

BALANS: Multi-Armed Bandits-based Adaptive Large Neighborhood Search for Mixed-Integer Programming Problems

Junyang Cai¹, Serdar Kadioğlu^{2,3}, Bistra Dilkina¹

¹Department of Computer Science, University of Southern California

²AI Center of Excellence, Fidelity Investments

³Department of Computer Science, Brown University
{caijunya, dilkina}@usc.edu, serdark@cs.brown.edu

Abstract

Mixed-integer programming (MIP) is a powerful paradigm for modeling and solving various important combinatorial optimization problems. Recently, learning-based approaches have shown a potential to speed up MIP solving via offline training that then guides important design decisions during the search. However, a significant drawback of these methods is their heavy reliance on offline training, which requires collecting training datasets and computationally costly training epochs yet offering only limited generalization to unseen (larger) instances. In this paper, we propose BALANS, an adaptive meta-solver for MIPs with online learning capability that does not require any supervision or *a priori* training. At its core, BALANS is based on adaptive large-neighborhood search, operating on top of an MIP solver by successive applications of destroy and repair neighborhood operators. During the search, the selection among different neighborhood definitions is guided on the fly for the instance at hand via multi-armed bandit algorithms. Our extensive experiments on hard optimization instances show that BALANS offers significant performance gains over the default MIP solver, is better than committing to any single best neighborhood, and improves over the state-of-the-art large-neighborhood search for MIPs. Finally, we release BALANS as a highly configurable, MIP solver agnostic, open-source software.

1 Introduction

Mixed-integer programming (MIP) can model many important combinatorial optimization problems, and hence, improving the efficiency of MIP solving is of great practical and theoretical interest. Broadly, solving MIPs falls under two main classes of algorithms: exact and meta-heuristic methods. For exact methods, branch-and-bound [Land and Doig, 2010], and its extensions, are at the core of solving MIPs to optimality. When proving optimality is beyond reach, meta-heuristics, such as Large Neighborhood Search (LNS), offer an attractive alternative for finding good solutions and increasing scalability.

Incorporating learning-based methods into MIP solving has shown great potential to improve exact and heuristic methods. For exact branch-and-bound based methods, these include learning to branch [Bengio *et al.*, 2021; Khalil *et al.*, 2016; Lodi and Zarpellon, 2017; Kadioglu *et al.*, 2012; Cai *et al.*, 2024a], and node selection [He *et al.*, 2014]. For meta-heuristics, these include predicting and searching for solutions [Ding *et al.*, 2020; Cai *et al.*, 2024b], integrating machine learning into LNS on top of MIP solvers [Song *et al.*, 2020; Huang *et al.*, 2023b], and learning to schedule heuristics within the branch-and-bound search tree [Khalil *et al.*, 2017; Chmiela *et al.*, 2021; Hendel, 2022].

The significant drawback of these learning-guided methods is their heavy dependency on offline training. Training is computationally costly, depends on carefully curating training datasets with desired properties and distributions, and has limited generalization. Moreover, training might even depend on using exact solvers in the first place to create the supervised datasets, which defeats the purpose of improving solving for very hard instances. Adapting offline learning-based methods to new distributions and domains remains a challenge, and hence, on-the-fly or online learning approaches to MIP solving are a much-needed alternative. This is exactly what we study in this paper.

In this paper, we focus on solving MIPs via online adaptive methods that do not depend on any offline training. Recent research has shown the potential of meta-solvers that embed a MIP solver within LNS [Song *et al.*, 2020; Tong *et al.*, 2024] and consequently achieve better performance than default MIP via LNS(MIP). The state-of-the-art in this line of research is based on a single neighborhood definition, referred to as local branching relaxation [Huang *et al.*, 2023a]. However, the previous work showing that offline learning can improve LNS [Song *et al.*, 2020; Huang *et al.*, 2023b] suffers from the above-mentioned limitations. We take the idea of machine learning-guided LNS for MIP further and propose a meta-solver based on *Adaptive LNS* (ALNS) operating on a MIP solver, i.e., ALNS(MIP), composed of a diverse set of neighborhood definitions that are driven by multi-armed bandits online learning policy.

This is, however, non-trivial since online learning is hard, but given the significant drawbacks of offline learning, it must be tackled on the fly, adaptively for the specific instance at hand. We show how to cast this as a *multi-armed bandit*

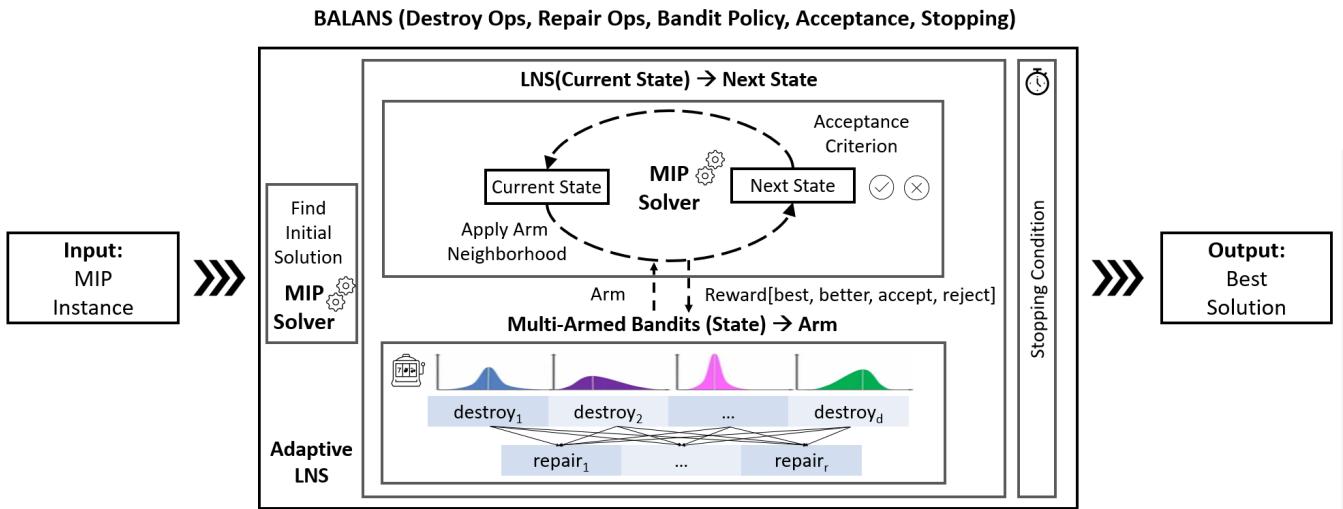


Figure 1: The high-level architecture of the BALANS Solver with its configurable components as input parameters.

problem that treats adaptive neighborhoods as different arm choices with unknown reward distributions to be estimated during the search. Bringing these components together, we propose BALANS, an online meta-solver for MIPs using multi-armed Bandits-based Adaptive Large Neighborhood Search operating across a diverse set of neighborhoods.

Our main findings and contributions are as follows:

- We show that the performance of our bandit-based ALNS(MIP), carefully implemented in our BALANS solver, significantly improves the default MIP solver SCIP [Bulusani *et al.*, 2024], outperforms LNS(MIP) that commits to any single neighborhood, and improves over the previous state-of-the-art LNS(MIP) approach on hard instances of optimization problems from different application domains.
- We show that our bandit-based ALNS(MIP) rarely depends on the single best neighborhood and instead improves the performance by exploring and sequencing other weaker neighborhoods.
- We perform ablation studies to show i) BALANS is solver agnostic by performing the same set of experiments on a different MIP solver Gurobi [Gurobi, 2024], and ii), BALANS is highly different from scheduling heuristics within the BnB tree [Hendel, 2022].
- Finally, we release BALANS¹ as an open-source meta-solver for conducting ALNS(MIP) available to other researchers and practitioners with a one-liner from PyPI².

More broadly, our BALANS solver subsumes the previous literature on LNS(MIP) when run with a single neighborhood while serving as a highly configurable, modular, and extensible integration technology at the intersection of adaptive search, meta-heuristics, multi-armed bandits, and mixed-integer programming.

¹<https://github.com/skadio/balans>

²pip install balans

In the following, let us start with a brief background and present our approach, followed by experimental results, discussions, and related work.

2 Background

Mixed-Integer Programming (MIP): We deal with MIP solving that formulates optimization problems of the form:

$$f(x) = \min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \forall j \in I\}$$

where $f(x)$ is the objective function, and $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n$, and the non-empty set $I \subseteq 1, \dots, n$ indexes the integer variables.

The Linear Programming (LP) relaxation of a MIP x_{lp} is obtained by relaxing integer variables to continuous variables, i.e., by replacing the integer constraint $x_j \in \mathbb{Z} \forall j \in I$ to $x_j \in \mathbb{R} \forall j \in I$. The LP relaxation is an essential part of the branch-and-bound (BnB) algorithm, and many destroy operators are used in large-neighborhood searches.

(Adaptive) Large-Neighborhood Search (LNS / ALNS): LNS is a meta-heuristic that starts with an initial solution and then iteratively destroys and repairs a part of the solution until hitting a stopping condition [Pisinger and Ropke, 2019]. When applied to MIPs, LNS(MIP), we first generate an initial solution, typically found by running BnB for a short time. This becomes the starting point of LNS. In iteration $t \geq 1$, given the solution of the previous state, x_{t-1} , a sub-MIP is created by a destroy heuristic. The re-optimized solution to this sub-MIP, x_t , then becomes the candidate for the next state, decided by an accepted criterion. Adaptive LNS (ALNS) is defined by multiple destroy heuristics to choose from at each iteration, thereby extending LNS(MIP) to ALNS(MIP).

Multi-Armed Bandits (MAB): MAB algorithms [Vermorel and Mohri, 2005] is a class of reinforcement learning aimed at solving online sequential decision-making problems. In MAB, each *arm* defines a decision that an agent can

make, generating either a deterministic or stochastic *reward*. At each step, the agent faces a decision whether to utilize an arm with the highest expected reward (“exploit”) or to try out new arms to learn something new (“explore”). The agent’s goal is to maximize the cumulative reward in the long run, for which it must balance exploration and exploitation. The reward of each arm is estimated from past decisions using a learning policy [Kuleshov and Precup, 2014].

3 BALANS: Online Meta-Solver for MIPs

As illustrated in Figure 1, our BALANS approach brings the complementary strengths of MIP, ALNS, and MAB together to create an online meta-solver for tackling hard combinatorial optimization problems.

Given an MIP instance as input, we first find an initial solution by running an MIP solver with a time limit to find the first feasible solution. This also allows filtering easy-to-solve instances up front. As part of this process, we instantiate a *single* underlying MIP model that we maintain throughout the entire search. Similarly, the LP relaxation at the root node is saved together with other information, such as the indexes of binary, discrete, and continuous variables. This incremental nature of BALANS solver is important for good performance, as shown later in our experiments.

The initial solution yields the current search state to start operating ALNS. As depicted in Figure 1, ALNS is a combination of LNS guided by MAB to adapt to diverse neighborhoods. For this purpose, we designed a diverse set of destroy operators and a single repair operator. The repair operator uses the MIP solver to re-optimize the instance resulting from the destroy operation. Next, we overview destroy operators.

3.1 Destroy Operators

We design eight built-in destroy operators in BALANS. These include Crossover [Rothberg, 2007], DINS [Ghosh, 2007], Local Branching [Fischetti and Lodi, 2003], Mutation [Rothberg, 2007], Proximity Search [Fischetti and Monaci, 2014], Random Objective, RENS [Berthold, 2014], and RINS [Danna *et al.*, 2005].

Conceptually, destroying neighborhoods is aimed at creating a sub-MIP that is easier than the original problem. This is achieved by either altering the objective function, as done in Proximity Search and Random Objective, or by adding constraints based on a certain set of destroy variables, as done in other operators. We adapted the original definitions of these operators for the ALNS(MIP) context. The main idea behind our adaptations is to ensure that the operator does not attempt an identical neighborhood in subsequent iterations. For example, local branching is inherently deterministic, and we modify it by introducing a randomized destroy size in every iteration. We provide the precise definitions and details of the selection of destroy heuristics in Appendix B [Cai *et al.*, 2024c].

Let us note that there are destroy neighborhood operators that are not implemented in BALANS. Most notable is the Local Branching Relaxation (lb-relax) from [Huang *et al.*, 2023a]. This is the state-of-the-art LNS(MIP) approach. However, lb-relax has a hyper-parameter to control the destroy size that must be chosen carefully for each problem

domain. In addition, the initial destroy size is then dynamically adjusted during the search according to a fixed schedule. The hybrid nature of lb_relax combined with a tuned destroy size and its dynamic adjustment is key to its state-of-the-art LNS(MIP) performance.

In contrast, in BALANS, our destroy operators are not mixing different flavors together and are designed to be distinct to constitute a diverse portfolio. In principle, an effective online learning algorithm would be able to sequence these distinct operators in a way to obtain the desired hybrid behavior for the instance at hand. Some destroy operators such as DINS, Local Branching, and Proximity work only on specific MIP subfamilies, e.g. binary or integer only. When combined together in BALANS, we cover all subfamilies of MIPs. Moreover, notice that we do not need to tune for the destroy size, a common requirement of previous work. We simply introduce the same operator multiple times in our portfolio with varying destroy sizes, serving as different options to choose from during the search. The question then becomes how to learn effectively during search and choose among different options on the fly.

3.2 Online Learning

We identify two types of exploration that are crucial for effective online learning. The first exploration decision stems from state exploration, which decides whether the search should continue with the next state or discard the move. The second exploration decision stems from neighborhood exploration, which decides the destroy neighborhood operator to apply at each state. Distinguishing the two different needs of exploration and addressing them separately in a principled manner is a novelty of our approach and key to good performance.

State Exploration: As shown in Figure 1, the acceptance criterion governs the search space among states. We consider two complementary acceptance criteria: Hill Climbing (HC) and Simulated Annealing (SA). HC mostly exploits yet allows the search to progress to the next state when the objective value is the same. On the other hand, SA offers more exploration capacity and allows the search to move to worsening next states.

Neighborhood Exploration: We employ multi-armed bandits for choosing among different neighborhoods. To apply MAB, there are three important design decisions: the definition of *arms*, the *reward mechanism*, and the *learning policy*.

As shown in Figure 1, in BALANS, we treat *every pair* of destroy and repair operators as a single arm. When an arm is selected, the neighborhood operation is *atomic*; it destroys the current state and then immediately repairs it to obtain the next state. In this paper, given that we only have a single repair operator, which re-optimizes the sub-MIP obtained from applying to destroy, the list of arms can be treated identically with our destroy operators. If desired, users of BALANS can introduce additional repair operators.

Our reward mechanism for MAB is designed specifically for the ALNS(MIP) framework.

We consider four distinct rewards aligned to the possible outcomes of the acceptance criterion as part of state exploration. These four distinct outcomes are: the next state is the

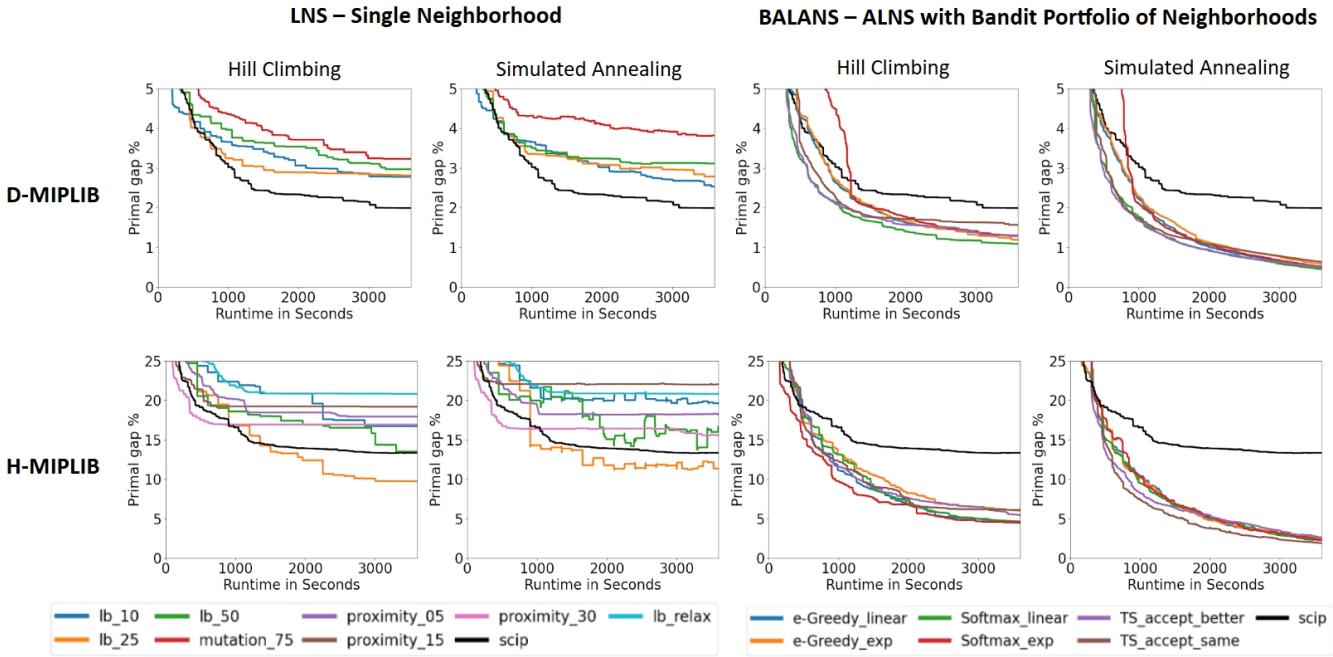


Figure 2: The primal gap (the lower, the better) as a function of time, averaged over instances from D-MIPLIB (top row) and H-MIPLIB (bottom row). We compare LNS (four plots on the left) that commits to a single neighborhood vs. BALANS (four plots on the right) that uses ALNS with a bandit portfolio of neighborhoods. For LNS, each line represents a different neighborhood. For BALANS, each line represents a different learning policy. Both approaches are run with Hill Climbing and Simulated Annealing. Baseline, LNS and BALANS use SCIP as the base MIP solver. Within each dataset (row), the SCIP baseline, shown as the black line, is identical.

best solution found so far, the next state is better than the previous state, the next state is neither the best nor better, but it is accepted as a move, or the next state is rejected. Each outcome is associated with a specific reward value, as later listed in our experiments. Finally, MAB learns from historical arm choices associated with their observed rewards based on its learning policy.

To summarize our overall algorithm, once the initial solution is found, the main ALNS loop starts, which is an interplay between LNS and MAB. First, MAB selects an arm among the different neighborhoods and provides it to LNS. Then, LNS applies the operation and observes the outcome. Based on the resulting state, its solution quality, and the acceptance criterion, LNS provides the reward feedback to MAB. MAB updates the reward estimate of the selected arm according to its learning policy. This process repeats until hitting the stopping condition, and at the end, the best solution found during the search is returned.

4 Experiments

What remains to be seen is the performance of BALANS in practice. For that purpose, our main research questions are:

Q1: What is the performance comparison between the default MIP, LNS(MIP) that commits to a single neighborhood, the state-of-the-art LNS(MIP), and our ALNS(MIP) using BALANS? Can BALANS achieve good performance without any offline training and explore states and neighborhoods simultaneously by adapting to the instance at hand on the fly using bandits?

Q2: How is arm selection among the portfolio of neighborhoods distributed in our bandit strategy? Does BALANS depend on the single best neighborhood, or can it improve over the single best by applying weaker operators sequentially in an adaptive fashion?

Datasets: In our experiments³, we use the recently introduced Distributional MIPLIB (D-MIPLIB) [Huang *et al.*, 2024] and the commonly used MIPLIB2017-Hard (H-MIPLIB) [Gleixner *et al.*, 2021]. For the former, we randomly select 10 instances from Multiple Knapsack, Set Cover, Maximum Independent Set, Minimum Vertex Cover, and Generalized Independent Set Problem, yielding 50 instances. For the latter, we consider a subset that permits a feasible solution within 20 seconds, yielding 43 instances. SCIP and Gurobi cannot solve any of these instances to optimality within 1 hour, ensuring the hardness of our benchmarks.

Approaches: For comparison, we consider the following:

- MIP: We use **SCIP** and **Gurobi**, the state-of-the-art open-source and commercial MIP solvers with default settings running single thread [Bolusani *et al.*, 2024; Gurobi, 2024].

- State-of-the-art LNS(MIP): We use **lb-relax** thanks to the original implementation from [Huang *et al.*, 2023a]. This algorithm selects the neighborhood with the local branching relaxation heuristic as discussed earlier.

- Single Neighborhood LNS(MIP): All eight operators are implemented and readily available in BALANS to serve in

³<https://huggingface.co/datasets/balans>

	Distributional MIPLIB (D-MIPLIB)				MIPLIB Hard (H-MIPLIB)			
	Hill Climbing		Simulated Annealing		Hill Climbing		Simulated Annealing	
	PG (%) ↓	PI ↓	PG (%) ↓	PI ↓	PG (%) ↓	PI ↓	PG (%) ↓	PI ↓
SCIP	2.11±1.68	105±56.5	N/A	N/A	13.3±3.95	514±134	N/A	N/A
lb_relax	5.72± 3.25	163±73.6	N/A	N/A	24.2±15.5	778±156	N/A	N/A
local_branching_25	2.80±2.27	110±81.2	2.77±2.03	115±77.4	9.73±2.11	485±95.9	11.4±2.34	502±93.0
mutation_75	2.21±3.98	104±166	2.81±4.9	114±176	57.7±139	971±473	58.8±139	991±472
proximity_15	29.6±27.8	771±253	29.6±27.8	771±254	16.9±25.6	583±88.4	15.6±23.7	562±86.0
e-Greedy_linear	0.92±1.47	69.5±62.6	0.53±0.85	61.1±51.8	4.47±0.86	316±56.3	3.43±0.59	295±49.9
e-Greedy_exp	0.84±1.15	71.1±63.6	0.65±1.01	73.9±86.0	6.01±1.01	361±61.5	3.74±0.64	294±49.4
Softmax_linear	0.71±1.02	59.1±68.5	0.49±0.81	53.8±57	4.65±0.78	325±56.6	3.47±0.53	296±50.5
Softmax_exp	0.94±1.39	127±285	0.55±0.91	111±241	4.5±0.77	277±44.6	3.53±0.63	307±51.4
TS_accept_better	0.88±1.51	73.2±94.4	0.54±0.91	59.4±77	5.45±0.92	341±56.9	4.09±0.72	284±47.5
TS_accept_same	1.19±2.02	84.2±111	0.75±1.25	69.6±86.9	6.11±1.30	340±63.2	2.94±0.55	244±39.5

Table 1: Primal gap (PG) percentage and primal integral (PI) at 1 hour time limit averaged over instances from D-MIPLIB and H-MIPLIB and their standard deviations. The lower, the better (“↓”). The best performance in each dataset is bolded. Baseline, LNS and BALANS use **SCIP** as the base solver.

LNS(MIP). That said, in our experiments, we found DINS and Random Objective to perform poorly, hence we focus on the remaining six destroy operators. By varying the parameters of these operators, we obtain 16 different destroy operators from 6 unique neighborhood definitions. These are: crossover, lb_10/25/50 (local branching), mutation_25/50/75, proximity_05/15/30, rns_25/50/75, and rins_25/50/75. For the accept criterion, we use Hill Climbing (HC) and Simulated Annealing (SA) with an initial temperature set to 20 and an end temperature set to 1 with a step size of 0.1.

- BALANS ALNS(MIP): Given the 16 different single destroy operators used in LNS(MIP), we build BALANS for ALNS(MIP) with a portfolio that includes all of the 16 operators. We again apply HC and SA as the acceptance criterion. For the learning policy, we use e-Greedy [Auer *et al.*, 2002] and Softmax [Luce, 1977] with numeric rewards and Thompson Sampling (TS) [Thompson, 1933] with binary rewards. For numeric rewards, we use a linear ([3, 2, 1, 0]) and an exponential setting ([8, 4, 2, 1]) corresponding to best, better, accept, and reject outcomes. For binary rewards, we consider [1, 1, 0, 0], where accept reward is the same as reject, and [1, 1, 1, 0], where accept is better than reject.

Evaluation Metrics: 1) Primal Gap (PG) [Berthold, 2006] is the normalized difference between the primal bound v and a precomputed best known objective value v^* and is defined as $\frac{|v-v^*|}{\max(|v^*|, \epsilon)}$ if v exists and $vv^* \geq 0$. We use $\epsilon = 10^{-8}$ to avoid zero division.

2) Primal Integral (PI) [Achterberg *et al.*, 2012] at time q is the integral on $[0, q]$ of the primal gap as a function of runtime. PI captures the quality of and the speed at which solutions are found.

Setup: We conduct experiments on AWS EC2 Trn1 with 128 vCPUs and 512GB memory. BALANS solver in-

tegrates ALNS library [Wouda and Lan, 2023], MAB-WISER library [Strong *et al.*, 2019; Strong *et al.*, 2021] and SCIP(v9.0.0) [Bolusani *et al.*, 2024] and GUROBI (v11.0.0) [Gurobi, 2024]. To find an initial solution, we run the solver for 20 seconds. Each LNS iteration is limited to 1 minute, except for Local Branching to 2.5 minutes, which solves larger sub-problems than other operators. The time limit to solve each instance is set to 1 hour.

4.1 Numerical Results

[Q1] Default MIP vs. LNS(MIP) vs. BALANS Figure 2 presents a detailed comparison of default MIP via SCIP, LNS(MIP) via single neighborhood operators including the state-of-the-art lb-relax [Huang *et al.*, 2023a], and BALANS solver conducting ALNS(MIP).

Performance of LNS(MIP): On the left four plots, the immediate observation is the mixed performance of single neighborhood LNS, losing to SCIP (above the black line) in almost all cases. On D-MIPLIB, only Local Branching (lb) and Mutation get close to SCIP, but still worse. All others, including lb-relax, perform poorly above the primal gap cut-off. On H-MIPLIB, both Proximity and Local Branching are comparable to SCIP, with lb_25 surpassing SCIP. It is important to note that these are aggregate results across problems. As expected, the state-of-the-art lb_relax performs comparable and even better than SCIP in some domains. Domain-specific results can be found in the supplemental material. Overall, when dealing with MIP instances from various distributions and mixed problem types, single neighborhood approaches, including lb_relax, struggle. LNS(MIP) requires carefully chosen neighborhood sizes and cannot adapt to the diversity of mixed problem types, number of variables, constraints, and problem structure across instances.

	Crossover	Local Branching	Mutation	Proximity	RENS	RINS
D-MIPLIB (Softmax_linear_SA)	6.4%	11%	25%	17.9%	19.8%	19.9%
H-MIPLIB (TS_accept_same_SA)	9.4%	0.6%	21%	7.3%	31%	30.8%

Table 2: The distribution of arm selection as a percentage for each unique destroy operator of the best BALANS configurations.

Performance of BALANS: Figure 2 shows that, contrary to LNS(MIP), the ALNS(MIP) approach of BALANS significantly outperforms SCIP (below the black line) *in all cases on both datasets*. In fact, it immediately starts performing better, diving into the lower optimality gap early in the search. This result is remarkable since BALANS uses the same 16 single neighborhoods from LNS(MIP), most of which do not even reach the gap cutoff when running standalone. An important insight is that the ranking of single LNS(MIP) operators varies considerably between datasets. For instance, Mutation performs the best on D-MIPLIB yet is absent in H-MIPLIB, whereas Proximity exhibits the opposite behavior. This is exactly the motivation behind our diverse portfolio of destroy operators. Notice also how *all bandit learning policies* are consistently better than SCIP. Our online bandits are able to select the appropriate operator adaptively throughout the search for the instance at hand. Moreover, unlike single operators that require carefully tuned destroy parameters, in the absence of which they perform poorly, BALANS simply introduces parameter variations as additional arms driven by the learning policy.

Table 1 zooms further into the primal gap and primal integral achieved by the top performers. As in Figure 2, it is evident that *any configuration* of BALANS significantly outperforms SCIP, the state-of-the-art lb-relax, and any LNS. Overall, we reduce the primal gap of SCIP by **75%** and the primal integral of SCIP by **50%** across datasets. The best BALANS configuration is Softmax_linear_SA for D-MIPLIB and TS_accept_same_SA for H-MIPLIB. Regardless, any BALANS configuration is better than the alternatives, revealing its robust out-of-the-box performance.

Figure 3 plots the primal gap performance of the best BALANS configuration within each dataset compared to SCIP on each instance. Above the diagonal is where SCIP wins, and below the diagonal is where BALANS wins. As clearly shown in Figure 3, BALANS achieves superior performance on a greater number of instances, particularly on those where SCIP struggles. This highlights the effectiveness of our approach in challenging instances.

Performance of Acceptance Criteria: For single operators in LNS(MIP), different acceptance criteria do not substantially change the performance. They can be considered too rigid to benefit from state exploration. In contrast, SA consistently outperforms HC for BALANS. The best BALANS performances are achieved with SA that allows navigating the search space even with worsening moves.

[Q2] Distribution of Arm Selection What is behind the operations of BALANS, and specifically, what neighborhoods does it depend on for its superior performance?

Table 2 shows the distribution of arm selection as percent-

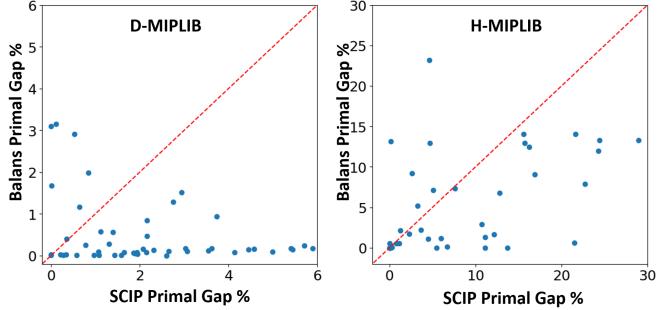


Figure 3: Primal gap of SCIP (x-axis) vs. BALANS (y-axis).

ages. Interestingly, the single best operator, local branching, is *not* a popular arm at all. Several worse operators are preferred within our bandit portfolio. In H-MIPLIB, BALANS utilizes local branching *only 0.6%* of the time, yet still, it outperforms local branching by a large margin. Likewise, in D-MIPLIB, local branching is second to last in popularity.

The RENS and RINS operators, which are not even visible in Figure 2 due to their poor performance above the primal gap cutoff, account for $\sim 40\%$ and $\sim 60\%$ of the distributions in Table 2 on these datasets. This shows that BALANS outperforms MIP and any single best LNS(MIP) out-of-the-box with almost no tuning by *using weaker operators sequentially* and carefully balancing the exploitation-exploration trade-off via online learning.

4.2 Ablation Studies

To demonstrate that BALANS is solver-agnostic, we conducted the same experiments using Gurobi [Gurobi, 2024], a state-of-the-art commercial solver.

Figure 4 compares the performance of default Gurobi, the Single Neighborhood LNS approach, and BALANS. The results show that BALANS outperforms both default Gurobi and LNS(Gurobi) with any single neighborhood approach significantly on D-MIPLIB. On H-MIPLIB, where Gurobi performs much better than any LNS methods, our BALANS can significantly reduce the gap from single LNS to Gurobi. This confirms that our findings and conclusions derived from experiments with the SCIP solver carry over when switched to another solver. By conducting experiments with state-of-the-art open-source and commercial solvers, we demonstrate that BALANS can be seamlessly applied as a meta-solver on top of a MIP solver, significantly improving their performance over single neighborhood LNS.

Next, it is important to note that the default configuration of SCIP applies ALNS as an internal heuristic *within* the Branch-and-Bound search, as discussed in [Hendel, 2022]. That means when we use the default SCIP, BALANS essen-

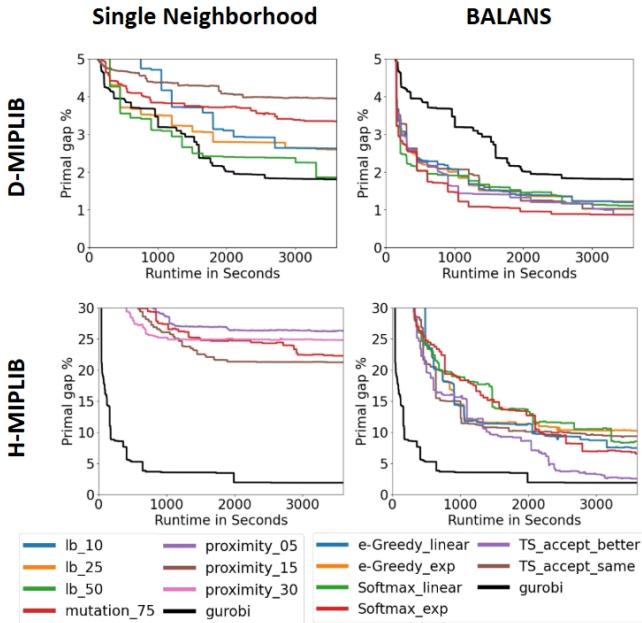


Figure 4: Ablation Study: The same experiments from Figure 2 using Gurobi as the MIP solver with Simulated Annealing shown for the acceptance criterion.

tially conducts ALNS(MIP(ALNS)), which outperforms the default SCIP conducting MIP(ALNS). This raises the question: how much of the good performance of BALANS on these instances is due to the internal ALNS of SCIP? To explore this, we conduct another ablation study with SCIP’s internal ALNS enabled and disabled, comparing MIP vs. MIP(ALNS). Our results show that the internal ALNS of SCIP conducted within the BnB search led to performance improvement in only 8 out of 94 instances, with negligible improvements of less than 0.2% in the primal gap. Contrarily, our novel bandit-based ALNS approach, conducted on top of the BnB search, improves **79 out of 94** instances (Figure 3) and reduces the primal gap by **75+%** (Table1).

5 Related Work

Primal heuristics are crucial in rapidly discovering feasible solutions to improve the primal bound. As such, there is a rich literature on learning-based methods for improving heuristics. Examples in this line of research include IL-LNS [Sonnerat *et al.*, 2021], which learns to select variables by imitating LB, RL-LNS [Wu *et al.*, 2021], which uses a similar framework but trained with reinforcement learning, and CL-LNS [Huang *et al.*, 2023b] uses contrastive learning to learn to predict variables. Our work complements these approaches as an *online method* that does not require any offline training to generate good quality solutions on challenging optimization instances, as shown in our experiments.

There is also a body of work that focuses on primal heuristic operations of BnB *within-the-solver*. These include offline learning methods, such as [Khalil *et al.*, 2017], which builds a mapping between BnB nodes and a binary decision for running a given primal heuristic, and [Chmiela *et al.*, 2021],

which constructs a schedule that assigns priority and computational budget to each heuristic.

When considering online methods [Attieh *et al.*, 2019; Hendel, 2022; Chmiela *et al.*, 2023] are the closest to our work. As in our paper, both are examples of online learning, focus on LNS neighborhoods, use SCIP, and are based on multi-armed bandits. The crucial difference is that BALANS is a *meta-solver* that operates on top of a MIP solver in a solver-agnostic manner, whereas the previous work only works as a heuristic that operates within the BnB search of SCIP, where LNS is invoked at internal search nodes. As a result, these works remain specific extensions of SCIP and are white-box, i.e., require access to solver internals. Contrarily, BALANS is an integration technology that offers a modular architecture to leverage best-in-class open-source software dedicated to their specific domain. We leverage MABWISER [Strong *et al.*, 2019; Strong *et al.*, 2021] for bandits, ALNS library [Wouda and Lan, 2023] for adaptive large-neighborhood search, and SCIP [Bulusani *et al.*, 2024] and GUROBI [Gurobi, 2024] for MIP solving. Future advances in these distinct fields, realized independently within each software, propagate to our meta-solver with compounding effects. This unique integration makes BALANS highly configurable. Beyond this paper, many more configurations are available through parameterization of ALNS and MABWiser.

Finally, let us acknowledge other successful MAB applications. Among those, [Phan *et al.*, 2024] uses MAB with ALNS for multi-agent pathfinding, [Almeida *et al.*, 2020] uses MAB with hyper-heuristics for multi-objective flow shop problems, and [Zheng *et al.*, 2022] uses MAB for MaxSAT. Beyond combinatorial problems, MAB is heavily used in recommender systems [Kadioglu and Kleynhans, 2024] and game-playing agents [Schaul *et al.*, 2019].

6 Conclusions

In this paper, we proposed BALANS, a multi-armed Bandits-based Adaptive Large Neighborhood Search for mixed-integer programming. Our experiments have shown that all configurations of our online approach, without any offline training and almost zero tuning, significantly improve over 1) the default BnB solver, and 2) any single large neighborhood search, including the state-of-the-art, on hard optimization problems. We release BALANS solver as open-source software with its high-level interface, modular and extendable architecture, and configurable design.

For future work, we would like to improve the performance of BALANS further by careful algorithm configuration and portfolio construction [Kadioglu *et al.*, 2010; Kadioglu *et al.*, 2011], and explore hybrid ALNS(MIP), where existing offline training methods are introduced as additional arms in our portfolio.

Acknowledgments

This paper reports on research conducted while Junyang Cai was interned at AI Center of Excellence, Fidelity. The research at the University of Southern California was supported by NSF under grant number 2112533: NSF Artificial

References

- [Achterberg *et al.*, 2012] Tobias Achterberg, Timo Berthold, and Gregor Hendel. Rounding and propagation heuristics for mixed integer programming. In *Operations Research Proceedings 2011: Selected Papers of the International Conference on Operations Research (OR 2011), August 30-September 2, 2011, Zurich, Switzerland*, pages 71–76. Springer, 2012.
- [Almeida *et al.*, 2020] Carolina P Almeida, Richard A Gonçalves, Sandra Venske, Ricardo Lüders, and Myriam Delgado. Hyper-heuristics using multi-armed bandit models for multi-objective optimization. *Applied Soft Computing*, 95:106520, 2020.
- [Attieh *et al.*, 2019] Saad Attieh, Nguyen Dang, Christopher Jefferson, Ian Miguel, and Peter Nightingale. Athanor: high-level local search over abstract constraint specifications in essence. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*. International Joint Conferences on Artificial Intelligence, 2019.
- [Auer *et al.*, 2002] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.
- [Bengio *et al.*, 2021] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d’horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.
- [Berthold, 2006] Timo Berthold. *Primal heuristics for mixed integer programs*. PhD thesis, Zuse Institute Berlin (ZIB), 2006.
- [Berthold, 2014] Timo Berthold. Rens: the optimal rounding. *Mathematical Programming Computation*, 6:33–54, 2014.
- [Bulusani *et al.*, 2024] Suresh Bulusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, et al. The scip optimization suite 9.0. *arXiv preprint arXiv:2402.17702*, 2024.
- [Cai *et al.*, 2024a] Junyang Cai, Taoan Huang, and Bistra Dilkina. Learning backdoors for mixed integer programs with contrastive learning. *arXiv e-prints*, pages arXiv–2401, 2024.
- [Cai *et al.*, 2024b] Junyang Cai, Taoan Huang, and Bistra Dilkina. Multi-task representation learning for mixed integer linear programming. *arXiv preprint arXiv:2412.14409*, 2024.
- [Cai *et al.*, 2024c] Junyang Cai, Serdar Kadioglu, and Bistra Dilkina. Balans: Multi-armed bandits-based adaptive large neighborhood search for mixed-integer programming problem. *arXiv preprint arXiv:2412.14382*, 2024.
- [Chmiela *et al.*, 2021] Antonia Chmiela, Elias Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. Learning to schedule heuristics in branch and bound. *Advances in Neural Information Processing Systems*, 34:24235–24246, 2021.
- [Chmiela *et al.*, 2023] Antonia Chmiela, Ambros Gleixner, Paweł Lichocki, and Sebastian Pokutta. Online learning for scheduling mip heuristics. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 114–123. Springer, 2023.
- [Danna *et al.*, 2005] Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102:71–90, 2005.
- [Ding *et al.*, 2020] Jian-Ya Ding, Chao Zhang, Lei Shen, Shengyin Li, Bing Wang, Yinghui Xu, and Le Song. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *Proceedings of the aaai conference on artificial intelligence*, volume 34, pages 1452–1459, 2020.
- [Fischetti and Lodi, 2003] Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical programming*, 98:23–47, 2003.
- [Fischetti and Monaci, 2014] Matteo Fischetti and Michele Monaci. Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics*, 20:709–731, 2014.
- [Ghosh, 2007] Shubhashis Ghosh. Dins, a mip improvement heuristic. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 310–323. Springer, 2007.
- [Gleixner *et al.*, 2021] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, et al. Miplib 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021.
- [Gurobi, 2024] Gurobi. Gurobi Optimizer Reference Manual, 2024. Accessed: 2024-08-10.
- [He *et al.*, 2014] He He, Hal Daume III, and Jason M Eisner. Learning to search in branch and bound algorithms. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.
- [Hendel, 2022] Gregor Hendel. Adaptive large neighborhood search for mixed integer programming. *Mathematical Programming Computation*, 14(2):185–221, 2022.
- [Huang *et al.*, 2023a] Taoan Huang, Aaron Ferber, Yuan-dong Tian, Bistra Dilkina, and Benoit Steiner. Local branching relaxation heuristics for integer linear programs. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 96–113. Springer, 2023.

- [Huang *et al.*, 2023b] Taoan Huang, Aaron M Ferber, Yuan-dong Tian, Bistra Dilkina, and Benoit Steiner. Searching large neighborhoods for integer linear programs with contrastive learning. In *International Conference on Machine Learning*, pages 13869–13890. PMLR, 2023.
- [Huang *et al.*, 2024] Weimin Huang, Taoan Huang, Aaron M Ferber, and Bistra Dilkina. Distributional miplib: a multi-domain library for advancing ml-guided milp methods. *arXiv preprint arXiv:2406.06954*, 2024.
- [Kadioglu and Kleynhans, 2024] Serdar Kadioglu and Bernard Kleynhans. Building higher-order abstractions from the components of recommender systems. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*, pages 22998–23004. AAAI Press, 2024.
- [Kadioglu *et al.*, 2010] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC - instance-specific algorithm configuration. In Helder Coelho, Rudi Studer, and Michael J. Wooldridge, editors, *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 751–756. IOS Press, 2010.
- [Kadioglu *et al.*, 2011] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 454–469. Springer, 2011.
- [Kadioglu *et al.*, 2012] Serdar Kadioglu, Yuri Malitsky, and Meinolf Sellmann. Non-model-based search guidance for set partitioning problems. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012.
- [Khalil *et al.*, 2016] Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.
- [Khalil *et al.*, 2017] Elias B Khalil, Bistra Dilkina, George L Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *Ijcai*, pages 659–666, 2017.
- [Kuleshov and Precup, 2014] Volodymyr Kuleshov and Doina Precup. Algorithms for multi-armed bandit problems. *CoRR*, abs/1402.6028, 2014.
- [Land and Doig, 2010] Ailsa H Land and Alison G Doig. *An automatic method for solving discrete programming problems*. Springer, 2010.
- [Lodi and Zarpellon, 2017] Andrea Lodi and Giulia Zarpellon. On learning and branching: a survey. *Top*, 25(2):207–236, 2017.
- [Luce, 1977] R Duncan Luce. The choice axiom after twenty years. *Journal of mathematical psychology*, 15(3):215–233, 1977.
- [Phan *et al.*, 2024] Thomy Phan, Taoan Huang, Bistra Dilkina, and Sven Koenig. Adaptive anytime multi-agent path finding using bandit-based large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17514–17522, 2024.
- [Pisinger and Ropke, 2019] David Pisinger and Stefan Ropke. Large neighborhood search. *Handbook of metaheuristics*, pages 99–127, 2019.
- [Rothberg, 2007] Edward Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.
- [Schaul *et al.*, 2019] Tom Schaul, Diana Borsa, David Ding, David Szepesvari, Georg Ostrovski, Will Dabney, and Simon Osindero. Adapting behaviour for learning progress. *arXiv preprint arXiv:1912.06910*, 2019.
- [Song *et al.*, 2020] Jialin Song, Ravi Lanka, Yisong Yue, and Bistra Dilkina. A general large neighborhood search framework for solving integer programs. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.
- [Sonnerat *et al.*, 2021] Nicolas Sonnerat, Pengming Wang, Ira Ktena, Sergey Bartunov, and Vinod Nair. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201*, 2021.
- [Strong *et al.*, 2019] Emily Strong, Bernard Kleynhans, and Serdar Kadioglu. Mabwiser: A parallelizable contextual multi-armed bandit library for python. In *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019*, pages 909–914. IEEE, 2019.
- [Strong *et al.*, 2021] Emily Strong, Bernard Kleynhans, and Serdar Kadioglu. MABWiser: parallelizable contextual multi-armed bandits. *Int. J. Artif. Intell. Tools*, 30(4):2150021:1–2150021:19, 2021.
- [Thompson, 1933] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.
- [Tong *et al.*, 2024] Jiatai Tong, Junyang Cai, and Thiago Serra. Optimization over trained neural networks: Taking a relaxing walk. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 221–233. Springer, 2024.
- [Vermorel and Mohri, 2005] Joannes Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical eval-

- uation. In *European conference on machine learning*, pages 437–448. Springer, 2005.
- [Wouda and Lan, 2023] Niels A. Wouda and Leon Lan. ALNS: a Python implementation of the adaptive large neighbourhood search metaheuristic. *Journal of Open Source Software*, 8(81):5028, 2023.
- [Wu *et al.*, 2021] Yaxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Learning large neighborhood search policy for integer programming. *Advances in Neural Information Processing Systems*, 34:30075–30087, 2021.
- [Zheng *et al.*, 2022] Jiongshi Zheng, Kun He, Jianrong Zhou, Yan Jin, Chu-Min Li, and Felip Manya. Bandmaxsat: A local search maxsat solver with multi-armed bandit. *arXiv preprint arXiv:2201.05544*, 2022.

Appendix

In this Appendix, we present a quick start example of BALANS showcasing its ease-of-use and high-level API design, a detailed explanation of our destroy operators and how they differ from their original reference, the detailed results on each problem distribution, and finally, detailed results and analysis about offline and online training comparison.

A BALANS Quick Start Example

```
# Adaptive large neighborhood via ALNS library
from alns.select import MABSelector
from alns.accept import HillClimbing
from alns.stop import MaxIterations

# Contextual multi-armed bandits via MABWiser library
from mabwiser.mab import LearningPolicy

# Meta-solver for MIPs via SCIP library
from balans.solver import Balans, DestroyOperators, RepairOperators

# Balans: Online Meta-Solver for MIPs
balans = Balans(destroy_ops=[DestroyOperators.Crossover,
                             DestroyOperators.Dins,
                             DestroyOperators.Mutation,
                             DestroyOperators.Local_Branching,
                             DestroyOperators.Proximity,
                             DestroyOperators.Rens,
                             DestroyOperators.Rins,
                             DestroyOperators.Random_Objective],
                 repair_ops=[RepairOperators.Repair],
                 selector=MABSelector(scores=[1, 1, 0, 0], num_destroy=8, num_repair=1,
                                       learning_policy=LearningPolicy.ThompsonSampling()),
                 accept=HillClimbing(),
                 stop=MaxIterations(100))

# Run on MIP instances
result = balans.solve("MIP_instance.mps")

# Result for the best solution found
print("Best solution:", result.best_state.solution())
print("Best solution objective:", result.best_state.objective())
```

As shown in our Quick Start Example, BALANS⁴ solver is characterized by a set of *destroy operators* and a set of *repair operators*. The pairwise combination of these two is treated as an *arm* in the multi-armed bandits, with the option to declare valid arm pairs if needed. A *selector* controls the selection among the arms. The selector uses a given learning policy, which is rewarded for finding the best, better, accepted, and rejected state according to the rewards given in the *scores* list. The ALNS iterates following the *acceptance criterion* and terminates with the *stopping condition*.

We contribute BALANS to the open-source as a meta-solver for solving MIPs. It serves as an integration technology that combines ALNS library for adaptive large-neighborhood search, MABWISER library for multi-armed bandits, and SCIP library for solving MIPs. Our integration yields an elegant interface with several configurable options, from reward mechanisms and learning policies to acceptance criteria. The framework is available in the Python Package Index (PyPI) and can be installed with a one-liner⁵.

BALANS provides eight built-in destroy operators and one repair operator. It takes advantage of *any learning policy* available in MABWISER library and *any acceptance and stopping criteria* available in ALNS library. Our modular software leverages best-in-class open-source software dedicated to their specific algorithms for bandits, adaptive large-neighborhood search, and MIP solving. Further advances in these distinct domains realized independently in each software propagate to enhancing and compounding our BALANS meta-solver.

⁴<https://github.com/anonymous/balans/>

⁵pip install balans

Destroy Neighborhood Definition		Added Constraint(s)
Crossover	$Destroy = \{x_{t-1}^k == x_{rnd}^k, \forall k \in D\}$	$\{x_t^k = x_{t-1}^k \mid k \notin Destroy, \forall k \in V\}$
DINS (IP)	$Destroy = \{ x_{t-1}^k - x_{lp}^k \geq 0.5, \forall k \in I\}$	$\begin{cases} x_t^k - x_{lp}^k \leq x_{t-1}^k - x_{lp}^k & \mid k \in Destroy \\ x_t^k = x_{t-1}^k & \mid k \notin Destroy \end{cases}, \forall k \in V$
Local Branching (BP)	$Destroy = \Delta B$	$\sum_{k \in B} x_t^k - x_{t-1}^k \leq Destroy $
Mutation	$Destroy = \Delta D$	$\{x_t^k = x_{t-1}^k \mid k \notin Destroy, \forall k \in V\}$
Proximity Search (BP)	$f'(x_t) = \sum_{k \in B} x_t^k - x_{t-1}^k $	$f(x_t) \leq \Delta f(x_{t-1})$
Random Objective	$f'(x_t) = \sum_{k \in V} \Delta x_t^k$	-
RENS	$Destroy = \Delta \{x_{lp}^k \notin \mathbb{Z}, \forall k \in D\}$	$\begin{cases} x_t \geq \lfloor x_{lp}^k \rfloor, x_t \leq \lceil x_{lp}^k \rceil & \mid k \in Destroy \\ x_t^k = x_{t-1}^k & \mid k \notin Destroy \end{cases}, \forall k \in V$
RINS	$Destroy = \Delta \{x_{t-1}^k == x_{lp}^k, \forall k \in D\}$	$\{x_t^k = x_{t-1}^k \mid k \notin Destroy, \forall k \in V\}$

Table 3: Eight destroy operators in BALANS with their destroy set of variables, modified objective, and constraints. We have a single repair operator that re-optimizes the MIP resulting from applying one of these destroy neighborhoods to the current state. The parentheses next to the operator name indicate the MIP problem type for which the operator is suitable for. Let x_{t-1}^n be the solution from previous state, x_t^n be the solution for current state, x_{lp}^n be the LP relaxation solution, and x_{rnd}^n be a random feasible solution. Let V denote list of variable indices, B denote list of binary indices, I denote list of integer indices, and D denote list of discrete indices. Let $f(x)$ to denote the objective function. The Δ is a hyper-parameter that defines either the size of the destroy set, the coefficient of variables, or the objective function.

B The Details of LNS and Destroy Operators

In this section, we provide more details on the LNS procedure and the details of our destroy operators as implemented in BALANS. The general LNS procedure is as follows:

- Given a MIP, (A)LNS starts with a feasible solution. That is, we have a “complete assignment” to all variables and the corresponding value of the objective function. This is called a “state”.
- Then, when a Destroy operator is applied to a state, it takes this “complete assignment” and returns a sub-MIP problem. Essentially, the destroy operation transitions from a complete solution back to an unsolved MIP problem. The main motivation is that this sub-problem is easier than solving the original MIP.
- Finally, a Repair operator takes an unsolved sub-MIP and solves/repairs it. This repair leads to another “complete assignment” of variables. This forms the next state. An Acceptance Criterion decides whether we move from the completed assignment (solution) of the current state to the complete assignment (solution) of the next state.
- The interplay between the Destroy/Repair cycle is repeated until a Stop Criterion is encountered.

The destroy operators implemented in BALANS with our modifications are as follows and given in Table 3.

- Crossover:** Crossover generates a random feasible solution using Random Objective and solution from the previous state to compare the value of discrete variables between them. If they have the same value, fix the variable to that value. This differs from the original, which takes two random feasible solutions and compares them against each other. Using the incumbent solution allows this approach to have connections to the current state information.
- DINS:** DINS takes the LP relaxed solution of the original MIP and solution from the previous state to record the difference of value between each discrete variable. If the difference is less than 0.5 for each discrete variable, we fix the variables to the previous state value. Otherwise, we bound the variables around the initial LP relaxed solution. In this approach, we remove the binary local branching part because we want each operator to be unique enough.
- Local Branching:** Local Branching tracks the value of binary variables in the previous state and only allows a fraction of the binary variables to flip by adding one single constraint to the original MIP that allows $\leq k$ to change value and solving this modified MIP maximizing the original objective. We add a delta variable to control how many percent of variables we allow to flip at most. We select a random flip size between 10% to the max percentage in every iteration. Instead of having a neighborhood with a fixed number of destroyed variables, we allow randomness in the neighborhood in every iteration.
- Mutation:** Mutation fixes a subset of the discrete variables to the value from the previous state. We add a delta variable to control what percent of the discrete variables is not in the subset to be fixed.

	Problem Type		fit ALNS			Modify SubProblem	
	only binary variables	no binary variables	state dependent	randomness	need LP relaxation	add constraint(s)	change objective
Crossover	green	white	green	green	red	green	red
DINS	red	white	white	white	green	white	red
Local Branching	green	red	white	green	red	white	red
Mutation	green	white	red	white	red	white	red
Proximity	white	red	white	white	red	white	white
Random Objective	green	white	red	green	red	white	green
RENS	red	white	white	white	white	white	red
RINS	green	white	green	green	white	green	red

Figure 5: Different characteristics for each operator, where green means satisfy and red means not satisfy.

- **Proximity Search:** Proximity Search finds a feasible solution with a better objective that is as close as the previous state's solution. It adds a constraint to make the objective function smaller than the previous one, replaces the objective function with the distance function between the current and previous state solutions, and minimizes the new objective. We add a delta variable to control what percent of improvement we want from the previous state. We also added a Slack variable in the constraint and objective to prevent the modified problem from becoming infeasible and make it fit in the ALNS context.
- **Random Objective:** Random Objective is a new heuristic we create to explore the problem space's feasible region randomly. We replace the objective function by generating a random coefficient from -1 to 1 for every variable in the problem. This acts as a single heuristic and works as a helper function for random feasible solutions in crossover and initial starting points for ALNS.
- **RENS:** RENS fix variables with an integer LP relaxation value to the previous state value. For variables with a fractional LP relaxation value, it restricts them from rounding up and down integer versions of that fractional value. To introduce randomness in this heuristic, we introduce a delta variable to control the percentage of floating variables to be rounded.
- **RINS:** RINS takes the LP relaxation solution of the original MIP and the solution from the previous state to compare the value of discrete variables between them. We choose a subset of variables with different values and fix all other variables not in the set. We introduce a delta variable to control how many percentages of variables have different values in the subset.

Combining these eight destroy heuristics creates a diverse portfolio, including state-dependent operators and not state-independent, random exploration operators and exploration close to the previous state neighborhood, binary problem operators, and general discrete operators. Figure 5 presents the characteristics of each destroy operator.

C Detailed Performance in Each Problem Domain

The following Figures 2-6 show the performance of single heuristics and BALANS on different distributions in D-MIPLIB: Multiple Knapsack (MK), Set Cover (SC), Generalized Independent Set (GISP), Minimum Vertex Cover (MVC), Maximum Independent Set (MIS). The set of experiments is done in SCIP as base solver for default solver, single heuristic and BALANS.

The figures on the Left show the performance of single heuristics, and the figures on the Right show the performance of BALANS. All the approaches use Simulated Annealing as accept criteria. As can be seen from these results, the performance of single heuristics varies considerably across different distributions. Meanwhile, our BALANS solver, composed of these same operators in a bandit portfolio, performs well on *every distribution*.

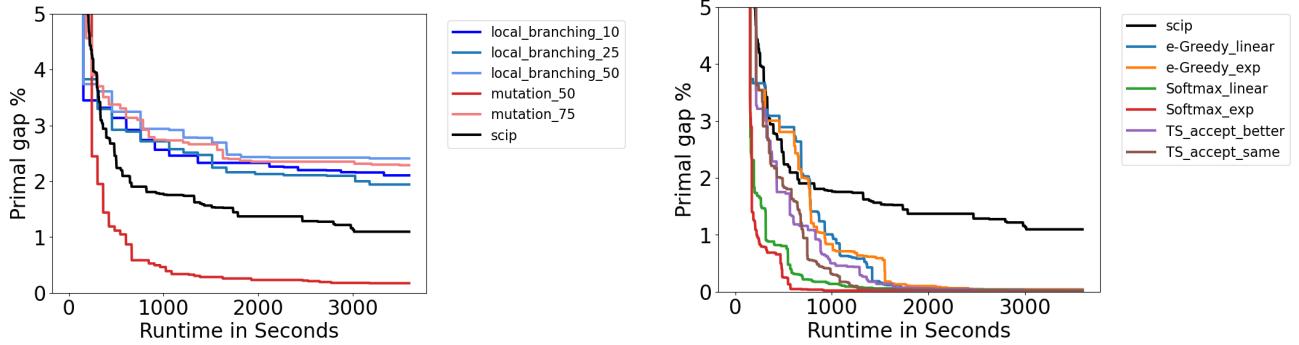


Figure 6: The primal gap (the lower, the better) as a function of time, averaged over instances from MK distribution

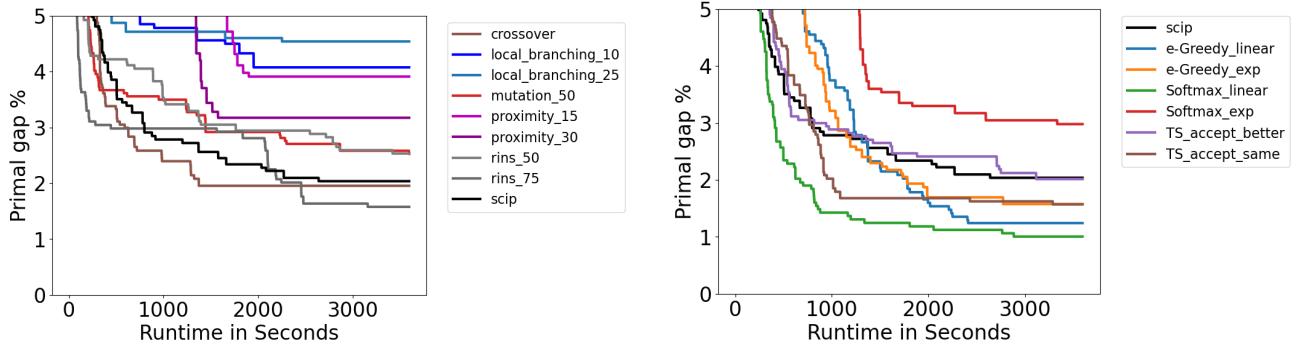


Figure 7: The primal gap (the lower, the better) as a function of time, averaged over instances from SC distribution

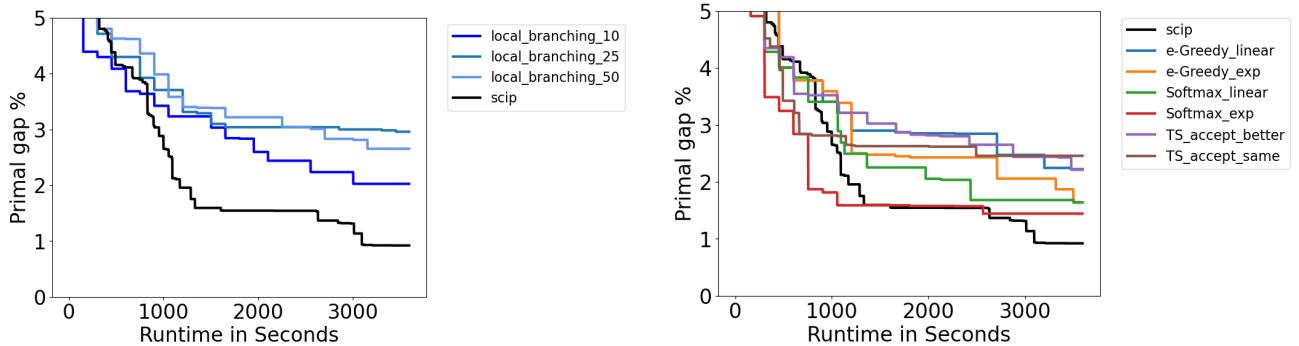


Figure 8: The primal gap (the lower, the better) as a function of time, averaged over instances from GISP distribution

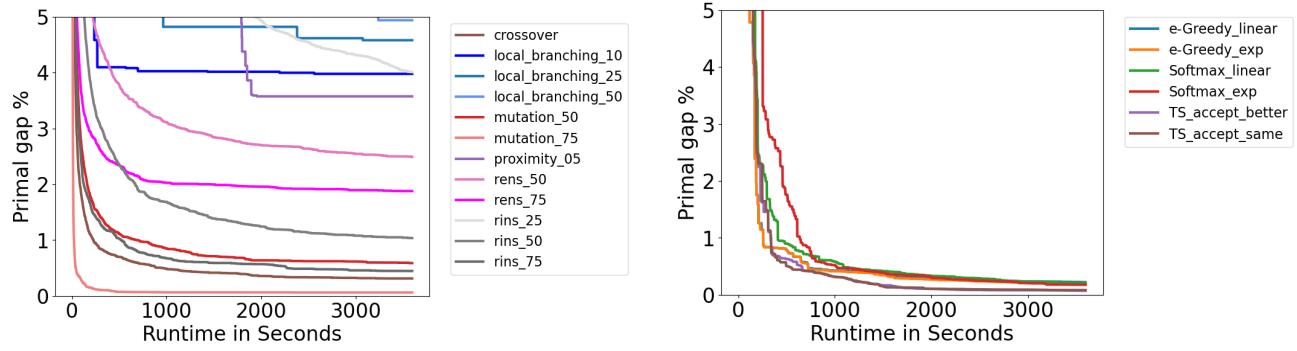


Figure 9: The primal gap (the lower, the better) as a function of time, averaged over instances from MVC distribution

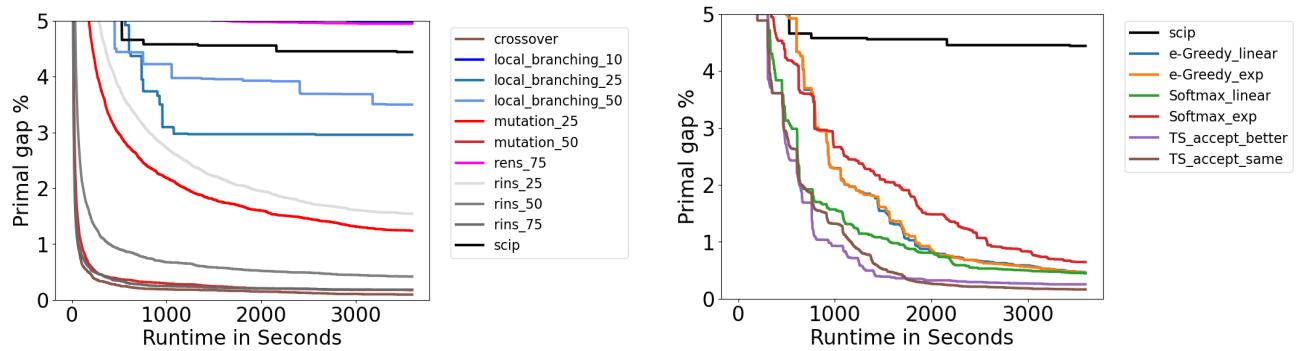


Figure 10: The primal gap (the lower, the better) as a function of time, averaged over instances from MIS distribution