arXiv:2212.08183v2 [cs.AI] 31 May 2023

Local Branching Relaxation Heuristics for Integer Linear Programs

Taoan Huang (✉)[1][0000−0002−8733−2379], Aaron Ferber[1][0000−0002−7422−0044], Yuandong Tian[2][0000−0003−4202−4847], Bistra Dilkina[1][0000−0002−6784−473X], and Benoit Steiner[2][0000−0002−5767−4976]

[1] University of Southern California, Los Angeles, USA
{taoanhua,aferber,dilkina}@usc.edu
[2] Meta AI (FAIR), Menlo Park, USA
yuandong@meta.com, benoit.papers@gmail.com

Abstract. Large Neighborhood Search (LNS) is a popular heuristic algorithm for solving combinatorial optimization problems (COP). It starts with an initial solution to the problem and iteratively improves it by searching a large neighborhood around the current best solution. LNS relies on heuristics to select neighborhoods to search in. In this paper, we focus on designing effective and efficient heuristics in LNS for integer linear programs (ILP) since a wide range of COPs can be represented as ILPs. Local Branching (LB) is a heuristic that selects the neighborhood that leads to the largest improvement over the current solution in each iteration of LNS. LB is often slow since it needs to solve an ILP of the same size as input. Our proposed heuristics, LB-RELAX and its variants, use the linear programming relaxation of LB to select neighborhoods. Empirically, LB-RELAX and its variants compute as effective neighborhoods as LB but run faster. They achieve state-of-the-art anytime performance on several ILP benchmarks.

Keywords: Integer Linear Program · Large Neighborhood Search · Heuristic Search.

1 Introduction

Combinatorial optimization problems (COP) concerns a wide variety of real-world applications, including vehicle routing [43], path planning [36] and resource allocation [35] problems. Many of them are difficult to solve with limited computational resources due to their NP-Hardness. Nonetheless, the widespread importance of COPs has inspired research in designing algorithms for solving them, including exact algorithms, approximation algorithms, heuristic algorithms and data-driven algorithms.

In this paper, we focus specifically on Integer Linear Programs (ILPs) since it is a powerful tool to model and solve a broad collection of COPs, including graph optimization [41], mechanism design [11], facility location [19,4] and network design [12,21] problems. Branch-and-Bound (BnB) is an optimal and complete

tree search algorithm and is one of the state-of-the-art algorithms for ILPs [27]. It is also the core of many ILP solvers such as SCIP [8] and Gurobi [17]. Huge research effort has been made to improve it over the past decades [2]. However, BnB still falls short of delivering practical impact due to scalability issues [24,14]. On the other hand, Large Neighborhood Search (LNS) is a powerful heuristic algorithm for hard COPs and has been recently applied to solve ILPs [41,44,42] in the machine learning (ML) community.

To solve ILPs, LNS starts with an initial solution, i.e., a feasible assignment of values to the variables. It then iteratively improves the best solution found so far (i.e., the *incumbent solution*), by applying *destroy heuristics* to select a subset of variables and solving a sub-ILP that optimizes only the selected variables while leaving others fixed. ML-based destroy heuristics are shown to be efficient and effective but they are often tailored for a specific problem domain and require extensive computational resources for learning. A few non-ML destroy heuristics have been studied, such as the randomized heuristics [41,42] and the Local Branching (LB) heuristic [13,42], but they are either less efficient or effective compared to the ML-based ones. The randomized heuristics select the neighborhood by quickly randomly sampling a subset of variables which is often of bad quality. LB computes the optimal solution across all possible search neighborhoods that differs from the current incumbent solutions on a limited number of variables; however, LB is computationally expensive since it requires solving an ILP that has the same size as the original problem.

To strike a balance between efficiency and effectiveness, we propose a simple yet effective destroy heuristic LB-RELAX that is based on the linear programming (LP) relaxation of LB. Instead of solving an ILP to find the neighborhood as LB does, LB-RELAX computes its LP relaxation. It then selects the variables greedily based on the difference between the values in the incumbent solution and the LP relaxation solution. We also propose two other variants, LB-RELAX-S and LB-RELAX-R, that deploy a sampling method and combine the randomized heuristic with LB-RELAX to help escape local optima more efficiently, respectively. In experiments, we compare LB-RELAX and its variants against LNS with baseline destroy heuristics and BnB on several ILP benchmarks and show that they achieve state-of-the-art anytime performance. We also show that LB-RELAX achieves competitive results with, sometimes even outperform, the ML-based destroy heuristics. We also test LB-RELAX and its variants on selected difficult MIPLIB instances [16] that encompass diverse problem domains, structures and sizes and show that they achieve best performance on at least 40% of the instances. We also empirically show that LB-RELAX and LB-RELAX-S find neighborhoods of similar quality but is much faster than LB. They sometimes even outperform LB due to LB being too slow to find good enough neighborhoods within a reasonable time cutoff.

2 Background

In this section, we first define ILP and introduce its LP relaxation. We then introduce LNS for ILP solving and the Local Branching (LB) heuristic.

Algorithm 1 LNS for ILPs

1: **Input:** An ILP.
2: $\boldsymbol{x}^0 \leftarrow$ Find an intial solution to the input ILP
3: $t \leftarrow 0$
4: **while** time limit not exceeded **do**
5: $\quad \mathcal{X}^t \leftarrow$ Select a subset of variables to destroy
6: $\quad \boldsymbol{x}^{t+1} \leftarrow$ Solve the ILP with additional constraints $\{x_i = x_i^t : x_i \notin \mathcal{X}^t\}$
7: $\quad t \leftarrow t + 1$
8: **return** $\boldsymbol{x}^t$

2.1 ILP and its LP Relaxation

An *integer linear program (ILP)* is defined as

$$\min \boldsymbol{c}^\mathsf{T}\boldsymbol{x} \quad \text{s.t. } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \text{ and } \boldsymbol{x} \in \{0,1\}^n,$$

where $\boldsymbol{x} = (x_1, \dots, x_n)^\mathsf{T}$ denotes the n binary variables to be optimized, $\boldsymbol{c} \in \mathbb{R}^n$ denotes the vector of objective coefficients and $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$ specify m linear constraints. A *solution* to the ILP is an feasible assignment of values to the variables.

The *linear programming (LP) relaxation* of an ILP is obtained by relaxing binary variables in the ILP to continuous variables between 0 and 1, i.e., by replacing the integer constraint $\boldsymbol{x} \in \{0,1\}^n$ with $\boldsymbol{x} \in [0,1]^n$.

Note that, in this paper, we focus on the formulation above that consists of only binary variables, but our methods can also be applied to mixed integer linear programs with continuous variables and/or non-binary integer variables.

2.2 LNS for ILP solving

LNS is a heuristic algorithm that starts with an initial solution and then iteratively reoptimizes a part of the solution by applying the destroy and repair operations until a time limit is exceeded. Let $\boldsymbol{x}^0$ be the initial solution. In iteration $t \geq 0$ of the LNS, given the *incumbent solution* $\boldsymbol{x}^t$, defined as the best solution found so far, a destroy operation is done by a *destroy heuristic* where it selects a subset of k_t variables $\mathcal{X}^t = \{x_{i_1}, \dots, x_{i_{k_t}}\}$. The repair operation is done by solving a sub-ILP with $\mathcal{X}^t$ being the variables while fixing the values of $x_j \notin \mathcal{X}^t$ to be the same as in $\boldsymbol{x}^t$. Compared to BnB, LNS is more effective in improving the objective value $\boldsymbol{c}^\mathsf{T}x$, or the primal bound, especially on difficult instances [41,42,44]. Compared to other local search methods, LNS explores a large neighborhood in each step and thus, is more effective in avoiding local minima. LNS for ILPs is summarized in Algorithm 1.

2.3 LB Heuristic

The LB Heuristic [13] is originally proposed as a primal heuristic in BnB but is also applicable in LNS for ILP solving [42,32]. Given the incumbent solution $\boldsymbol{x}^t$

in iteration t of LNS, the LB heuristic [13] aims to find the subset of variables to destroy $\mathcal{X}^t$ such that it leads to the optimal $\boldsymbol{x}^{t+1}$ that differs from $\boldsymbol{x}^t$ on at most k_t variables, i.e., it computes the optimal solution $\boldsymbol{x}^{t+1}$ that sits within a given Hamming ball of radius k_t centered around $\boldsymbol{x}^t$. To find $\boldsymbol{x}^{t+1}$, the LB heuristic solves the LB ILP that is exactly the same ILP from input but with one additional constraint that limits the distance between $\boldsymbol{x}^t$ and $\boldsymbol{x}^{t+1}$:

$$\sum_{i\in[n]:x_i^t=0} x_i^{t+1} + \sum_{i\in[n]:x_i^t=1} (1 - x_i^{t+1}) \leq k_t.$$

The LB ILP is of the same size of the input ILP (i.e., it has the same number of variables and one more constraint), therefore, it is often slow to run in practice.

3 Related Work

In this section, we summarize related work on LNS for ILPs, LNS-based primal heuristics in BnB and LNS for other COPs.

3.1 LNS for ILPs

While a lot of effort has been made to improve BnB for ILPs in the past decades, LNS for ILPs has not been studied extensively in the past. Recently, Song et al. [41] show that even a randomized destroy heuristic in LNS can outperform state-of-the-art BnB in runtime. In the same paper, they show that an ML-guided decomposition-based LNS can achieve even better performance, where they apply reinforcement learning and imitation learning to learn the destroy heuristics. Since then, there have been a few more recent studies on ML-based LNS for ILPs. Sonnerat et at. [42] learn to select variables to destroy via imitating LB. Wu et al. [44] learn the same thing but they use reinforcement learning instead. The main difference between LB-RELAX and ML-based heuristics is that LB-RELAX does not require extra computational resource for learning and is agnostic to the underlying problem distributions. LB-RELAX also has a better balance between efficiency and effectiveness than those existing non-ML heuristics.

3.2 LNS-based primal heuristics in BnB

LNS-based primal heuristics is one of the rich set of primal heuristics in BnB for ILPs and many techniques have been proposed in past decades. With the same purpose of improving primal bounds of the ILPs, the main differences between the LNS-based primal heuristics in BnB and LNS for ILPs are the following: (1) Since LNS-based primal heuristics are often more expensive to run than the others in BnB, they are executed periodically at different search tree nodes during the main search and the execution schedule is itself dynamic; (2) the destroy heuristics for LNS in BnB are often designed to use information, such

as the dual bound and the LP relaxation at a search tree node, that is specific to BnB and not directly applicable in LNS for ILPs in our setting.

Next, we briefly summarize the destroy heuristics in LNS-based primal heuristics. The Crossover heuristics [38] destroy variables that have different values in a set of selected known solutions (typically two). The Mutation heuristics [38] destroys a random subset of variables. Relaxation Induced Neighborhood Search (RINS) [10] destroys variables whose values disagree in the solution of the LP relaxation at the current search tree node and the current incumbent solution. Relaxation Enforced Neighborhood Search (RENS) [7] restricts the neighborhood to be the feasible roundings of the LP relaxation at the current search tree node. Local Branching [13] restricts the neighborhood to a ball around the current incumbent solution. Distance Induced Neighborhood Search (DINS) [15] takes the intersection of the neighborhoods of the Crossover, LB and RINS heuristics. Graph-Induced Neighborhood Search (GINS) [34] destroys the breadth-first-search neighborhood of a variable in the bipartite graph representation of the ILP. An adaptive LNS primal heuristic that essentially solves a multi armed bandit problem has been proposed to combine the power of these heuristics [18].

LB-RELAX is closely related to RINS [10] since they both use LP relaxations to select neighborhoods. However, RINS is more suitable in BnB since it can adapt dynamically to the constraints added by branching. It uses the LP relaxation of the original problem, whereas LB-RELAX uses that of the LB ILP which takes into account the incumbent solutions that could change from iteration to iteration in LNS.

3.3 LNS for other COPs

LNS has been applied to solve a wide range of COPs, such as the vehicle routing problem [37,5], the traveling salesman problem [40], scheduling problems [26,45] and path planning problems [29,30,23]. Recently, ML-based methods have been applied to improve LNS for those applications [9,33,20,31,22].

4 The Local Branching Relaxation Heuristic

Recently, designing effective destroy heuristics in LNS for ILPs has been a focus in the ML community [41,42,44]. However, it is difficult to apply ML-based destroy heuristics to general ILPs since they are often customized for ILPs from certain problem distributions, e.g., graph optimization problems from a given graph distribution or scheduling problems where resources and demands follow the distribution of historical data, and require extra computational resources for training. There has been a lack of study on destroy heuristics that are agnostic to the underlying distribution of the problem. Existing ones such as randomized heuristics are simple and fast but sometimes not effective [41,42]. LB are effective but not efficient [42,32] since it exhaustively solves an ILP the same size as input for the best improvement.

Algorithm 2 LB-RELAX (LB-RELAX-S)

1: **Input:** An ILP, incumbent solution $\boldsymbol{x}^t$ and neighborhood size k_t.
2: Construct the LB ILP given $\boldsymbol{x}^t$ and k_t
3: $\bar{\boldsymbol{x}}^{t+1} \leftarrow$ Solve the LP relaxation of the LB ILP
4: $\Delta_i \leftarrow |\bar{x}_i{}^{t+1} - x_i^t|$ for all $i \in [n]$
5: $\bar{\mathcal{X}}^t \leftarrow \{x_i : \Delta_i > 0, i \in [n]\}$
6: **if** $|\bar{\mathcal{X}}^t| \geq k_t$ **then**
7: $\quad\mathcal{X}^t \leftarrow$ Select k_t variables greedily with the largest Δ_i from $\bar{\mathcal{X}}^t$
$\quad$($\mathcal{X}^t \leftarrow$ Select k_t variables uniformly at random from $\bar{\mathcal{X}}^t$)
8: **else**
9: $\quad\mathcal{X}' \leftarrow$ a random subset of $k_t - |\bar{\mathcal{X}}^t|$ variables from $\{x_i : \Delta_i = 0, i \in [n]\}$
10: $\quad\mathcal{X}^t \leftarrow \bar{\mathcal{X}}^t \cup \mathcal{X}'$
11: **return** $\mathcal{X}^t$

There are well-known approximation algorithms for NP-hard COPs based on LP relaxation [25]. Typically, they solve the LP relaxation of the ILP of the original problem and apply deterministic or randomized rounding afterwards to construct an integral solution. These algorithms often have theoretical guarantee on the effectiveness and are fast, since LP can be solved in polynomial time. Inspired by those algorithms, we propose destroy heuristic LB-RELAX that first solves the LP relaxation of the LB ILP and then constructs the neighborhood (selects variables $\mathcal{X}^t$ to destroy) based on the LP relaxation solution. Specifically, given an ILP and the incumbent solution $\boldsymbol{x}^t$ in iteration t, we construct the LB ILP with neighborhood size k_t and solve its LP relaxation. Let $\bar{\boldsymbol{x}}^{t+1}$ be the LP relaxation solution to the LB ILP. Also, let $\Delta_i = |\bar{x}_i{}^{t+1} - x_i^t|$ and $\bar{\mathcal{X}}^t = \{x_i : \Delta_i > 0, i \in [n]\}$. $\bar{\mathcal{X}}^t$ includes all the fractional variables in the LP relaxation solution and all integral variables that have different values from $\boldsymbol{x}^t$. In the following, we introduce (1) LB-RELAX, (2) LB-RELAX-S, a variant of LB-RELAX with randomized sampling and (3) LB-RELAX-R, another variant of LB-RELAX that combines a randomized destroy with LB-RELAX to help avoid local minima more effectively.

LB-RELAX first gets the LP relaxation solution $\bar{\boldsymbol{x}}^{t+1}$ of the LB ILP and then calculates Δ_i and $\bar{\mathcal{X}}^t$ from $\bar{\boldsymbol{x}}^{t+1}, \boldsymbol{x}^t$. To construct $\mathcal{X}^t$ (the set of variables to destroy), it then greedily selects k_t variables with the largest Δ_i and breaks ties uniformly at random. Intuitively, LB-RELAX greedily selects the variables whose values are more likely to change in the incumbent solution $\boldsymbol{x}^t$ after solving the LB ILP. LB-RELAX is summarized in Algorithm 2. Instead of using the LP relaxation of the LB ILP, one could argue that we alternatively use that of the original ILP similar to RINS [10]. However, the advantage of LB-RELAX over using the LP relaxation of the original problem is that, by approximating the solution to the LB ILP, LB-RELAX selects neighborhoods based on the incumbent solutions that change from iteration to iteration, whereas the original LP relaxation is a static and less informative feature that is pre-computed before the LNS procedure.

LB-RELAX-S is a variant of LB-RELAX with randomized sampling. To construct $\mathcal{X}^t$, instead of greedily choosing variables with the largest Δ_i, it selects k_t variables from $\bar{\mathcal{X}}^t$ uniformly at random. If $|\bar{\mathcal{X}}^t| < k_t$, it selects all variables from $\bar{\mathcal{X}}^t$ and $k_t - |\bar{\mathcal{X}}^t|$ variables from the remaining uniformly at random. LB-RELAX is summarized in Algorithm 2 where the parts in blue highlight the differences between LB-RELAX and LB-RELAX-S. Since $0 \leq \Delta_i \leq 1$, one could treat Δ_i as a probability distribution and sample k_t variables accordingly (see [42] for an example of how to normalize the distribution to sample k_t variables). However, this variant performs similarly to or slightly worse than LB-RELAX-S empirically and require extra hyperparameter tunings for the normalization. We therefore omit it and focus on the simpler variant in this paper.

LB-RELAX-R is another variant of LB-RELAX that leverages a randomized destroy to avoid local minima more effectively. Once LB-RELAX fails to find an improving solution in iteration t, if we let $k_{t+1} = k_t$, it will solve the exact same LP relaxation of the LB ILP again in the next iteration since the incumbent solution $\boldsymbol{x}^{t+1} = \boldsymbol{x}^t$ and the neighborhood size stay the same. Also, since LB-RELAX uses a greedy rule, it will select the same set of variables with the largest Δ_i's deterministically, except that it might need to break ties randomly in some cases when there are multiple variables with the same Δ_i. Therefore, it is susceptible to getting stuck at local minima. To tackle this issue, once LB-RELAX fails to find a new incumbent solution, we update k_{t+1} using the adaptive method described in the next paragraph. If it fails again in the next iteration, we switch to a randomized destroy heuristic that uniformly samples variables at random without replacement to construct the neighborhood. We switch back to LB-RELAX after running the randomized destroy heuristic for at least γ seconds and a new incumbent solution is found.

Next, we discuss an adaptive method to set the neighborhood size k_t for LB-RELAX and its variants. The initial neighborhood size k_0 is set to a constant or a fraction of the number of variables in the input ILP. In iteration t, if LNS finds a new incumbent solution, we let $k_{t+1} = k_t$. Otherwise, we increase k_t by a factor $\alpha > 1$. Also, we upper bound the neighborhood size k_t to a fraction $\beta < 1$ of the number of variables to make sure the sub-ILP in each iteration is not too difficult to solve, i.e., we let $k_{t+1} = \min\{\alpha \cdot k_t, \beta \cdot n\}$. This adaptive way of choosing k_t also helps address the issue of local minima by expanding the search neighborhood when LNS fails to improve the solution. It is applicable to not only LB-RELAX and its variants but also any destroy heuristics that require a given neighborhood size k_t.

5 Empirical Evaluation

In this section, we demonstrate the efficiency and effectiveness of LB-RELAX and its variants through extensive experiments on ILP benchmarks.

5.1 Setup

Instance Generation We evaluate on four NP-hard problem benchmarks selected from previous work [44,41,39], which consist of synthetic minimum vertex cover (MVC), maximum independent set (MIS), set covering (SC) and multiple knapsack (MK) instances. MVC and MIS instances are generated according to the Barabasi-Albert random graph model [3], with 9,000 nodes and average degree 5 following [41]. SC instances are generated with 4,000 variables and 5,000 constraints following [44]. MK instances are generated with 400 items and 40 knapsacks following [39]. For each problem, we generate 100 instances.

Baselines We compare LB-RELAX, LB-RELAX-R and LB-RELAX-S with the following baselines:

- BnB using SCIP (v8.0.1) as the solver with the aggressive mode turned on to focus on improving the primal bound;
- LB: LNS which selects the neighborhood with the LB heuristics;
- RANDOM: LNS which selects the neighborhood by uniformly sampling a subset of variables of a given neighborhood size k_t;
- GRAPH: LNS which selects the neighborhood based on the bipartite graph representation of the ILP similar to GINS [34]. A bipartite graph representation consists of nodes representing the variables and constraints on two sides, respectively, with an edge connecting a variable and a constraint if a variable has a non-zero coefficient in the constraint. It runs a breadth-first search starting from a random variable node in the bipartite graph and selects the first k_t variable nodes expanded.

Furthermore, we compare our approaches with state-of-the-art ML approaches:

- IL-LNS: LNS which selects the neighborhood using a GCN-based policy obtained by learning to imitate the LB heuristic [42]. We implement IL-LNS since the authors do not fully open source the code;
- RL-LNS: LNS which selects the neighborhood using a GCN-based policy obtained by reinforcement learning [44]. Note that this approach does not require a given neighborhood size k_t since the size is defined implicitly by how the trained policy is used. We use the code made available by the authors.

Hyperparameters We conduct our experiments on 2.5GHz Intel Xeon Platinum 8259CL CPUs with 32 GB RAM. All experiments use the hyperparameters described below unless stated otherwise. We use SCIP (v8.0.1) [8], the state-of-the-art open source ILP solver for the repair operations in LNS. To run LNS, we find an initial solution by running SCIP for 10 seconds for MVC, MIS and SC and 20 seconds for MK. We set the time limit to 60 minutes to solve each instance and 2 minutes for each repair operation in LNS. Except for LB, we set the time limit to 10 minutes for each repair operation since LB solves a larger ILP than other approaches in each iteration and typically requires a longer time

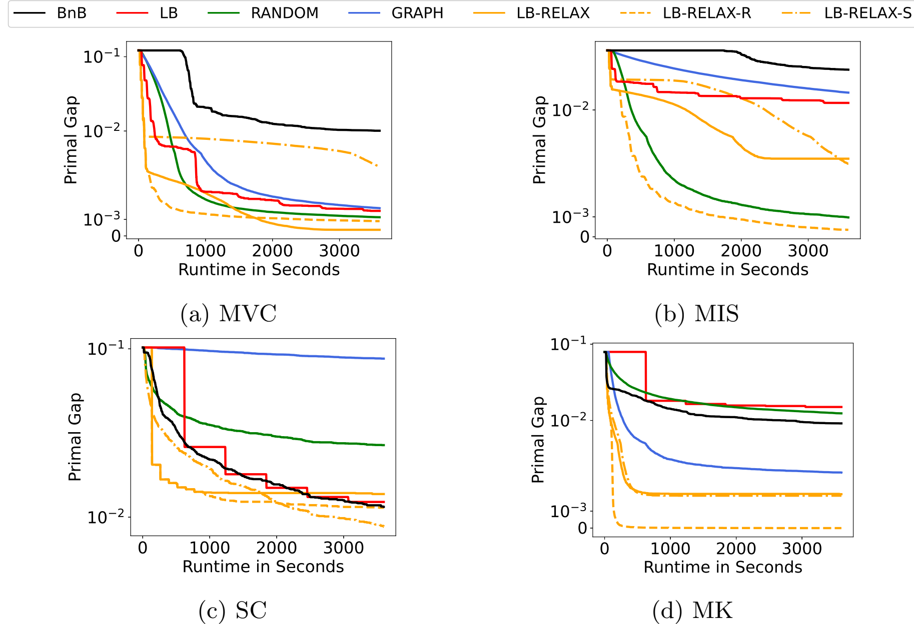
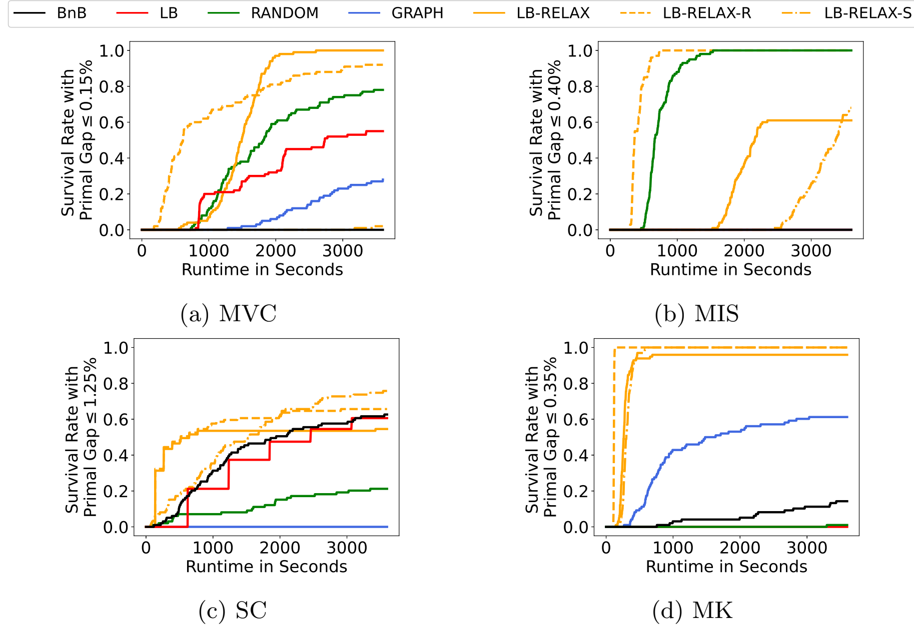


Fig. 1: Comparison with non-ML approaches: The primal gap as a function of time, averaged over 100 instances.

limit. All approaches require a neighborhood size k_t in LNS, except for BnB and RL-LNS. The initial neighborhood size (k_0) is set to $k_0 = 400, 200, 150$ and 400 for MVC, MIS, SC and MK, respectively. For fair comparison, all baselines use adaptive neighborhood sizes with $\alpha = 1.02$ and $\beta = 0.5$, except for BnB and RL-LNS. For LB-RELAX-R, we set $\gamma = 30$ seconds. Additional details on tuning hyperparameters are included in Appendix.

Metrics We use the following metrics to evaluate the efficiency and effectiveness of different approaches: (1) The *primal bound* is the objective value of the ILP. (2) The *primal gap* [6] is the normalized difference between the primal bound v and a precomputed best known objective value v^*, defined as $\frac{|v-v^*|}{\max(v,v^*,\epsilon)}$ if v exists and $v \cdot v^* \geq 0$, or 1 otherwise. We use $\epsilon = 10^{-8}$ to avoid division by zero and v^* is the best primal bound found within 60 minutes by any approach in the portfolio for comparison. (3)The *primal integral* [1] at time q is the integral on $[0, q]$ of the primal gap as a function of time. It captures the quality of and the speed at which solutions are found. (4) The *survival rate* to meet a certain primal gap threshold is the fraction of instances with the primal gap below the threshold [42]. Since BnB and LNS are both anytime algorithms, we show the metrics as a function of time or the number of iterations in LNS (when applicable) to demonstrate their anytime performance.



(a) MVC (b) MIS

(c) SC (d) MK

Fig. 2: Comparison with non-ML approaches: The survival rate over 100 instances as a function of time to meet a certain primal gap threshold. The primal gap threshold is chosen from Table 1 as the median of the average primal gaps at 60 minutes time cutoff over all approaches rounded to the nearest 0.05%.

5.2 Results

Comparison with Non-ML Approaches First, we compare LB-RELAX, LB-RELAX-R and LB-RELAX-S with non-ML approaches, namely BnB, LB, RANDOM and GRAPH. Figure 1 shows the primal gap as a function of time, averaged over 100 instances. The results show that LB-RELAX, LB-RELAX-R and LB-RELAX-S consistently improve the primal gap a lot faster than the baselines in the first few minutes of LNS. LB-RELAX improves the primal gap slightly faster than LB-RELAX-S in all cases. On average, LB-RELAX is always better than the baselines at any point of time on MK instances and LB-RELAX-S is always better than the baselines on SC and MK instances. However, both LB-RELAX and LB-RELAX-S could get stuck at some local minima. In those cases, they need some time to escape local minima by adjusting the neighborhood size and sometimes could be outperformed by some baselines with longer time on the MVC and MIS instances. By adding randomization to LB-RELAX, LB-RELAX-R escapes local minima more efficiently than LB-RELAX and LB-RELAX-S. On average, LB-RELAX-R is always better than the baselines at any point of time in the search on the MVC, MIS and MK instances.

Table 1: Primal gap (PG) (in percent) and primal integral (PI) at 60 minutes time cutoff, averaged over 100 instances, and their standard deviations.

	MVC		MIS	
	PG (%)	PI	PG (%)	PI
BnB	1.01±0.46	128.6±14.6	2.80±1.36	144.0±20.1
LB	0.15±0.08	22.1±3.6	1.20±0.31	56.3±9.4
RANDOM	0.11±0.05	32.3±2.3	0.10±0.05	18.0±2.5
GRAPH	0.17±0.04	40.8±2.5	1.56±0.18	90.2±7.6
LB-RELAX	**0.04±0.03**	10.3±1.7	0.39±0.12	29.4±4.3
LB-RELAX-R	0.09±0.04	**9.6±1.7**	**0.04±0.04**	**9.3±1.7**
LB-RELAX-S	0.42±0.20	28.8±8.1	0.37±0.11	51.7±10.1
	SC		MK	
BnB	1.15±0.98	87.4±38.6	0.91±0.59	60.7±17.9
LB	1.23±0.98	114.1±35.7	1.50±0.48	97.7±13.0
RANDOM	2.68±1.31	124.4±45.7	1.24±0.36	68.9±14.7
GRAPH	8.75±2.15	338.2±77.0	0.33±0.14	23.6±4.9
LB-RELAX	1.37±0.96	63.9±34.0	0.20±0.09	11.3±3.0
LB-RELAX-R	1.14±0.90	**58.9±31.5**	**0.00±0.00**	**3.7±0.4**
LB-RELAX-S	**0.88±0.85**	63.8±32.4	0.19±0.07	11.8±2.4

Table 2: The time (in seconds) to improve the inital solution in one iteration and the improvement of the primal bound, averaged over 100 instances. The time for LB is the solving time of the LB ILP. The time for LB-RELAX and LB-RELAX-S is the sum of the solving times of the LB relaxation and the sub-ILP. The numbers in parentheses are the speed-ups. The improvement is computed by taking the difference between the initial solution and the new incumbent solution and the numbers in parentheses are the losses in quality in percent compared to LB. ↑ means higher is better, ↓ means lower is better.

		MVC	MIS	SC	MK
LB	Time↓	40.2	56.0	600.0	600.0
	Imp.↑	129.79	65.50	12.21	216.51
LB-RELAX	Time↓	12.1 (3.3x)	19.5 (2.9x)	125.3 (4.8x)	5.87 (102.2x)
	Imp.↑	129.41 (-0.3%)	65.19 (-0.5%)	15.77 (+29.2%)	141.10 (-34.8%)
LB-RELAX-S	Time↓	12.0 (3.4x)	19.5 (2.9x)	24.51 (24.5x)	5.12 (117.6x)
	Imp.↑	128.61 (-0.9%)	62.46 (-4.6%)	5.65 (-53.7%)	113.48 (-47.6%)

Table 1 presents the average primal gap and primal integral at 60 minutes time cutoff. (See results at 15, 30 and 45 minutes time cutoff in Appendix.) On MVC, SC and MK instances, all LB-RELAX, LB-RELAX-S and LB-RELAX-R have lower primal gaps and primal integrals on average than any baselines, demonstrating that they not only find higher quality solutions but also find them at a faster speed. On MIS and MK instances, LB-RELAX-R achieves the lowest primal gap and primal integral among all approaches. It also achieves the lowest primal integral on MVC and SC instances. Overall, LB-RELAX-R always comes up in the top 2 in both metrics on all problems.

Figure 2 shows the survival rate over 100 instances as a function of time to meet a certain primal gap threshold. On MVC instances, LB-RELAX and LB-RELAX-R achieve final survival rates above 0.9 while the best baseline RANDOM stays below 0.8. On MIS instances, both LB-RELAX-R and RANDOM achieve final survival rates of 1.0 but LB-RELAX-R uses shorter time. On SC instances, LB-RELAX-S and LB-RELAX-R consistently has a higher survival

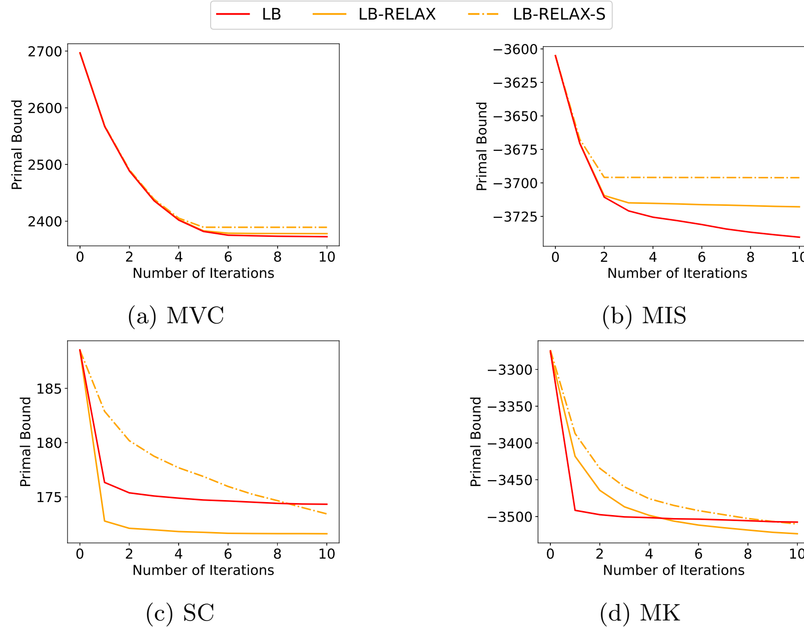


Fig. 3: Comparison with LB: The primal bound as a function of the number of iterations, averaged over 100 instances.

rate than the baselines. On MK instances, LB-RELAX and its variants achieve survival rates above 0.9 within 15 minutes while the best baseline GRAPH only gets to around 0.6 with 60 minutes.

One limitation of LB-RELAX and its variants is that they do not perform well on some problem domains, for example the maximum cut and combinatorial auction problems. Please see Appendix for more results.

Next, we run LB, LB-RELAX and LB-RELAX-S for 10 iterations to compare their effectiveness. We follow the same setup as earlier described, except that we do not use adaptive neighborhood sizes to make sure they have the same k_t in each iteration t. Note that the time limit for solving the sub-ILP in each iteration is set to 10 minutes for LB and 2 minutes for LB-RELAX and LB-RELAX-S. Table 2 shows the average time to improve the initial solutions and the average improvement of the primal bound in the first iteration of LNS. This allows us to compare how closely LB-RELAX and LB-RELAX-S approximate the quality of the neighborhood selected by LB and study the trade-off between quality and time. Compared to LB, LB-RELAX and LB-RELAX-S have 2.9x-117.6x speed-up but only lose at most 53.7% in quality. In particular, on MVC and MIS instances, both LB-RELAX and LB-RELAX-S lose 0.5% to 4.6% in quality but have at least 2.9x speed-up; on SC instances, LB-RELAX even gains 29.2% in quality and save 79.1% in time, due to LB cannot find a good enough neighborhood within its time limit.

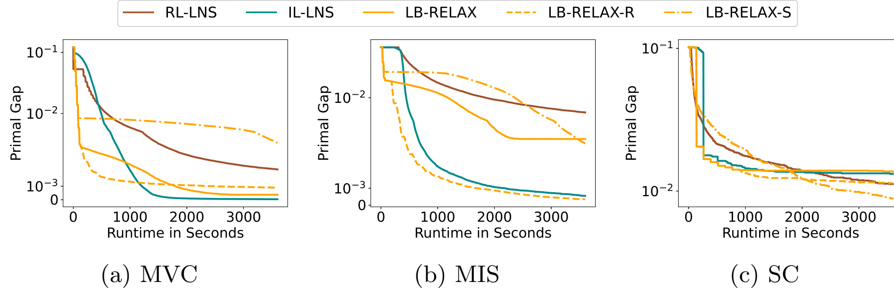


Fig. 4: Comparison with ML approaches: The primal bound as a function of time, averaged over 100 instances.

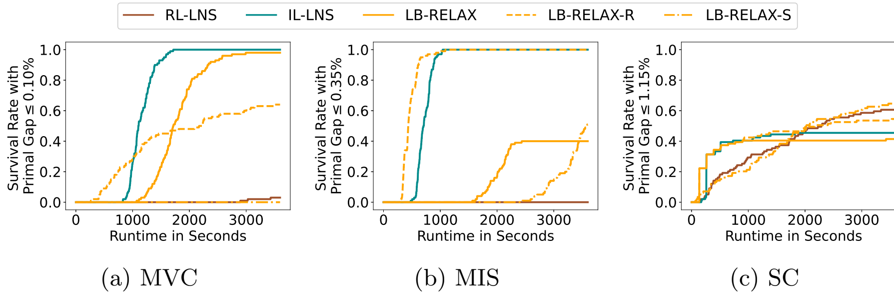


Fig. 5: Comparison with ML approaches: The survival rate over 100 instances as a function of time to meet a certain primal gap threshold. The primal gap thresholds are chosen in the same way as Fig. 2.

In Figure 3, we show the primal bound as a function of the number of iterations. It allows comparing the effectiveness of different heuristics independently of their speed. On the MVC instances, both LB-RELAX and LB-RELAX-S perform similarly to but slightly worse than LB. On the SC and MK instances, LB-RELAX achieves better performance than LB, again due to scalability issues of LB, and LB-RELAX-S achieves competitive performance with LB after 10 iterations. However on the MIS instances, both LB-RELAX and LB-RELAX-S are able to quickly improve the primal bound in the first 2-3 iterations, but afterwards converge to local minima and the gaps between them and LB increase. To complete the first 10 iterations, both LB-RELAX and LB-RELAX-S take less than 21 minutes on SC instances and 3.3 minutes on the others, while LB takes at least 57 minutes and sometimes up to 100 minutes.

Comparison with ML Approaches Then, we compare LB-RELAX, LB-RELAX-R and LB-RELAX-S on MVC, MIS and SC instances with ML approaches, namely IL-LNS and RL-LNS. Figure 4 shows the primal gap as a function of time averaged over 100 instances. The results show that LB-RELAX, LB-

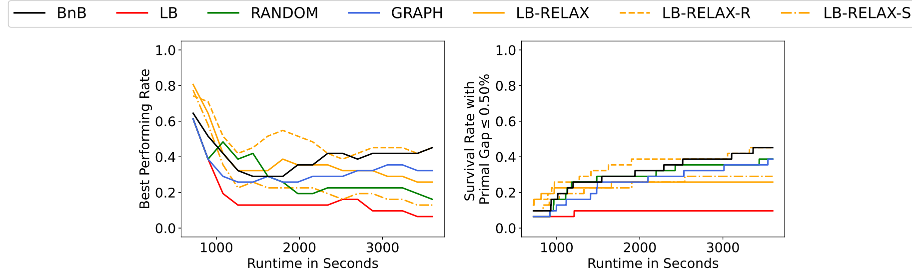


Fig. 6: Results on 31 selected MIPLIB instances: The best performing rate as a function of time (left) and the survival rate over 31 instances as a function of time to meet the primal gap threshold 0.50% (right).

RELAX-R and LB-RELAX-S consistently improve the primal bound a lot faster than IL-LNS and RL-LNS in the first few minutes of LNS. On MVC instances, IL-LNS surpasses LB-RELAX-R with the smallest average primal gap best after 20 minutes and achieve (close-to-)zero gaps after 30 minutes. On MIS instances, LB-RELAX-R has a smaller gap than both IL-LNS and RL-LNS throughout the first 60 minutes. On SC instances, IL-LNS is very competitive with LB-RELAX and converges to a similar but slightly higher gap than LB-RELAX-R and LB-RELAX-S; RL-LNS converges to almost the same primal gap as LB-RELAX-R on average but is worse than the best performer LB-RELAX-S. Overall, LB-RELAX and its variants, that do not require extra computational resources for training, are competitive with and more often even better than state-of-the-art ML approaches, suggesting that they are agnostic to the distributions of the instances and easily applicable to different problem domains.

Results on Selected MIPLIB Instances Finally, we examine how well LB-RELAX and its variants perform on ILPs that are diverse in structures and sizes. We test them on the MIPLIB dataset [16]. MIPLIB contains COPs from various real-world domains. We follow a procedure similar to [44] to filter out instances where we first filter to retain ILP instances with only binary variables. Among these, we select instances that are not too easy to solve but relatively easy to find a feasible solution for. Specifically, we filter out those that BnB can optimally solve within 3 hours (too easy) or BnB cannot find any solutions within 10 minutes (too hard), which gives us 35 instances. For all LNS approaches, we run BnB for 10 minutes to find the initial solution and set the time limit to 10 minutes for each repair operation. The initial neighborhood size k_0 is set to 20% of the number of binary variables. We compare LB-RELAX, LB-RELAX-R and LB-RELAX-S with the non-ML baselines. We further filter out 4 instances that no approach can find a better solution than the initial one, which finally gives us 31 instances.

Figure 6 shows the winning rate as a function of time for each approach on the 31 instances. The best performing rate at a time q for an approach is the

fraction of instances on which it achieves the best performance (including ties) compared to all approaches in the portfolio. LB-RELAX, LB-RELAX-R and LB-RELAX-S achieve the best performance with less than 1000 seconds on 25, 23 and 24 instances out of 35, respectively. LB-RELAX-R has the highest best performing rates at different time cutoffs and ties with BnB at 14 instances at the 60-minute mark. Figure 6 also shows the survival rate over the 31 instances as a function of time to meet the primal gap threshold 0.50%. It demonstrates that RANDOM, GRAPH and BnB are competitive with our approaches but overall LB-RELAX-R has the highest survival rate over time. On some instances, LB-RELAX and its variants can significantly outperform the baselines and we show the anytime performance on those in Appendix.

6 Conclusion

In this paper, we focused on designing effective and efficient destroy heuristics to select neighborhoods in LNS for ILPs. LB is an effective destroy heuristic but is slow to run. We therefore proposed LB-RELAX, LB-RELAX-S and LB-RELAX-R to approximate LB's decisions by solving its LP relaxation that is a lot faster to run. Empirically, we showed that LB-RELAX, LB-RELAX-S and LB-RELAX-R efficiently selected almost as effective neighborhoods as LB and achieved state-of-the-art performance when compared against non-ML and ML approaches. One limitation of our approaches is that they do not work well on some problem domains, however we showed that they still outperformed the baselines on 14 to 25 (depending on the time cutoff) out of 31 difficult MIPLIB instances that are diverse in problem domains, structures and sizes. The other limitation is that they can get stuck at local minima. To address this issue, we proposed techniques to randomize the heuristics and adaptively adjust the neighborhood sizes. For future work, one could improve LB-RELAX and its variants to make them applicable on more problem domains. In addition, instead of using hard-coded rules for scheduling the randomized heuristic in LB-RELAX-R, one could use adaptive LNS to select destroy heuristics to run. It is also future work to develop theoretical claims to help support and explain the effectiveness of LB-RELAX, LB-RELAX-S, LB-RELAX-R and possibly their other variants.

Acknowledgements. This paper reports on research done while Taoan Huang and Aaron Ferber were interns at Meta AI (FAIR). The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant number 2112533.

References

1. Achterberg, T., Berthold, T., Hendel, G.: Rounding and propagation heuristics for mixed integer programming. In: Operations research proceedings 2011, pp. 71–76. Springer (2012)

2. Achterberg, T., Wunderling, R.: Mixed integer programming: Analyzing 12 years of progress. In: Facets of combinatorial optimization, pp. 449–481. Springer (2013)
3. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Reviews of modern physics **74**(1), 47 (2002)
4. Amaral, A.R.: An exact approach to the one-dimensional facility layout problem. Operations research **56**(4), 1026–1033 (2008)
5. Azi, N., Gendreau, M., Potvin, J.Y.: An adaptive large neighborhood search for a vehicle routing problem with multiple routes. Computers & Operations Research **41**, 167–173 (2014)
6. Berthold, T.: Primal heuristics for mixed integer programs. Ph.D. thesis, Zuse Institute Berlin (ZIB) (2006)
7. Berthold, T.: Rens. Mathematical Programming Computation **6**(1), 33–54 (2014)
8. Bestuzheva, K., Besançon, M., Chen, W.K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., Gottwald, L., Graczyk, C., Halbig, K., Hoen, A., Hojny, C., van der Hulst, R., Koch, T., Lübbecke, M., Maher, S.J., Matter, F., Mühmer, E., Müller, B., Pfetsch, M.E., Rehfeldt, D., Schlein, S., Schlösser, F., Serrano, F., Shinano, Y., Sofranac, B., Turner, M., Vigerske, S., Wegscheider, F., Wellner, P., Weninger, D., Witzig, J.: The SCIP Optimization Suite 8.0. Technical report, Optimization Online (December 2021), http://www.optimization-online.org/DB_HTML/2021/12/8728.html
9. Chen, X., Tian, Y.: Learning to perform local rewriting for combinatorial optimization. Advances in Neural Information Processing Systems **32** (2019)
10. Danna, E., Rothberg, E., Pape, C.L.: Exploring relaxation induced neighborhoods to improve mip solutions. Mathematical Programming **102**(1), 71–90 (2005)
11. De Vries, S., Vohra, R.V.: Combinatorial auctions: A survey. INFORMS Journal on computing **15**(3), 284–309 (2003)
12. Dilkina, B., Gomes, C.P.: Solving connected subgraph problems in wildlife conservation. In: CPAIOR. vol. 6140, pp. 102–116. Springer (2010)
13. Fischetti, M., Lodi, A.: Local branching. Mathematical programming **98**(1), 23–47 (2003)
14. Gasse, M., Chételat, D., Ferroni, N., Charlin, L., Lodi, A.: Exact combinatorial optimization with graph convolutional neural networks. Advances in Neural Information Processing Systems **32** (2019)
15. Ghosh, S.: Dins, a mip improvement heuristic. In: International Conference on Integer Programming and Combinatorial Optimization. pp. 310–323. Springer (2007)
16. Gleixner, A., Hendel, G., Gamrath, G., Achterberg, T., Bastubbe, M., Berthold, T., Christophel, P.M., Jarck, K., Koch, T., Linderoth, J., Lübbecke, M., Mittelmann, H.D., Ozyurt, D., Ralphs, T.K., Salvagnin, D., Shinano, Y.: MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. Mathematical Programming Computation (2021). https://doi.org/10.1007/s12532-020-00194-3, https://doi.org/10.1007/s12532-020-00194-3
17. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2022), https://www.gurobi.com
18. Hendel, G.: Adaptive large neighborhood search for mixed integer programming. Mathematical Programming Computation **14**(2), 185–221 (2022)
19. Heragu, S.S., Kusiak, A.: Efficient models for the facility layout problem. European Journal of Operational Research **53**(1), 1–13 (1991)
20. Hottung, A., Tierney, K.: Neural large neighborhood search for the capacitated vehicle routing problem. In: ECAI 2020, pp. 443–450. IOS Press (2020)

21. Huang, T., Dilkina, B.: Enhancing seismic resilience of water pipe networks. In: Proceedings of the 3rd ACM SIGCAS Conference on Computing and Sustainable Societies. pp. 44–52 (2020)
22. Huang, T., Li, J., Koenig, S., Dilkina, B.: Anytime multi-agent path finding via machine learning-guided large neighborhood search. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). pp. 9368–9376 (2022)
23. Huang, T., Shivashankar, V., Caldara, M., Durham, J., Li, J., Dilkina, B., Koenig, S.: Deadline-aware multi-agent tour planning. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) (2023)
24. Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., Dilkina, B.: Learning to branch in mixed integer programming. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 30 (2016)
25. Kleinberg, J., Tardos, E.: Algorithm design. Pearson Education India (2006)
26. Kovacs, A.A., Parragh, S.N., Doerner, K.F., Hartl, R.F.: Adaptive large neighborhood search for service technician routing and scheduling problems. Journal of scheduling **15**(5), 579–600 (2012)
27. Land, A.H., Doig, A.G.: An automatic method for solving discrete programming problems. In: 50 Years of Integer Programming 1958-2008, pp. 105–132. Springer (2010)
28. Leyton-Brown, K., Pearson, M., Shoham, Y.: Towards a universal test suite for combinatorial auction algorithms. In: Proceedings of the 2nd ACM conference on Electronic commerce. pp. 66–76 (2000)
29. Li, J., Chen, Z., Harabor, D., Stuckey, P.J., Koenig, S.: Anytime multi-agent path finding via large neighborhood search. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 4127–4135 (2021)
30. Li, J., Chen, Z., Harabor, D., Stuckey, P.J., Koenig, S.: MAPF-LNS2: Fast repairing for multi-agent path finding via large neighborhood search. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). pp. 10256–10265 (2022)
31. Li, S., Yan, Z., Wu, C.: Learning to delegate for large-scale vehicle routing. Advances in Neural Information Processing Systems **34**, 26198–26211 (2021)
32. Liu, D., Fischetti, M., Lodi, A.: Revisiting local branching with a machine learning lens
33. Lu, H., Zhang, X., Yang, S.: A learning-based iterative method for solving vehicle routing problems. In: International conference on learning representations (2019)
34. Maher, S.J., Fischer, T., Gally, T., Gamrath, G., Gleixner, A., Gottwald, R.L., Hendel, G., Koch, T., Lübbecke, M., Miltenberger, M., et al.: The scip optimization suite 4.0 (2017)
35. Manne, A.S.: On the job-shop scheduling problem. Operations research **8**(2), 219–223 (1960)
36. Pohl, I.: Heuristic search viewed as path finding in a graph. Artificial intelligence **1**(3-4), 193–204 (1970)
37. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation science **40**(4), 455–472 (2006)
38. Rothberg, E.: An evolutionary algorithm for polishing mixed integer programming solutions. INFORMS Journal on Computing **19**(4), 534–541 (2007)
39. Scavuzzo, L., Chen, F.Y., Chételat, D., Gasse, M., Lodi, A., Yorke-Smith, N., Aardal, K.: Learning to branch with tree mdps. arXiv preprint arXiv:2205.11107 (2022)

40. Smith, S.L., Imeson, F.: Glns: An effective large neighborhood search heuristic for the generalized traveling salesman problem. Computers & Operations Research **87**, 1–19 (2017)
41. Song, J., Yue, Y., Dilkina, B., et al.: A general large neighborhood search framework for solving integer linear programs. Advances in Neural Information Processing Systems **33**, 20012–20023 (2020)
42. Sonnerat, N., Wang, P., Ktena, I., Bartunov, S., Nair, V.: Learning a large neighborhood search algorithm for mixed integer programs. arXiv preprint arXiv:2107.10201 (2021)
43. Toth, P., Vigo, D.: The vehicle routing problem. SIAM (2002)
44. Wu, Y., Song, W., Cao, Z., Zhang, J.: Learning large neighborhood search policy for integer programming. Advances in Neural Information Processing Systems **34**, 30075–30087 (2021)
45. Žulj, I., Kramer, S., Schneider, M.: A hybrid of adaptive large neighborhood search and tabu search for the order-batching problem. European Journal of Operational Research **264**(2), 653–664 (2018)

Appendix

A Details on Tuning Hyperparameters

For β which upper bounds the neighborhood size, we tried different values from $\{0.25, 0.5, 0.6, 0.7\}$. $\beta = 0.25$ is the worst for all approaches, resulting in the highest primal gap. For LB-RELAX and its variants, IL-LNS and LB, all values perform similarly (because they select effective neighborhoods early in the search and their neighborhood sizes either do not reach the upper bounds or they already converge to good solutions before reaching it). For RANDOM and GRAPH, beta=0.5 is the best for them. So we set beta=0.5 consistently for all approaches.

For initial neighborhood sizes k_0, it is sensitive for approaches that need long runtime to select variables, such as LB-RELAX and its variants, LB and IL-LNS (they need to solve an LP relaxation, an LB ILP and run a forward pass of GCN in each iteration, respectively). They need the right k_0 from the beginning and we fine tune it for them. For RANDOM and GRAPH, their runtime for selecting

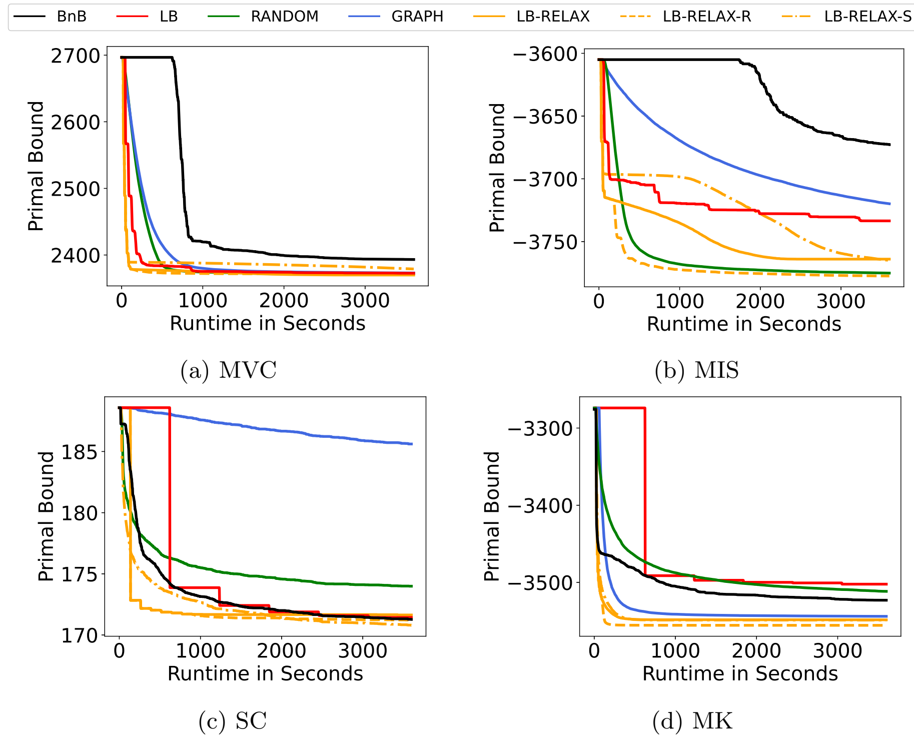


Fig. 7: Comparison with non-ML approaches: The primal bound as a function of time as a function of time, averaged over 100 instances.

Table 3: Primal gap (PG) (in percent) and primal integral (PI) at 15 minutes time cutoff, averaged over 100 instances, and their standard deviations.

	MVC		MIS	
	PG (%)	PI	PG (%)	PI
BnB	2.12±1.48	93.3±6.2	4.60±0.31	43.2±2.8
LB	0.30±0.15	16.7±1.5	1.58±0.28	19.5±2.7
RANDOM	0.24±0.07	28.3±1.7	0.32±0.10	13.6±1.3
GRAPH	0.53±0.09	34.0±1.9	3.01±0.26	33.3±2.4
LB-RELAX	0.28±0.08	7.8±0.8	1.22±0.19	14.5±1.9
LB-RELAX-R	**0.14±0.05**	**6.7±0.6**	**0.18±0.08**	**7.0±0.9**
LB-RELAX-S	0.80±0.26	11.7±2.2	2.14±0.42	20.3±3.7
	SC		MK	
BnB	2.30±1.36	46.0±13.1	1.49±0.67	31.5±3.6
LB	2.61±1.32	70.7±15.8	1.81±0.45	54.6±6.5
RANDOM	3.60±1.45	43.7±14.0	1.99±0.45	28.7±4.9
GRAPH	9.78±2.19	90.0±20.0	0.41±0.14	14.1±1.9
LB-RELAX	**1.40±0.98**	**26.3±8.8**	0.20±0.09	5.8±0.8
LB-RELAX-R	**1.40±0.98**	**26.3±8.8**	**0.00±0.01**	**3.7±0.4**
LB-RELAX-S	2.04±1.26	30.1±11.7	0.19±0.07	6.6±0.7

Table 4: Primal gap (PG) (in percent) and primal integral (PI) at 30 minutes time cutoff, averaged over 100 instances, and their standard deviations.

	MVC		MIS	
	PG (%)	PI	PG (%)	PI
BnB	1.36±0.57	108.5±9.0	4.51±0.55	84.6±5.6
LB	0.22±0.12	18.9±2.2	1.43±0.33	32.9±4.9
RANDOM	0.15±0.05	29.9±1.8	0.18±0.07	15.7±1.8
GRAPH	0.26±0.07	37.1±2.1	2.27±0.22	56.6±4.4
LB-RELAX	**0.10±0.05**	9.5±1.3	0.52±0.12	22.1±3.1
LB-RELAX-R	0.11±0.05	**7.8±1.0**	**0.10±0.05**	**8.2±1.3**
LB-RELAX-S	0.70±0.23	18.5±4.3	1.52±0.26	37.4±6.8
	SC		MK	
BnB	1.65±1.22	63.2±22.8	1.11±0.62	42.7±8.3
LB	1.80±1.13	89.6±22.6	1.64±0.46	69.9±7.5
RANDOM	3.09±1.38	73.5±24.9	1.54±0.42	44.3±8.4
GRAPH	9.34±2.30	175.9±39.1	0.36±0.14	17.5±2.8
LB-RELAX	1.39±0.97	38.9±16.9	0.20±0.09	7.7±1.5
LB-RELAX-R	**1.23±0.91**	**37.7±16.2**	**0.00±0.00**	**3.7±0.4**
LB-RELAX-S	1.37±1.04	44.8±19.9	0.19±0.07	8.4±1.2

Table 5: Primal gap (PG) (in percent) and primal integral (PI) at 45 minutes time cutoff, averaged over 100 instances, and their standard deviations.

	MVC		MIS	
	PG (%)	PI	PG (%)	PI
BnB	1.08±0.48	119.3±11.7	3.09±1.22	117.8±10.6
LB	0.18±0.10	20.7±3.0	1.29±0.31	45.1±7.2
RANDOM	0.13±0.05	31.2±2.0	0.12±0.06	17.0±2.2
GRAPH	0.20±0.05	39.1±2.3	1.84±0.21	75.0±6.0
LB-RELAX	**0.04±0.03**	10.0±1.5	0.39±0.12	25.9±3.6
LB-RELAX-R	0.10±0.05	**8.7±1.3**	**0.06±0.04**	**8.9±1.5**
LB-RELAX-S	0.59±0.21	24.3±6.3	0.68±0.23	47.3±9.1
	SC		MK	
BnB	1.29±1.03	76.4±31.1	1.00±0.61	52.2±13.1
LB	1.32±0.99	102.7±28.9	1.55±0.47	84.0±9.8
RANDOM	2.78±1.30	99.8±35.5	1.36±0.38	57.3±11.6
GRAPH	9.01±2.24	258.4±58.5	0.34±0.13	20.6±3.8
LB-RELAX	1.39±0.97	51.4±25.2	0.20±0.09	9.5±2.2
LB-RELAX-R	1.17±0.89	**48.5±23.9**	**0.00±0.00**	**3.7±0.4**
LB-RELAX-S	**1.01±0.89**	55.1±26.3	0.19±0.07	10.1±1.8

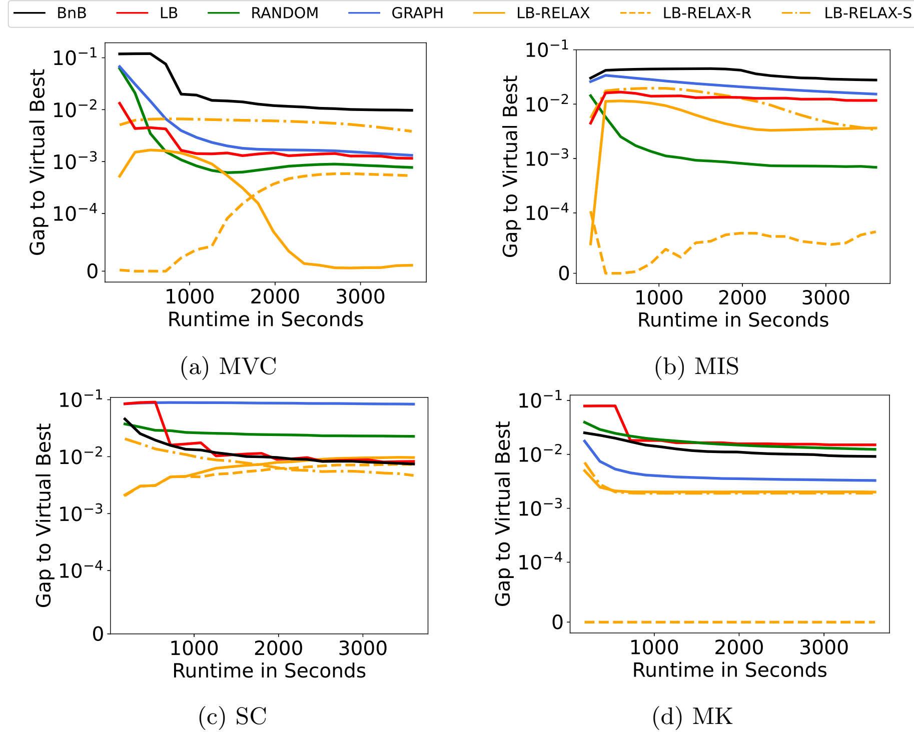


Fig. 8: Comparison with non-ML approaches: The gap to virtual best as a function of time, averaged over 100 instances.

variables is short, and with the adaptive neighborhood size mechanism, they could very quickly find the right neighborhood size and are insensitive to k_0. They converge to the same primal gaps ($< 1\%$ relative differences) with similar primal integrals ($< 2\%$ relative differences) using different k_0.

For α that controls the rate at which k_t increases, we tried values from $\{1, 1.01, 1.02, 1.05\}$. Overall, α does not have a big impact on the performance if $\alpha > 1$, however $\alpha = 1$ is far worse than the others.

For the runtime limit for repair operations, we tried 0.5, 1, 2 and 5 minutes. All approaches are not sensitive to it since most repairs are finished within 20 seconds. Except for LB-RELAX and IL-LNS on SC instances, they both select neighborhoods that require a longer time to repair and a 2-minute runtime limit is necessary and best for them. We therefore use 2 minutes consistently.

B Additional Comparison with Non-ML Approaches

We present additional results on comparison with non-ML approaches.

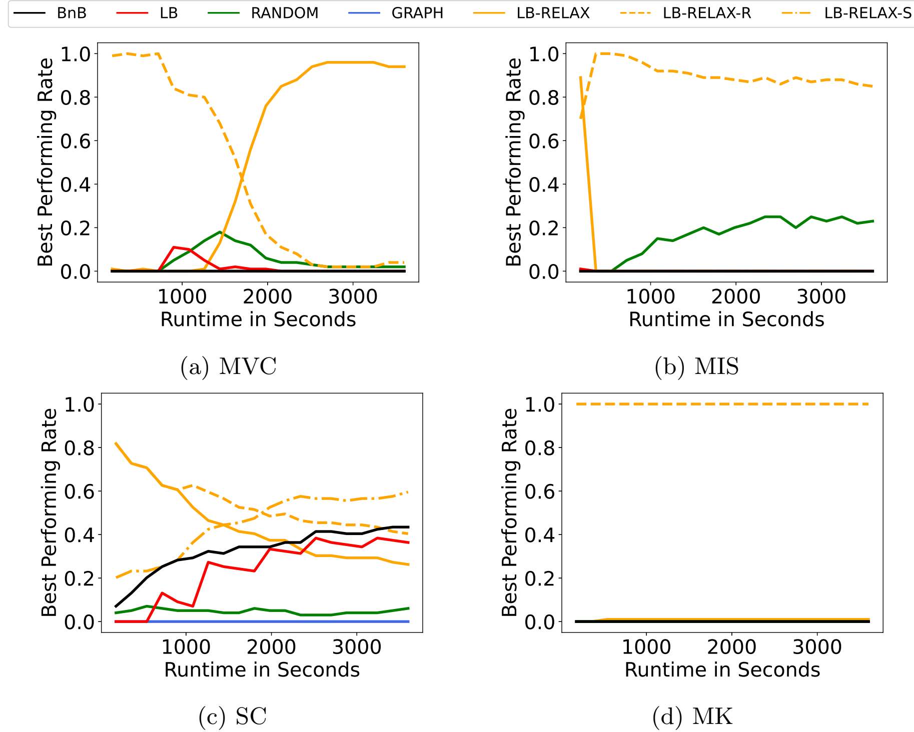


Fig. 9: Comparison with non-ML approaches: The best performing rate as a function of time, calculated over 100 instances. The best performing rate of an approach is the fraction of instances on which it has the best primal bound (including ties) at a given time cutoff among all approaches. The sum of the best performing rates at a given time might sum up greater than 1 since ties are counted multiple times.

Figure 7 shows the primal gap as a function of time, averaged over 100 instances.

Tables 3, 4 and 5 present the average primal gap and primal integral at 15, 30 and 45 minutes time cutoff, respectively.

We also introduce another metric for comparison: the *gap to virtual best* of an approach at time q. It is defined as the normalized difference between its best primal bound found up to time q and the best primal bound found up to time q by any approach in the portfolio for comparison, similar to the primal gap. Figure 8 shows the gap to virtual best as a function of time, averaged over 100 instances.

Figure 9 shows the winning rate as a function of time, averaged over 100 instances. The best performing rate at time q for an approach is the fraction of

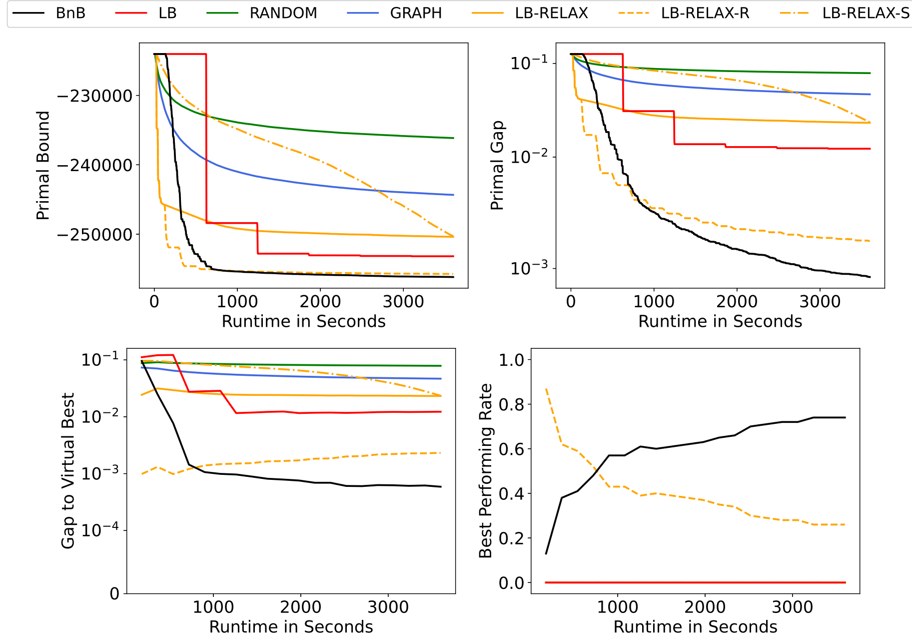
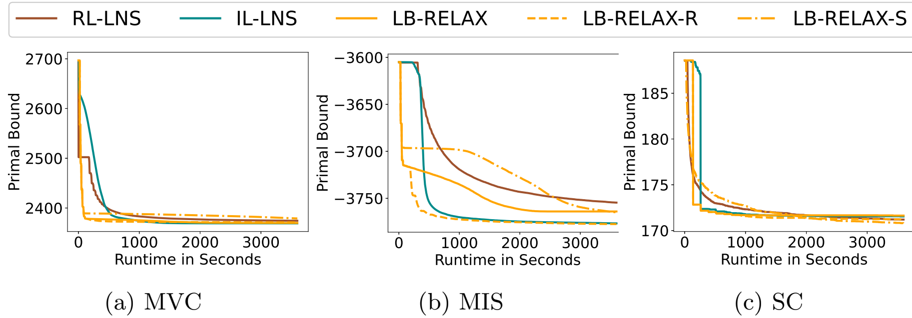


Fig. 10: Results on CAT instances.



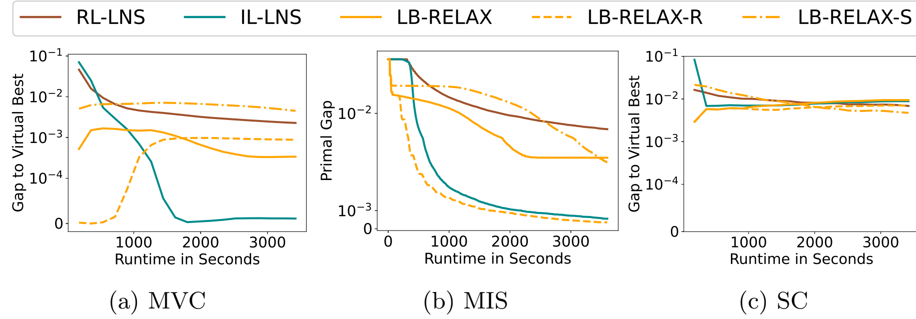
(a) MVC (b) MIS (c) SC

Fig. 11: Comparison with ML approaches: The primal bound as a function of time, averaged over 100 instances.

instances on which it achieves the best performance (including ties) compared to all approaches in the portfolio.

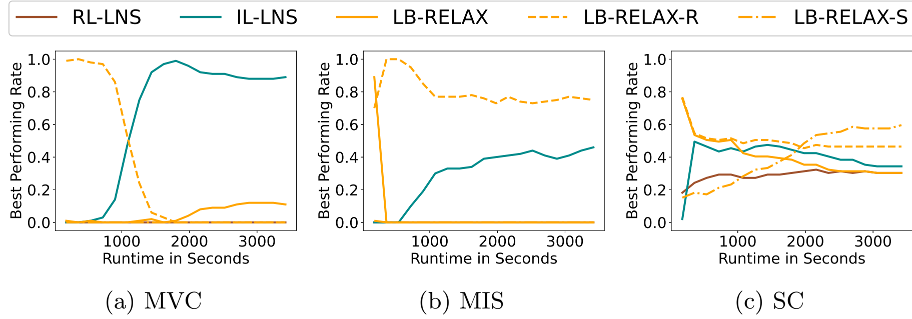
C Experimental Results on CAT Instances

We generate 100 combinatorial auction (CAT) instances with 4000 items and 8,000 bids according to arbitrary relationships in [28]. The main results are shown



(a) MVC (b) MIS (c) SC

Fig. 12: Comparison with ML approaches: The gap to virtual best as a function of time, averaged over 100 instances.



(a) MVC (b) MIS (c) SC

Fig. 13: Comparison with ML approaches: The gap to virtual best as a function of time, averaged over 100 instances.

in Figure 10. We show that LB-RELAX can improve the primal bound fast early in the search but get stuck at local minima and afterwards get outperformed by BnB and LB. LB-RELAX-R can help escape the local minima but on average converges to slightly worse solutions than BnB.

D Additional Comparison with ML Approaches

We present additional results on comparison with ML approaches. Figure 11 shows the primal gap as a function of time, averaged over 100 instances. Figure 12 shows the gap to virtual best as a function of time, averaged over 100 instances. Figure 13 shows the winning rate as a function of time, averaged over 100 instances.

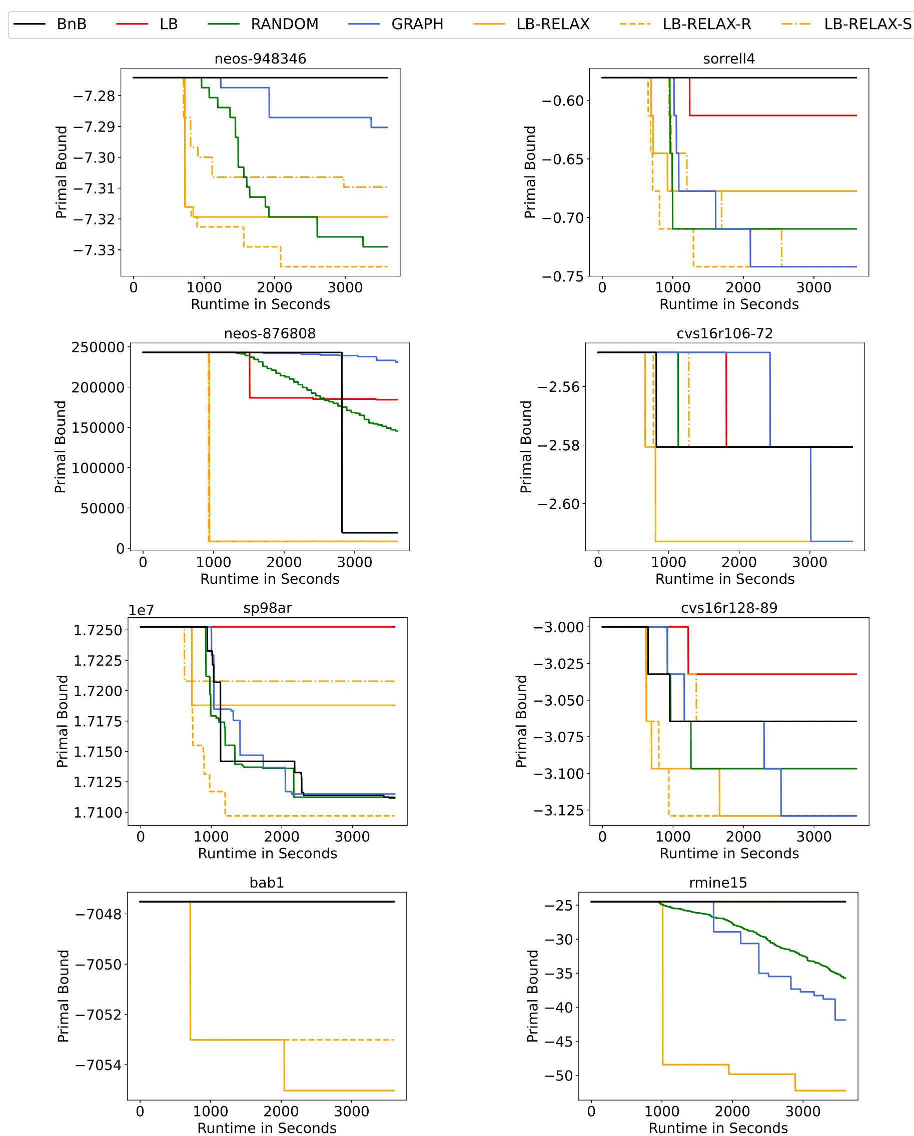


Fig. 14: The primal bound as a function of time on selected MIPLIB instances. The titles of the plots are the instance names for which the results are presented.

E Additional Results on Selected MIPLIB Instances

On some instances, LB-RELAX and its variants can significantly outperform the baselines and we present the anytime performance on those. Figure 14 shows the primal bound as a function of time on 8 selected MIPLIB instances.