIP* to find a feasible solution within reasonable times.

**The ability to find high-quality feasible solutions (*#Win*).** *Local-MIP* performs best on 5 benchmarks in the 10s time limit, 4 benchmarks in 60s, and 2 benchmarks in 300s. For all benchmarks, *Local-MIP* performs best on most benchmarks in the 10s and 60s, and the second most in the 300s. Particularly, *Local-MIP* exhibits the best performance for JSP and OSP benchmarks on all time limits, indicating the advantages of our solver in solving scheduling problems. In terms of total instances, *Local-MIP* consistently establishes the best performance for 10s and 60s, but in 300s, *Gurobi* wins more instances than *Local-MIP*, especially its heuristic version. Overall, for the ability to find high-quality feasible solutions, *Local-MIP* significantly outperforms the two academic MIP solvers *HiGHS* and *SCIP*, and another local search solver *Feasibility Jump*. Moreover, *Local-MIP* performs better than the commercial MIP solver *CPLEX*, and is competitive with the most powerful solver *Gurobi*.

According to Table 1, *Local-MIP* outperforms *Feasibility Jump* on almost all settings, indicating a significant improvement in the field of local search solver for MIP.

Furthermore, the run time comparison on all instances for finding the first feasible solution is presented in Figure 2. For comparison with each solver, there are obviously more instances above the red line, which confirms the powerful solving ability of *Local-MIP*.

## 6.3 New Records to Open Instances

In the MIPLIB dataset, the instances labeled as open are those that the optimal solution has not yet been solved. The current best solutions known for each open instance are available on the corresponding page on the MIPLIB website. These open instances are representative of the hard-solving problems.

**Table 2** *Local-MIP* establishes new records to 4 open instances. #var and #cons denote the number of variables and constraints of the corresponding instance, respectively.

| Instance name | #var | #cons | constraint types | Previous best | *Local-MIP* |
|---|---|---|---|---|---|
| genus-sym-g31-8 | 3484 | 32073 | knapsack, precedence, etc. | -21 | -23 |
| genus-sym-g62-2 | 12912 | 78472 | set partitioning, set covering, etc. | -34 | -38 |
| genus-g61-25 | 14380 | 94735 | cardinality, general linear, etc. | -34 | -40 |
| neos-4232544-orira | 87060 | 180600 | aggregations, variable bound, etc. | 17540506.0 | 15108527.512195 |

Excitingly, *Local-MIP* established the new best-known solutions for 4 open instances. The new records have been submitted to MIPLIB 2017 and have been accepted; the links to the website are denoted in the footnotes.[11] [12] [13] [14] As shown in Table 2, each of these 4 instances contains multiple different constraint types,[15] simultaneously indicating the powerful solving ability and its extensive applicability.

## 6.4 Analysis on the Proposed Ideas

**Table 3** Comparison bettewn *Local-MIP* and its modified versions. *#better* and *#worse* 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *#better* | *#worse* | *#better* | *#worse* | *#better* | *#worse* | *#better* | *#worse* | *#better* | *#worse* | *#better* | *#worse* |
| | | Comparison with $V_{no-bm}$ | | | | | | Comparison with $V_{no-weight}$ | | | | | |
| 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 |

[11] https://miplib.zib.de/instance_details_genus-sym-g31-8.html
[12] https://miplib.zib.de/instance_details_genus-sym-g62-2.html
[13] https://miplib.zib.de/instance_details_genus-g61-25.html
[14] https://miplib.zib.de/instance_details_neos-4232544-orira.html
[15] https://miplib.zib.de/statistics.html

To verify the effectiveness of the proposed strategies, we conduct comparative experiments on 4 alternative versions of *Local-MIP*, which are obtained as follows.

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

As shown in Table 3, *Local-MIP* significantly outperforms other variations in almost all settings, confirming the effectiveness of the proposed strategies.

## 6.5 Stability with Repetitive Experiments

To examine the stability of *Local-MIP* which involves randomness, we run *Local-MIP* 10 times with seeds ranging from 1 to 10, and measure the coefficient of variation [1, 11].

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 of each instance is $STD/AVG$, and the lower value indicates greater stability. The experimental results are presented in Table 4, where over 85% have a coefficient of variation less than 0.1, indicating *Local-MIP* exhibits stable performance.

**Table 4** 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 |

# 7 Conclusions and Future Work

In this paper, for solving MIP, we proposed two operators, a weighting scheme, and a tow-level scoring function structure. Based on these novel strategies, we developed an efficient local search solver for MIP. Experimental results demonstrate our solver's excellent performance for solving MIP. Moreover, we establish 4 new records for MIPLIB open instances by finding new best solutions.

For future work, we would like to develop more sophisticated operators and scoring functions to improve the performance of the local search solver for MIP.

## References


1 Hervé Abdi. Coefficient of variation. *Encyclopedia of research design*, 1(5), 2010.

2 Tobias Achterberg. SCIP: solving constraint integer programs. *Math. Program. Comput.*, 1(1):1–41, 2009. doi:10.1007/S12532-008-0001-1.

3 Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. In *Facets of combinatorial optimization: Festschrift for martin grötschel*, pages 449–481. Springer, 2013.

4 VL Beresnev, EN Goncharov, and AA Mel'nikov. Local search with a generalized neighborhood in the optimization problem for pseudo-boolean functions. *Journal of Applied and Industrial Mathematics*, 6:22–30, 2012.

5 Ksenia Bestuzheva, Mathieu Besanc$_c$on, Weikun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros M. Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco E. Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. Enabling research through the SCIP optimization suite 8.0. *ACM Trans. Math. Softw.*, 49(2):22:1–22:21, 2023. doi:10.1145/3585516.

6 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

7 Shaowei Cai, Bohan Li, and Xindi Zhang. Local search for SMT on linear integer arithmetic. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II*, volume 13372 of *Lecture Notes in Computer Science*, pages 227–248. Springer, 2022. doi:10.1007/978-3-031-13188-2_12.

8 Shaowei Cai, Chuan Luo, and Kaile Su. Scoring functions based on second level score for k-sat with long clauses. *J. Artif. Intell. Res.*, 51:413–441, 2014. doi:10.1613/JAIR.4480.

9 Shaowei Cai and Kaile Su. Local search for boolean satisfiability with configuration checking and subscore. *Artif. Intell.*, 204:75–98, 2013. doi:10.1016/J.ARTINT.2013.09.001.

10 Byungki Cha, Kazuo Iwama, Yahiko Kambayashi, and Shuichi Miyazaki. Local search algorithms for partial MAXSAT. In Benjamin Kuipers and Bonnie L. Webber, editors, *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, USA*, pages 263–268. AAAI Press / The MIT Press, 1997. URL: http://www.aaai.org/Library/AAAI/1997/aaai97-041.php.

11 Yi Chu, Shaowei Cai, Chuan Luo, Zhendong Lei, and Cong Peng. Towards more efficient local search for pseudo-boolean optimization. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming, CP 2023, August 27-31, 2023, Toronto, Canada*, volume 280 of *LIPIcs*, pages 12:1–12:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CP.2023.12.

12 George B Dantzig, Alex Orden, Philip Wolfe, et al. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5(2):183–195, 1955.

13 Jessica Davies and Fahiem Bacchus. Exploiting the power of mip solvers in maxsat. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013. doi:10.1007/978-3-642-39071-5_13.

14 Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *Eur. J. Oper. Res.*, 255(1):1–20, 2016. doi:10.1016/J.EJOR.2016.04.030.

15 Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *J. Heuristics*, 2(1):5–30, 1996. doi:10.1007/BF00226291.

16 Christodoulos A. Floudas and Xiaoxia Lin. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Ann. Oper. Res.*, 139(1):131–162, 2005. doi:10.1007/S10479-005-3446-X.

17 Sethavidh Gertphol and Viktor K. Prasanna. MIP formulation for robust resource allocation in dynamic real-time systems. *J. Syst. Softw.*, 77(1):55–65, 2005. doi:10.1016/J.JSS.2003.12.040.

18 Ambros M. Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff T. Linderoth, Marco E. Lübbecke, Hans D. Mittelmann, Derya B. Özyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. *Math. Program. Comput.*, 13(3):443–490, 2021. doi:10.1007/S12532-020-00194-3.

19 Fred Glover and Manuel Laguna. *Tabu search*. Springer, 1998.

20 Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs *and* an algorithm for the mixed integer problem. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 77–103. Springer, 2010. doi:10.1007/978-3-540-68279-0_4.

21 LLC Gurobi Optimization. Gurobi optimizer ref. manual, 2024.

22 Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.

23 Qi Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Math. Program. Comput.*, 10(1):119–142, 2018. doi:10.1007/S12532-017-0130-5.

24 Ravindran Kannan and Clyde 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*, pages 161–172. Springer, 1978.

25 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *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*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

26 Wen-Yang Ku and J. Christopher Beck. Mixed integer programming models for job shop scheduling: A computational analysis. *Comput. Oper. Res.*, 73:165–173, 2016. doi:10.1016/J.COR.2016.04.006.

27 Ailsa H. Land and Alison G. Doig. An automatic method for solving discrete programming problems. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 105–132. Springer, 2010. doi:10.1007/978-3-540-68279-0_5.

28 Eugene L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Oper. Res.*, 14(4):699–719, 1966. doi:10.1287/OPRE.14.4.699.

29 Zhendong Lei and Shaowei Cai. Solving (weighted) partial maxsat by dynamic local search for SAT. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1346–1352. ijcai.org, 2018. doi:10.24963/IJCAI.2018/187.

30 Peng Lin, Shaowei Cai, Mengchuan Zou, and Jinkun Lin. New characterizations and efficient local search for general integer linear programming. *arXiv preprint arXiv:2305.00188*, 2023. arXiv:2305.00188.

31 Peng Lin, Mengchuan Zou, and Shaowei Cai. Local-MIP. Software, version 1.0., swhId: swh:1:dir:883191ffb9b4503105cce3e9d3da6d50421956f3 (visited on 2024-08-19). URL: https://github.com/shaowei-cai-group/Local-MIP.

32 Andrea Lodi and Michele Monaci. Integer linear programming models for 2-staged two-dimensional knapsack problems. *Math. Program.*, 94(2-3):257–278, 2003. doi:10.1007/S10107-002-0319-9.

**33** Bjørnar Luteberget and Giorgio Sartor. Feasibility jump: an lp-free lagrangian MIP heuristic. *Math. Program. Comput.*, 15(2):365–388, 2023. doi:10.1007/S12532-023-00234-8.

**34** Rafael A. Melo and Celso C. Ribeiro. MIP formulations for induced graph optimization problems: a tutorial. *Int. Trans. Oper. Res.*, 30(6):3159–3200, 2023. doi:10.1111/ITOR.13299.

**35** B Naderi and M Zandieh. Modeling and scheduling no-wait open shop problems. *International Journal of Production Economics*, 158:256–266, 2014.

**36** Stefan Nickel, Claudius Steinhardt, Hans Schlenker, and Wolfgang 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)*, pages 9–21. Springer, 2022.

**37** Gábor Pataki. Teaching integer programming formulations using the traveling salesman problem. *SIAM Rev.*, 45(1):116–123, 2003. doi:10.1137/S00361445023685.

**38** Yves Pochet and Laurence A Wolsey. *Production planning by mixed integer programming*, volume 149. Springer, 2006.

**39** SD Prestwich and 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.

**40** Martin W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *INFORMS J. Comput.*, 6(4):445–454, 1994. doi:10.1287/IJOC.6.4.445.

**41** J Cole Smith and Z Caner Taskin. A tutorial guide to mixed-integer programming models and solution techniques. *Optimization in medicine and biology*, pages 521–548, 2008.

**42** Eric Taillard. Benchmarks for basic scheduling problems. *european journal of operational research*, 64(2):278–285, 1993.

**43** John Thornton. Clause weighting local search for SAT. *J. Autom. Reason.*, 35(1-3):97–142, 2005. doi:10.1007/S10817-005-9010-1.

**44** John Thornton, Duc Nghia Pham, Stuart Bain, and Valnir Ferreira Jr. Additive versus multiplicative clause weighting for SAT. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pages 191–196. AAAI Press / The MIT Press, 2004. URL: http://www.aaai.org/Library/AAAI/2004/aaai04-031.php.

**45** John Thornton and Abdul Sattar. Dynamic constraint weighting for over-constrained problems. In Hing-Yan Lee and Hiroshi Motoda, editors, *PRICAI'98, Topics in Artificial Intelligence, 5th Pacific Rim International Conference on Artificial Intelligence, Singapore, November 22-27, 1998, Proceedings*, volume 1531 of *Lecture Notes in Computer Science*, pages 377–388. Springer, 1998. doi:10.1007/BFB0095285.

**46** Laurence A. Wolsey. Mixed integer programming. In Benjamin W. Wah, editor, *Wiley Encyclopedia of Computer Science and Engineering*. John Wiley & Sons, Inc., 2008. doi:10.1002/9780470050118.ECSE244.