Searching Large Neighborhoods for Integer Linear Programs with Contrastive Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Integer Linear Programs (ILPs) are powerful tools for modeling and solving many combinatorial optimization problems. Recently, it has been shown that Large Neighborhood Search (LNS), as a heuristic algorithm, can find high-quality solutions to ILPs faster than Branch and Bound. However, how to find the right heuristics to maximize the performance of LNS remains an open problem. In this paper, we propose a novel approach, CL-LNS, that delivers state-of-the-art anytime performance on several ILP benchmarks measured by metrics including the primal gap, the primal integral, survival rates and the best performing rate. Specifically, CL-LNS collects positive and negative solution samples from an expert heuristic that is slow to compute and learns a more efficient one with contrastive learning.

1 Introduction

Algorithm designs for combinatorial optimization problems (COPs) are important and challenging tasks. A wide variety of real-world problems are COPs, such as vehicle routing [73], path planning [61] and resource allocation [57] problems, and a majority of them are NP-hard to solve. In the past few decades, algorithms, including optimal algorithms, approximation algorithms and heuristic algorithms, have been studied extensively due to the importance of COPs. Those algorithms are mostly designed by humans through costly processes that often require a deep understanding of the problem domains and their underlying structures as well as considerable time and effort. Recently, there has been an increased interest in automating algorithm designs for COPs with machine learning (ML). Many ML approaches learn to either construct or improve solutions within an algorithmic framework, such as greedy search, local search or tree search, for a specific COP, such as the traveling salesman problem (TSP) [75, 80], vehicle routing problem (VRP) [45] or independent set problem [53], and are often not easily applicable to other COPs.

In contrast, Integer Linear Programs (ILPs) can flexibly encode and solve a broad family of COPs, such as network design [39, 15, 32], mechanism design [14], facility location [28, 3] problems. ILPs can be solved by Branch and Bound (BnB) [48], an optimal tree search algorithm that can achieve state-of-the-art for ILPs. Over the past decades, BnB has been improved tremendously to become the core of many popular ILP solvers such as SCIP [8] and Gurobi [24]. However, due to its exhaustive search nature, it is hard for BnB to scale to large instances [40, 21].

On the other hand, Large Neighborhood Search (LNS) has been shown to find high-quality solutions much faster than BnB for large ILP instances [68, 74, 69, 36]. LNS starts from an initial solution (i.e., a feasible assignment of values to variables) and then improves the current best solution by iteratively picking a subset of variables to reoptimize while leaving others fixed. Picking which subset to reoptimize, i.e., the *destroy heuristic*, is a critical component in LNS. Hand-crafted destroy heuristics, such as the randomized heuristic [68, 69] and the Local Branching (LB) heuristic [20], are often either inefficient (slow to find good subsets) or ineffective (find subsets of bad quality). ML-based

Submitted to ICML 2023 Workshop: Sampling and Optimization in Discrete Space. Do not distribute.

destroy heuristics have also been proposed and outperformed hand-crafted ones. State-of-the-art approaches include IL-LNS [69] that uses imitation learning (IL) to imitate the LB heuristic and RL-LNS [74] that uses a similar framework to IL-LNS but trained with reinforcement learning (RL).

In this paper, we propose a novel ML-based LNS for ILPs, namely *CL-LNS*, that uses contrastive learning (CL) [10, 43] to learn efficient and effective destroy heuristics. Similar to IL-LNS [69], we learn to imitate the *Local Branching (LB)* heuristic, a destroy heuristic that selects the optimal subset of variables within the Hamming ball of the incumbent solutions. LB requires solving another ILP with the same size as the original problem and thus is computationally expensive. We not only use the optimal subsets provided by LB as the expert demonstration (as in IL-LNS) but also leverage intermediate solutions and perturbations. When solving the ILP for LB, intermediate solutions are found and those that are close to optimal in terms of effectiveness become *positive samples*. We also collect *negative samples* by randomly perturbing the optimal subset. With both positive and negative samples, instead of a classification loss as in IL-LNS, we use a contrastive loss that encourages the model to predict the subset similar to the positive samples but dissimilar to the negative ones with similarity measured by dot products [59, 26]. Finally, we also use a richer set of features and graph attention networks (GAT) instead of GCN to further boost performance.

Empirically, we show that CL-LNS outperforms state-of-the-art ML and non-ML approaches at different runtime cutoffs ranging from a few minutes to an hour in terms of multiple metrics, including the primal gap, the primal integral, the best performing rate and the survival rate, demonstrating the effectiveness and efficiency of CL-LNS. In addition, CL-LNS shows great generalization performance on test instances two times larger than training instances.

2 Background

2.1 ILPs

An *integer linear program (ILP)* is defined as $\min \boldsymbol{c}^\mathsf{T}\boldsymbol{x}$ s.t. $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ and $\boldsymbol{x} \in \{0,1\}^n$, where $\boldsymbol{x} = (x_1, \ldots, x_n)^\mathsf{T}$ denotes the n binary variables to be optimized, $\boldsymbol{c} \in \mathbb{R}^n$ is the vector of objective coefficients, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$ specify m linear constraints. A *solution* to the ILP is a feasible assignment of values to the variables. In this paper, we focus on the formulation above that consists of only binary variables, but our methods can 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 destroys and reoptimizes a part of the solution until a runtime limit is exceeded or some stopping condition is met. Let $\mathcal{I} = (\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ be the input ILP, where $\boldsymbol{A}, \boldsymbol{b}$ and $\boldsymbol{c}$ are the coefficients defined in Equation (**??**), and $\boldsymbol{x}^0$ be the initial solution (typically found by running BnB for a short runtime). In iteration $t \geq 0$ of LNS, given the *incumbent solution* $\boldsymbol{x}^t$, defined as the best solution found so far, a *destroy heuristic* selects a subset of k^t variables $\mathcal{X}^t = \{x_{i_1}, \ldots, x_{i_{k^t}}\}$. The reoptimization is done by solving a sub-ILP with $\mathcal{X}^t$ being the variables while fixing the values of $x_j \notin \mathcal{X}^t$ the same as in $\boldsymbol{x}^t$. The solution to the sub-ILP is the new incumbent solution $\boldsymbol{x}^{t+1}$ and then LNS proceeds to iteration $t+1$. Compared to BnB, LNS is more effective in improving the objective value $\boldsymbol{c}^\mathsf{T}x$, especially on difficult instances [68, 69, 74]. Compared to other local search methods, LNS explores a large neighborhood in each step and thus, is more effective in avoiding local minima.

Adaptive Neighborhood Size Adaptive methods are commonly used to set the neighborhood size k^t in previous work [69, 36]. The initial neighborhood size k^0 is set to a constant or a fraction of the number of variables. In this paper, we consider the following adaptive method [36]: in iteration t, if LNS finds an improved solution, we let $k^{t+1} = k^t$, otherwise $k^{t+1} = \min\{\gamma \cdot k^t, \beta \cdot n\}$ where $\gamma > 1$ is a constant and we upper bound k^t to a constant fraction $\beta < 1$ of the number of variables to make sure the sub-ILP is not too large (thus, too difficult) to solve. Adaptively setting k^t helps LNS escape local minima by expanding the search neighborhood when it fails to improve the solution.

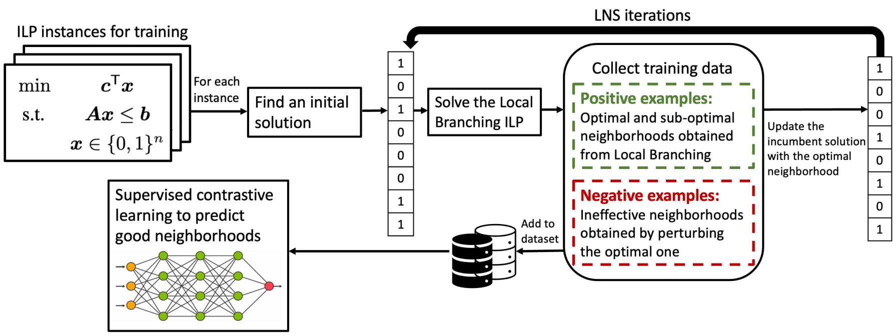


Figure 1: An overview of training and data collection for CL-LNS. For each ILP instance for training, we run several LNS iterations with LB. In each iteration, we collect both positive and negative neighborhood samples and add them to the training dataset, which is used in downstream supervised contrastive learning for neighborhood selections.

2.3 LB Heuristic

The LB Heuristic [20] is originally proposed as a primal heuristic in BnB but also applicable in LNS for ILP solving [69, 54]. Given the incumbent solution $\boldsymbol{x}^t$ in iteration t of LNS, LB 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 $\boldsymbol{x}^{t+1}$: $\sum_{i\in[n]:x_i^t=0} x_i^{t+1} + \sum_{i\in[n]:x_i^t=1}(1-x_i^{t+1}) \le k^t$. 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 too slow to be useful in practice.

3 Related Work

In this section, we summarize related work on LNS for ILPs and other COPs, learning to solve ILPs with BnB and contrastive learning for COPs. We also summarize additional related work on LNS-based primal heuristics for BnB and learning to solve other COPs in Appendix.

3.1 LNS for ILPs and Other COPs

A huge effort has been made to improve BnB for ILPs in the past decades, but LNS for ILPs has not been studied extensively. Recently, Song et al. [68] show that even a randomized destroy heuristic in LNS can outperform state-of-the-art BnB. They also show that an ML-guided decomposition-based LNS can achieve even better performance, where they apply RL and IL to learn destroy heuristics that decompose the set of variables into equally-sized subsets using a classification loss. Sonnerat et al. [69] learn to select variables by imitating LB. RL-LNS [74] uses a similar framework but trained with RL and outperforms Song et al. [68]. Both Wu et al. [74] and Sonnerat et al. [69] use the bipartite graph representations of ILPs to learn the destroy heuristics represented by GCNs. Another line of related work focuses on improving LB. Liu et al. [54] use ML to tune the runtime limit and neighborhood sizes for LB. Huang et al. [36] propose LB-RELAX to select variables by solving the LP relaxation of LB.

Besides ILPs, LNS has been applied to solve many COPs, such as VRP [63, 5], TSP [67], scheduling [46, 81] and path planning problems [51, 50, 37]. ML methods have also been applied to improve LNS for those applications [11, 55, 30, 52, 35].

3.2 Learning to Solve ILPs with BnB

Several studies have applied ML to improve BnB. The majority of works focus on learning to either select variables to branch on [40, 21, 23, 78] or select nodes to expand [25, 47]. There are also works on learning to schedule and run primal heuristics [42, 12] and to select cutting planes [70, 60, 38].

3.3 Contrastive Learning for COPs

While contrastive learning of visual representations [29, 26, 10] and graph representations [76, 72] have been studied extensively, it has not been explored much for COPs. Mulamba et al. [58] derive a contrastive loss for decision-focused learning to solve COPs with uncertain inputs that can be learned from historical data, where they view non-optimal solutions as negative samples. Duan et al. [16] use contrastive pre-training to learn good representations for the boolean satisfiability problem.

4 Contrastive Learning for LNS

Our goal is to learn a policy, a destroy heuristic represented by an ML model, that selects a subset of variables to destroy and reoptimize in each LNS iteration. Specifically, let $\boldsymbol{s}^t = (\mathcal{I}, \boldsymbol{x}^t)$ be the current state in iteration t of LNS where $\mathcal{I} = (\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ is the ILP and $\boldsymbol{x}^t$ is the incumbent solution, the policy predicts an action $\boldsymbol{a}^t = (a_1^t, \ldots, a_n^t) \in \{0,1\}^n$, a binary representation of the selected variables $\mathcal{X}^t$ indicating whether x_i is selected ($a_i^t = 1$) or not ($a_i^t = 0$). We use contrastive learning to learn to predict high quality $\boldsymbol{a}^t$ such that, after solving the sub-ILP derived from $\boldsymbol{a}^t$ (or $\mathcal{X}^t$), the resulting incumbent solution $\boldsymbol{x}^{t+1}$ is improved as much as possible. We use contrastive learning instead of other approaches since it is shown to be effective theoretically [71] and has outperformed other learning techniques empirically in other domains [18]. Next, we describe our novel data collection process, the policy network and the contrastive loss used in training. An overview of our training and data collection pipeline is shown in Figure 1. Finally, we introduce how the learned policy is used in CL-LNS.

4.1 Data Collection

Following previous work by Sonnerat et al. [69], we use LB as the expert policy to collect good demonstrations to learn to imitate. Formally, for a given state $\boldsymbol{s}^t = (\mathcal{I}, \boldsymbol{x}^t)$, we use LB to find the optimal action $\boldsymbol{a}^t$ that leads to the minimum $\boldsymbol{c}^\mathsf{T}\boldsymbol{x}^{t+1}$ after solving the sub-ILP. Different from the previous work, we use contrastive learning to learn to make discriminative predictions of $\boldsymbol{a}^t$ by contrasting positive and negative samples (i.e., good and bad examples of actions $\boldsymbol{a}^t$). In the following, we describe how we collect the positive sample set $\mathcal{S}_\mathsf{p}^t$ and the negative sample set $\mathcal{S}_\mathsf{n}^t$.

Collecting Positive Samples $\mathcal{S}_\mathsf{p}^t$ During data collection, given $\boldsymbol{s}^t = (\mathcal{I}, \boldsymbol{x}^t)$, we solve the LB ILP with the incumbent solution $\boldsymbol{x}^t$ and neighborhood size k^t to find the optimal $\boldsymbol{x}^{t+1}$. LNS proceeds to iteration $t+1$ with $\boldsymbol{x}^{t+1}$ until no improving solution $\boldsymbol{x}^{t+1}$ could be found by the LB ILP within a runtime limit. In experiments, the LB ILP is solved with SCIP 8.0.1 [8] with an hour runtime limit and k^t is fine-tuned for each type of instances. After each solve of the LB ILP, in addition to the best solution found, SCIP records all intermediate solutions found during the solve. We look for intermediate solutions $\boldsymbol{x}'$ whose resulting improvements on the objective value is at least $0 < \alpha_\mathsf{p} \leq 1$ times the best improvement (i.e., $\boldsymbol{c}^\mathsf{T}(\boldsymbol{x}^t - \boldsymbol{x}') \geq \alpha_\mathsf{p} \cdot \boldsymbol{c}^\mathsf{T}(\boldsymbol{x}^t - \boldsymbol{x}^{t+1})$) and consider their corresponding actions as positive samples. We limit the number of the positive samples $|\mathcal{S}_\mathsf{p}^t|$ to u_p. If more than u_p positive samples are available, we record the top u_p ones to avoid large computational overhead with too many samples when computing the contrastive loss (see Section 4.3). α_p and u_p are set to 0.5 and 10, respectively, in experiments.

Collecting Negative Samples $\mathcal{S}_\mathsf{n}^t$ Negative samples are critical parts of contrastive learning to help distinguish between good and bad demonstrations. We collect a set of c_n^t negative samples $\mathcal{S}_\mathsf{n}^t$, where $c_\mathsf{n}^t = \kappa|\mathcal{S}_\mathsf{p}^t|$ and κ is a hyperparameter to control the ratio between the numbers of positive and negative samples. Suppose $\mathcal{X}^t$ is the optimal set of variables selected by LB. We then perturb $\mathcal{X}^t$ to get $\hat{\mathcal{X}}^t$ by replacing 5% of the variables in $\mathcal{X}^t$ with the same number of those not in $\mathcal{X}^t$ uniformly at random. We then solve the corresponding sub-ILP derived from $\hat{\mathcal{X}}^t$ to get a new incumbent solution $\hat{\boldsymbol{x}}^{t+1}$. If the resulting improvement of $\hat{\boldsymbol{x}}^{t+1}$ is less than $0 \leq \alpha_\mathsf{n} < 1$ times the best improvement (i.e., $\boldsymbol{c}^\mathsf{T}(\boldsymbol{x}^t - \hat{\boldsymbol{x}}^{t+1}) \leq \alpha_\mathsf{n} \cdot \boldsymbol{c}^\mathsf{T}(\boldsymbol{x}^t - \boldsymbol{x}^{t+1})$), we consider its corresponding action as a negative sample. We repeat this c_n^t times to collect negative samples. If less than c_n^t negative samples is collected, we increase the perturbation rate from 5% to 10% and generate another c_n^t samples. We keep increasing the perturbation rate at an increment of 5% until c_n^t negative samples are found or it reaches 100%. In experiments, we set $\kappa = 9$ and $\alpha_\mathsf{n} = 0.05$.

4.2 Policy Network

Following previous work on learning for ILPs [21, 69, 74], we use a bipartite graph representation of ILP to encode a state $\boldsymbol{s}^t$. The bipartite graph consists of $n + m$ nodes representing the n variables and m constraints on two sides, respectively, with an edge connecting a variable and a constraint if the variable has a non-zero coefficient in the constraint. Following Sonnerat et al. [69], we use features proposed in Gasse et al. [21] for node features and edge features in the bipartite graph and also include a fixed-size window of most recent incumbent values as variable node features with the window size set to 3 in experiments. In addition to features used in Sonnerat et al. [69], we include features proposed in Khalil et al. [40] computed at the root node of BnB to make it a richer set of variable node features.

We learn a policy $\pi_{\boldsymbol{\theta}}(\cdot)$ represented by a graph attention network (GAT) [9] parameterized by learnable weights $\boldsymbol{\theta}$. The policy takes as input the state $\boldsymbol{s}^t$ and outputs a score vector $\pi_{\boldsymbol{\theta}}(\boldsymbol{s}^t) \in [0, 1]^n$, one score per variable. To increase the modeling capacity and to manipulate node interactions proposed by our architecture, we use embedding layers to map each node feature and edge feature to space $\mathbb{R}^d$. Let $\mathbf{v}_j, \mathbf{c}_i, \mathbf{e}_{i,j} \in \mathbb{R}^d$ be the embeddings of the j-th variable, i-th constraint and the edge connecting them output by the embedding layers. Since our graph is bipartite, following previous work [21], we perform two rounds of message passing through the GAT. In the first round, each constraint node $\mathbf{c}_i$ attends to its neighbors $\mathcal{N}_i$ using an attention structure with H attention heads to get updated constraint embeddings $\mathbf{c}'_i$ (computed as a function of $\mathbf{v}_j, \mathbf{c}_i, \mathbf{e}_{i,j}$). In the second round, similarly, each variable node attends to its neighbors to get updated variable embeddings $\mathbf{v}'$ (computed as a function of $\mathbf{v}_j, \mathbf{c}'_i, \mathbf{e}_{i,j}$) with another set of attention weights. After the two rounds of message passing, the final representations of variables $\mathbf{v}'$ are passed through a multi-layer perceptron (MLP) to obtain a scalar value for each variable and, finally, we apply the sigmoid function to get a score between 0 and 1. Full details of the network architecture are provided in Appendix. In experiments, d and H are set to 64 and 8, respectively.

4.3 Training with a Contrastive Loss

Given a set of ILP instances for training, we follow the expert's trajectory to collect training data. Let $\mathcal{D} = \{(\boldsymbol{s}, \mathcal{S}_{\mathsf{p}}, \mathcal{S}_{\mathsf{n}})\}$ be the set of states with their corresponding sets of positive and negative samples in the training data. A contrastive loss is a function whose value is low when the predicted action $\pi_{\boldsymbol{\theta}}(\boldsymbol{s})$ is similar to the positive samples $\mathcal{S}_{\mathsf{p}}$ and dissimilar to the negative samples $\mathcal{S}_{\mathsf{n}}$. With similarity measured by dot products, a form of supervised contrastive loss, called InfoNCE [59, 26], is used in this paper:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{(\boldsymbol{s}, \mathcal{S}_{\mathsf{p}}, \mathcal{S}_{\mathsf{n}}) \in \mathcal{D}} \frac{-1}{|\mathcal{S}_{\mathsf{p}}|} \sum_{\boldsymbol{a} \in \mathcal{S}_{\mathsf{p}}} \log \frac{\exp(\boldsymbol{a}^{\mathsf{T}} \pi_{\boldsymbol{\theta}}(\boldsymbol{s})/\tau)}{\sum_{\boldsymbol{a}' \in \mathcal{S}_{\mathsf{n}} \cup \{\boldsymbol{a}\}} \exp(\boldsymbol{a}'^{\mathsf{T}} \pi_{\boldsymbol{\theta}}(\boldsymbol{s})/\tau)}$$

where τ is a temperature hyperparameter set to 0.07 [26] in experiments.

4.4 Applying Learned Policy $\pi_{\boldsymbol{\theta}}$

During testing, we apply the learned policy $\pi_{\boldsymbol{\theta}}$ in LNS. In iteration t, let $(v_1, \cdots, v_n) := \pi_{\boldsymbol{\theta}}(\boldsymbol{s}^t)$ be the variable scores output by the policy. To select k^t variables, CL-LNS greedily selects those with the highest scores. Previous works [69, 74] use sampling methods to select the variables, but those sampling methods are empirically worse than our greedy method in CL-LNS. However, when the adaptive neighborhood size k^t reaches its upper bound $\beta \cdot n$, CL-LNS may repeat the same prediction due to the deterministic selection process. When this happens, we switch to the sampling method introduced in [69]. The sampling method selects variables sequentially: at each step, a variable x_i that has not been selected yet is selected with probability proportional to v_i^{η}, where η is a temperature parameter set to 0.5 in experiments.

5 Empirical Evaluation

5.1 Setup

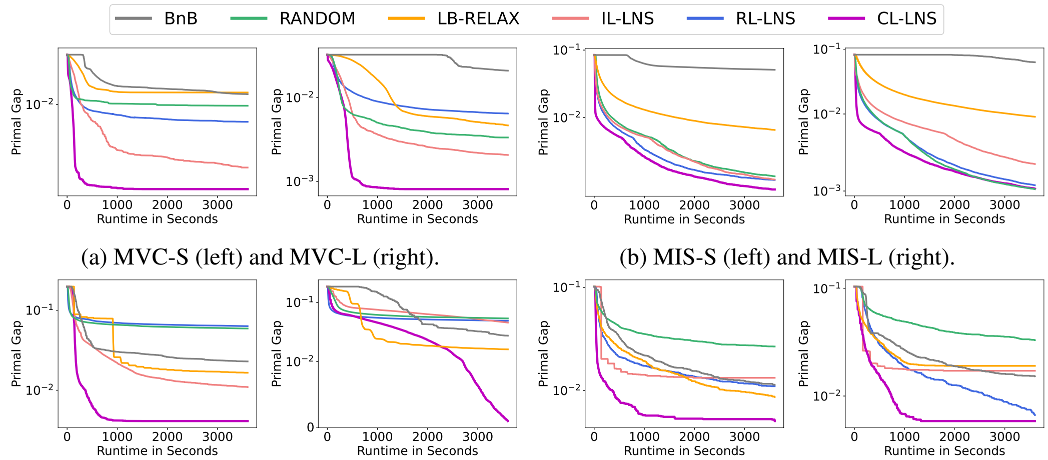
Instance Generation We evaluate on four NP-hard problem benchmarks that are widely used in existing studies [74, 68, 65], which consist of two graph optimization problems, namely the minimum vertex cover (MVC) and maximum independent set (MIS) problems, and two non-graph optimization problems, namely the combinatorial auction (CA) and set covering (SC) problems. We first generate a test set of 100 *small instances* for each problem, namely MVC-S, MIS-S, CA-S and SC-S. MVC-S instances are generated according to the Barabasi-Albert random graph model [2], with 1,000 nodes and an average degree of 70 following [68]. MIS-S instances are generated according to the Erdos-Renyi random graph model [17], with 6,000 nodes and an average degree of 5 following [68]. CA-S instances are generated with 2,000 items and 4,000 bids according to the arbitrary relations in Leyton-Brown et al. [49]. SC-S instances are generated with 4,000 variables and 5,000 constraints following Wu et al. [74]. We then generate another test set of 100 *large instances* for each problem by doubling the number of variables, namely MVC-L, MIS-L, CA-L and SC-L. More details of instance generation are included in Appendix. For data collection and training, we generate another set of 1,024 small instances for each problem. We split them into training and validation sets, each consisting of 896 and 128 instances, respectively.

Baselines We compare CL-LNS with five baselines: (1) BnB: using SCIP (v8.0.1), the state-of-the-art open-source ILP solver, with the aggressive mode fine-tuned to focus on improving the objective value; (2) RANDOM: LNS which selects the neighborhood by uniformly sampling k^t variables without replacement; (3) LB-RELAX [36]: LNS which selects the neighborhood with the LB-RELAX heuristics; (4) IL-LNS [69]; (5) RL-LNS [74]. We compare with two more baselines in Appendix. For each ML approach, a separate model is trained for each problem on the small training set and tested on both small and large test sets. We implement IL-LNS and fine-tune its hyperparameters for each problem since the authors do not fully open source the code. For RL-LNS, we use the code and hyperparameters provided by the authors and train the models with five random seeds to select one with the best performance on the validation sets. We do not compare to the approach by Song et al. [68] since it performs worse than RL-LNS on multiple problems [74].

Metrics We use the following metrics to evaluate all 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; (3) The *primal integral* [1] at time q is the integral on $[0, q]$ of the primal gap as a function of runtime. 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 primal gaps below the threshold [69]; Since BnB and LNS are both anytime algorithms, we show these metrics as a function of runtime or the number of iterations in LNS (when applicable) to demonstrate their anytime performance.

Hyperparameters We conduct experiments on 2.5GHz Intel Xeon Platinum 8259CL CPUs with 32 GB memory. Training is done on a NVIDIA A100 GPU with 40 GB memory. All experiments use the hyperparameters described below unless stated otherwise. We use SCIP (v8.0.1) [8] to solve the sub-ILP in every iteration of LNS. To run LNS, we find an initial solution by running SCIP for 10 seconds. We set the time limit to 60 minutes to solve each instance and 2 minutes for solving the sub-ILP in every LNS iteration. All approaches require a neighborhood size k^t in LNS, except for BnB and RL-LNS (k^t in RL-LNS is defined implicitly by how the policy is used). For LB-RELAX, IL-LNS and CL-LNS, the initial neighborhood size k^0 is set to $100, 3000, 1000$ and 150 for MVC, MIS, CA and SC, respectively, except k^0 is set to 150 for SC for IL-LNS; for RANDOM, it is set to $200, 3000, 1500$ and 200 for MVC, MIS, CA and SC, respectively. All approaches use adaptive neighborhood sizes with $\gamma = 1.02$ and $\beta = 0.5$, except for BnB and RL-LNS. For IL-LNS, when applying its learned policies, we use the sampling methods on MVC and CA instances and the greedy method on SC and MIS instances. For CL-LNS, the greedy method is used on all instances. Additional details on hyperparameter tunings are provided in Appendix.

For data collection, we use different neighborhood sizes $k^0 = 50, 500, 200$ and 50 for MVC, MIS, CA and SC, respectively, which we justify in Section 5.2. We set $\gamma = 1$ and run LNS with LB until no new incumbent solution is found. The runtime limit for solving LB in every iteration is set to 1 hour. For training, we use the Adam optimizer [44] with learning rate 10^{-3}. We use a batch size of 32 and train for 30 epochs.



(a) MVC-S (left) and MVC-L (right).

(b) MIS-S (left) and MIS-L (right).

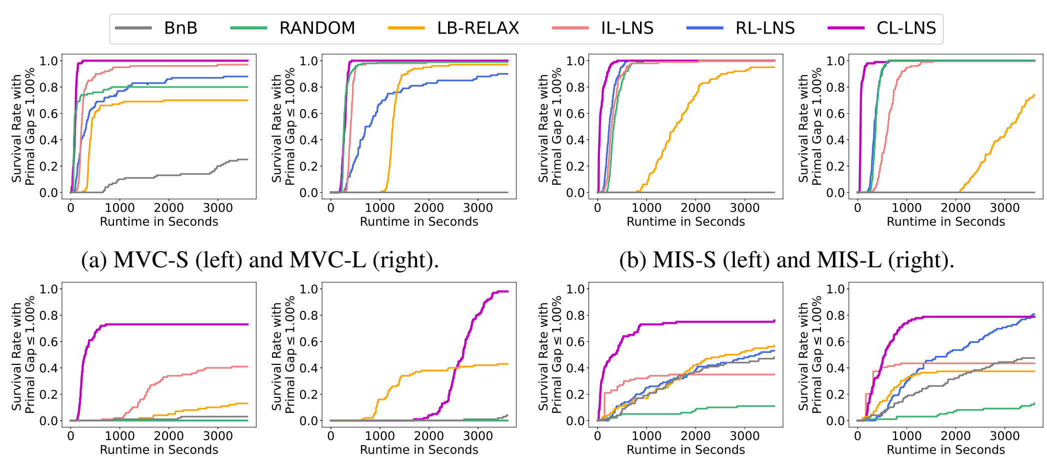
(c) CA-S (left) and CA-L (right).

(d) SC-S (left) and SC-L (right).

Figure 2: The primal gap (the lower the better) as a function of runtime, averaged over 100 test instances. For ML approaches, the policies are trained on only small training instances but tested on both small and large test instances.

Table 1: Primal gap (PG) (in percent), primal integral (PI) at 60 minutes runtime cutoff, averaged over 100 test instances and their standard deviations. "↓" means the lower the better. For ML approaches, the policies are trained on only small training instances but tested on both small and large test instances.

	PG (%) ↓	PI ↓	PG (%) ↓	PI ↓	PG (%) ↓	PI ↓	PG (%) ↓	PI ↓
	MVC-S		MIS-S		CA-S		SC-S	
BnB	1.32±0.43	66.1±13.1	5.10±0.69	222.8±25.9	2.28±0.59	137.4±25.9	1.13±0.95	86.7±37.9
RANDOM	0.96±1.26	38.0±44.8	0.24±0.14	22.1±5.0	5.90±1.02	235.6±34.9	2.67±1.29	124.3±45.4
LB-RELAX	1.38±1.51	57.0±51.2	0.65±0.20	46.9±6.5	1.65±0.57	140.5±18.3	0.86±0.83	63.2±31.6
IL-LNS	0.29±0.23	19.2±10.2	0.22±0.17	19.4±5.8	1.09±0.51	90.0±20.8	1.33±0.97	63.2±34.3
RL-LNS	0.61±0.34	29.6±11.5	0.22±0.14	17.2±5.2	6.32±1.03	249.2±35.9	1.10±0.77	77.8±28.9
CL-LNS	**0.17±0.09**	**8.7±6.7**	**0.15±0.15**	**12.8±5.4**	**0.65±0.32**	**50.7±22.7**	**0.50±0.58**	**26.2±12.8**
	MVC-L		MIS-L		CA-L		SC-L	
BnB	2.41±0.40	130.2±11.1	6.29±1.62	285.1±18.2	2.74±1.87	320.9±83.1	1.54±1.33	115.0±42.5
RANDOM	0.38±0.24	22.7±8.0	**0.11±0.08**	19.0±3.1	5.37±0.75	229.2±24.4	3.31±1.79	166.4±61.3
LB-RELAX	0.46±0.23	48.4±7.5	0.91±0.16	68.6±5.5	1.61±1.50	153.0±50.3	1.91±1.42	88.3±48.9
IL-LNS	0.27±0.23	21.2±8.1	0.29±0.15	27.1±5.5	4.56±0.98	254.2±33.4	1.72±1.19	79.1±42.4
RL-LNS	0.59±0.30	37.3±9.6	0.14±0.12	18.9±4.1	4.91±0.81	197.0±28.5	0.66±0.72	116.2±27.1
CL-LNS	**0.05±0.04**	**9.1±3.4**	0.12±0.11	**12.9±4.4**	**0.09±0.10**	**116.1±18.0**	**0.58±0.45**	**39.2±23.2**



(a) MVC-S (left) and MVC-L (right).

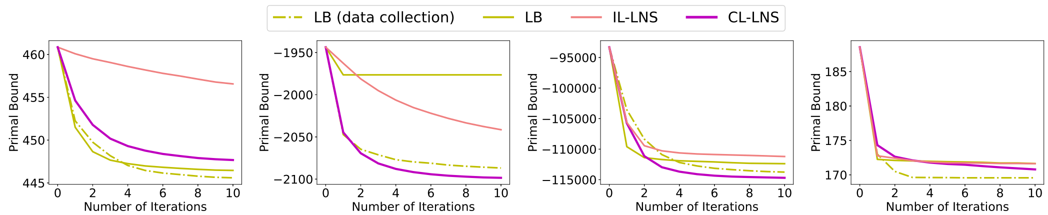
(b) MIS-S (left) and MIS-L (right).

(c) CA-S (left) and CA-L (right).

(d) SC-S (left) and SC-L (right).

Figure 3: The survival rate (the higher the better) over 100 test instances as a function of runtime to meet primal gap threshold 1.00%. For ML approaches, the policies are trained on only small training instances but tested on both small and large test instances.

	MVC-S		MIS-S		CA-S		SC-S	
	NH size	Runtime	NH size	Runtime	NH size	Runtime	NH size	Runtime
LB	100	3600±0	3,000	3600±0	1,000	3600±0	100	3600±0
LB (data collection)	50	3600±0	500	3600±0	200	3600±0	50	3600±0
IL-LNS	100	2.1±0.1	3,000	1.3±0.2	1,000	20.8±13.1	150	120.9±1.3
CL-LNS	100	2.2±0.1	3,000	1.3±0.1	1,000	25.1±15.3	100	50.1±10.4



(a) MVC-S (b) MIS-S (c) CA-S (d) SC-S

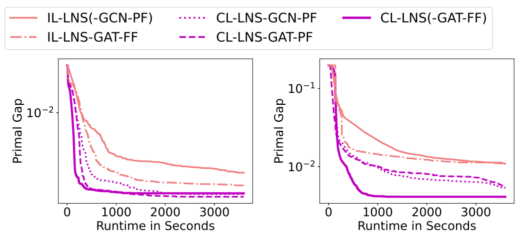
Figure 4: The primal bound as a function of number of iterations, averaged over 100 small test instances. LB and LB (data collection) are LNS with LB using the neighborhood sizes fune-tunded for CL-LNS and for data collection, respectively. The table shows the neighborhood size (NH size) and the average runtime in seconds (with standard deviations) per iteration for each approach.

5.2 Results

Figure 2 shows the primal gap as a function of runtime. Table 1 presents the average primal gap and primal integral at 60 minutes runtime cutoff on small and large instances, respectively (see results at 15, 30 and 45 minutes runtime cutoff in Appendix). Note that we were not able to reproduce the results on CA-S and CA-L reported in Wu et al. [74] for RL-LNS despite using their code and repeating training with five random seeds. CL-LNS shows significantly better anytime performance than all baselines on all problems, achieving the smallest average primal gap and primal integral. It also demonstrates strong generalization performance on large instances unseen during training. Figure 3 shows the survival rate to meet the 1.00% primal gap threshold. CL-LNS achieves the best survival rate at 60 minutes runtime cutoff on all instances, except that, on SC-L, its final survival rate is slightly worse than RL-LNS but it achieves the rate with a much shorter runtime. On MVC-L, MIS-S and MIS-L instances, several baselines achieve the same survival rate as CL-LNS but it always achieves the rates with the shortest runtime. In Appendix, we present more results in comparison with two more baselines.

Comparison with LB (the Expert) Both IL-LNS and CL-LNS learn to imitate LB. On the small test instances, we run LB with two different neighborhood sizes, one that is fine-tuned in data collection and the other the same as CL-LNS, for 10 iterations and compare its per iteration performance with IL-LNS and CL-LNS. This allows us to compare the quality of the learned policies to the expert independently of their speed. The runtime limit per iteration for LB is set to 1 hour. Figure 4 shows the primal bound as a function of the number of iterations. The table in the figure summarizes the neighborhood sizes and the average runtime per iteration. For LB, the result shows that the neighborhood size affects the overall performance. Intuitively, using a larger neighborhood size in LB allows LNS to find better incumbent solutions due to being able to explore larger neighborhoods. However, in practice, LB becomes less efficient in finding good incumbent solutions as the neighborhood size increases, sometimes even performs worse than using a smaller neighborhood size (the one for data collection). The neighborhood size for data collection is fine-tuned on validation sets to achieve the best primal bound upon convergences, allowing the ML models to observe demonstrations that lead to as good primal bounds as possible in training. However, when using the ML models in testing, we have the incentive to use a larger neighborhood size and fine-tune it since we no longer suffer from the bottleneck of LB. Therefore, we fine-tune the neighborhood sizes for IL-LNS and CL-LNS separately on validation sets. CL-LNS has a strong per-iteration performance that is consistently better than IL-LNS. With the fine-tuned neighborhood size, it even outperforms the expert that it learns from (LB for data collection) on MIS-S and CA-S.

Ablation Study We evaluate how contrastive learning and two enhancements contribute to CL-LNS's performance. Compared to IL-LNS, CL-LNS uses (1) addition features from Khalil et al. [40] and (2) GAT instead of GCN. We denote by "FF" the full feature set used in CL-LNS and "PF"



(a) MVC-S (b) CA-S

Figure 6: Ablation study: The primal gap as a function of time, averaged over 100 test instances.

Table 2: Ablation study: Primal gap (PG) (in percent) and primal integral (PI) at 60 minutes runtime cutoff, averaged over 100 small test instances and their standard deviations. "↓" means the lower the better.

	PG (%) ↓	PI ↓	PG (%) ↓	PI ↓
	MVC-S		CA-S	
IL-LNS(-GCN-PF)	0.29±0.23	19.2±10.2	1.09±0.51	90.0±20.8
IL-LNS-GAT-FF	0.24±0.17	15.3±7.3	1.13±0.63	78.9±22.7
CL-LNS-GCN-PF	0.17±0.10	11.4±8.8	0.75±0.40	57.9±21.2
CL-LNS-GAT-PF	**0.16±0.09**	10.1±0.6	0.76±0.39	53.8±22.1
CL-LNS(-GAT-FF)	0.17±0.09	**8.7±6.7**	**0.65±0.32**	**50.7±22.7**

the partial feature set in IL-LNS. We also evaluate the performance of IL-LNS with FF and GAT (denoted by IL-LNS-GAT-FF), CL-LNS with GCN and PF (denoted by CL-LNS-GCN-PF) as well as CL-LNS with GAT and PF (denoted by CL-LNS-GAT-PF) on MVC-S and CA-S. Figure 6 shows the primal gap as a function of runtime. Table 2 presents the primal gap and primal integral at 60 minutes runtime cutoff. The result shows that IL-LNS-GAT-FF, imitation learning with the two enhancements, still performs worse than CL-LNS-GCN-PF without any enhancements. CL-LNS-GCN-PF and CL-LNS-GAT-PF perform similarly in terms of the primal gaps but CL-LNS-GAT-PF has better primal integrals, showing the benefit of replacing GCN with GAT. On MVC-S, three variants of CL-LNS have similar average primal gaps and on CA-S, CL-LNS has better average primal gap than the other two variants. But adding the two enhancements helps improve the primal integral, leading to the overall best performance of CL-LNS on both MVC-S and CA-S.

6 Conclusion

We proposed CL-LNS, which uses a contrastive loss to learn efficient and effective destroy heuristics in LNS for ILPs. We presented a novel data collection process tailored for CL-LNS and used GAT with a richer set of features to further improve its performance. Empirically, CL-LNS significantly outperformed state-of-the-art approaches on four ILP benchmarks w.r.t. to the primal gap, the primal integral, the best performing rate and the survival rate. CL-LNS achieved good generalization performance on out-of-distribution instances that are two times larger than those used in training. It is future work to learn policies that can generalize across problem domains. CL-LNS does not guarantee optimality and it is also interesting future work to integrate it in BnB for which many other learning techniques are developed. Our approach is closely related to and could be useful for many problems of identifying substructures in combinatorial searches, for example, identifying backdoor variables in ILPs [19] and selecting neighborhoods in LNS for other COPs.

References

[1] T. Achterberg, T. Berthold, and G. Hendel. Rounding and propagation heuristics for mixed integer programming. In *Operations research proceedings 2011*, pages 71–76. Springer, 2012.

[2] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[3] A. R. Amaral. An exact approach to the one-dimensional facility layout problem. *Operations research*, 56(4):1026–1033, 2008.

[4] S. Amizadeh, S. Matusevych, and M. Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *International Conference on Learning Representations*, 2018.

[5] N. Azi, M. Gendreau, and J.-Y. Potvin. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, 41:167–173, 2014.

[6] T. Berthold. *Primal heuristics for mixed integer programs*. PhD thesis, Zuse Institute Berlin (ZIB), 2006.

[7] T. Berthold. Rens. *Mathematical Programming Computation*, 6(1):33–54, 2014.

[8] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, and J. Witzig. The SCIP Optimization Suite 8.0. Technical report, Optimization Online, December 2021. URL http://www.optimization-online.org/DB_HTML/2021/12/8728.html.

[9] S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? *International conference on learning representations*, 2022.

[10] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[11] X. Chen and Y. Tian. Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[12] A. Chmiela, E. Khalil, A. Gleixner, A. Lodi, and S. Pokutta. Learning to schedule heuristics in branch and bound. *Advances in Neural Information Processing Systems*, 34:24235–24246, 2021.

[13] E. Danna, E. Rothberg, and C. L. Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

[14] S. De Vries and R. V. Vohra. Combinatorial auctions: A survey. *INFORMS Journal on computing*, 15(3):284–309, 2003.

[15] B. Dilkina and C. P. Gomes. Solving connected subgraph problems in wildlife conservation. In *CPAIOR*, volume 6140, pages 102–116. Springer, 2010.

[16] H. Duan, P. Vaezipoor, M. B. Paulus, Y. Ruan, and C. Maddison. Augment with care: Contrastive learning for combinatorial problems. In *International Conference on Machine Learning*, pages 5627–5642. PMLR, 2022.

[17] P. Erdos, A. Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[18] B. Eysenbach, T. Zhang, S. Levine, and R. R. Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems*, 35: 35603–35620, 2022.

[19] A. Ferber, J. Song, B. Dilkina, and Y. Yue. Learning pseudo-backdoors for mixed integer programs. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 91–102. Springer, 2022.

[20] M. Fischetti and A. Lodi. Local branching. *Mathematical programming*, 98(1):23–47, 2003.

[21] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[22] S. Ghosh. Dins, a mip improvement heuristic. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 310–323. Springer, 2007.

[23] P. Gupta, M. Gasse, E. Khalil, P. Mudigonda, A. Lodi, and Y. Bengio. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33:18087–18097, 2020.

[24] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL https://www.gurobi.com.

[25] H. He, H. Daume III, and J. M. Eisner. Learning to search in branch and bound algorithms. *Advances in neural information processing systems*, 27, 2014.

[26] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[27] G. Hendel. Adaptive large neighborhood search for mixed integer programming. *Mathematical Programming Computation*, 14(2):185–221, 2022.

[28] S. S. Heragu and A. Kusiak. Efficient models for the facility layout problem. *European Journal of Operational Research*, 53(1):1–13, 1991.

[29] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio. Learning deep representations by mutual information estimation and maximization. *International conference on learning representations*, 2019.

[30] A. Hottung and K. Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *ECAI 2020*, pages 443–450. IOS Press, 2020.

[31] Y. Hu, Y. Yao, and W. S. Lee. A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs. *Knowledge-Based Systems*, 204:106244, 2020.

[32] T. Huang and B. Dilkina. Enhancing seismic resilience of water pipe networks. In *Proceedings of the 3rd ACM SIGCAS Conference on Computing and Sustainable Societies*, pages 44–52, 2020.

[33] T. Huang, B. Dilkina, and S. Koenig. Learning node-selection strategies in bounded suboptimal conflict-based search for multi-agent path finding. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2021.

[34] T. Huang, S. Koenig, and B. Dilkina. Learning to resolve conflicts for multi-agent path finding with conflict-based search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11246–11253, 2021.

[35] T. Huang, J. Li, S. Koenig, and B. Dilkina. Anytime multi-agent path finding via machine learning-guided large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 9368–9376, 2022.

[36] T. Huang, A. Ferber, Y. Tian, B. Dilkina, and B. 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.

[37] T. Huang, V. Shivashankar, M. Caldara, J. Durham, J. Li, B. Dilkina, and S. Koenig. Deadline-aware multi-agent tour planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2023.

[38] Z. Huang, K. Wang, F. Liu, H.-L. Zhen, W. Zhang, M. Yuan, J. Hao, Y. Yu, and J. Wang. Learning to select cuts for efficient mixed-integer programming. *Pattern Recognition*, 123: 108353, 2022.

[39] D. S. Johnson, J. K. Lenstra, and A. R. Kan. The complexity of the network design problem. *Networks*, 8(4):279–285, 1978.

[40] E. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[41] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

[42] E. B. Khalil, B. Dilkina, G. L. Nemhauser, S. Ahmed, and Y. Shao. Learning to run heuristics in tree search. In *Ijcai*, pages 659–666, 2017.

[43] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan. Supervised contrastive learning. *Advances in Neural Information Processing Systems*, 33:18661–18673, 2020.

[44] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. 2015.

[45] W. Kool, H. Van Hoof, and M. Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

[46] A. A. Kovacs, S. N. Parragh, K. F. Doerner, and R. F. Hartl. Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of scheduling*, 15(5):579–600, 2012.

[47] A. G. Labassi, D. Chételat, and A. Lodi. Learning to compare nodes in branch and bound with graph neural networks. *Advances in neural information processing systems*, 2022.

[48] A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer, 2010.

[49] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 66–76, 2000.

[50] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig. Anytime multi-agent path finding via large neighborhood search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4127–4135, 2021.

[51] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig. MAPF-LNS2: Fast repairing for multi-agent path finding via large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 10256–10265, 2022.

[52] S. Li, Z. Yan, and C. Wu. Learning to delegate for large-scale vehicle routing. *Advances in Neural Information Processing Systems*, 34:26198–26211, 2021.

[53] Z. Li, Q. Chen, and V. Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31, 2018.

[54] D. Liu, M. Fischetti, and A. Lodi. Learning to search in local branching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3796–3803, 2022.

[55] H. Lu, X. Zhang, and S. Yang. A learning-based iterative method for solving vehicle routing problems. In *International conference on learning representations*, 2020.

[56] S. J. Maher, T. Fischer, T. Gally, G. Gamrath, A. Gleixner, R. L. Gottwald, G. Hendel, T. Koch, M. Lübbecke, M. Miltenberger, et al. The scip optimization suite 4.0. 2017.

[57] A. S. Manne. On the job-shop scheduling problem. *Operations research*, 8(2):219–223, 1960.

[58] M. Mulamba, J. Mandi, M. Diligenti, M. Lombardi, V. B. Lopez, and T. Guns. Contrastive losses and solution caching for predict-and-optimize. In *30th International Joint Conference on Artificial Intelligence*, page 2833. International Joint Conferences on Artificial Intelligence, 2021.

[59] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[60] M. B. Paulus, G. Zarpellon, A. Krause, L. Charlin, and C. Maddison. Learning to cut by looking ahead: Cutting plane selection via imitation learning. In *International conference on machine learning*, pages 17584–17600. PMLR, 2022.

[61] I. Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4): 193–204, 1970.

[62] A. Prouvost, J. Dumouchelle, L. Scavuzzo, M. Gasse, D. Chételat, and A. Lodi. Ecole: A gym-like library for machine learning in combinatorial optimization solvers. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020. URL `https://openreview.net/forum?id=IVc9hqgibyB`.

[63] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

[64] E. Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.

[65] L. Scavuzzo, F. Y. Chen, D. Chételat, M. Gasse, A. Lodi, N. Yorke-Smith, and K. Aardal. Learning to branch with tree mdps. *arXiv preprint arXiv:2205.11107*, 2022.

[66] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

[67] S. L. Smith and F. Imeson. Glns: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, 87:1–19, 2017.

[68] J. Song, Y. Yue, B. Dilkina, et al. A general large neighborhood search framework for solving integer linear programs. *Advances in Neural Information Processing Systems*, 33:20012–20023, 2020.

[69] N. Sonnerat, P. Wang, I. Ktena, S. Bartunov, and V. Nair. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201*, 2021.

[70] Y. Tang, S. Agrawal, and Y. Faenza. Reinforcement learning for integer programming: Learning to cut. In *International conference on machine learning*, pages 9367–9376. PMLR, 2020.

[71] Y. Tian. Understanding deep contrastive learning via coordinate-wise optimization. In *Advances in Neural Information Processing Systems*, 2022.

[72] Z. Tong, Y. Liang, H. Ding, Y. Dai, X. Li, and C. Wang. Directed graph contrastive learning. *Advances in Neural Information Processing Systems*, 34:19580–19593, 2021.

[73] P. Toth and D. Vigo. *The vehicle routing problem.* SIAM, 2002.

[74] Y. Wu, W. Song, Z. Cao, and J. Zhang. Learning large neighborhood search policy for integer programming. *Advances in Neural Information Processing Systems*, 34:30075–30087, 2021.

[75] L. Xin, W. Song, Z. Cao, and J. Zhang. Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. *Advances in Neural Information Processing Systems*, 34:7472–7483, 2021.

[76] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33:5812–5823, 2020.

[77] C. Yu, Q. Li, S. Gao, and A. Prorok. Accelerating multi-agent planning using graph transformers with bounded suboptimality. *arXiv preprint arXiv:2301.08451*, 2023.

[78] G. Zarpellon, J. Jo, A. Lodi, and Y. Bengio. Parameterizing branch-and-bound search trees to learn branching policies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3931–3939, 2021.

[79] S. Zhang, J. Li, T. Huang, S. Koenig, and B. Dilkina. Learning a priority ordering for prioritized planning in multi-agent path finding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, pages 208–216, 2022.

[80] J. Zheng, K. He, J. Zhou, Y. Jin, and C.-M. Li. Combining reinforcement learning with lin-kernighan-helsgaun algorithm for the traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12445–12452, 2021.

[81] I. Žulj, S. Kramer, and M. Schneider. 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 Additional Related Work

A.1 LNS-Based Primal Heuristics in BnB

LNS-based primal heuristics is a family of primal heuristics in BnB and have been studied extensively in past decades. With the same purpose of improving primal bounds, the main differences between the LNS-based primal heuristics in BnB and LNS for ILPs are: (1) LNS-based primal heuristics are executed periodically at different search tree nodes during the search and the execution schedule is itself dynamic, because they are often more expensive to run than the other primal heuristics in BnB; (2) the destroy heuristics in LNS-based primal heuristics are often designed to use information specific to BnB, such as the dual bound and the LP relaxation at a search tree node, and they are not directly applicable in LNS for ILPs in our setting.

Next, we briefly summarize the destroy heuristics in LNS-based primal heuristics:

- Crossover heuristics [64]: it destroys variables that have different values in a set of selected known solutions (typically two). The Mutation heuristics [64] destroys a random subset of variables.
- Relaxation Induced Neighborhood Search (RINS) [13]: it destroys variables whose values disagree in the solution of the LP relaxation at the search tree node and the incumbent solution.
- Relaxation Enforced Neighborhood Search (RENS) [7]: it restricts the neighborhood to be the feasible roundings of the LP relaxation at the current search tree node.
- Local Branching (LB)[20]: it restricts the neighborhood to a ball around the current incumbent solution.
- Distance Induced Neighborhood Search (DINS) [22]: it takes the intersection of the neighborhoods of the Crossover, Local Branching and Relaxation Induced Neighborhood Search heuristics.
- Graph-Induced Neighborhood Search (GINS) [56]: it destroys the breadth-first-search neighborhood of a variable in the bipartite graph representation of the ILP.

Recently, an adaptive LNS primal heuristic [27] has been proposed to combine the power of these heuristics, where it essentially solves a multi-armed bandit problem to choose which heuristic to apply.

A.2 Learning to Solve Other COPs

ML has been applied to solve a number of COPs, including TSP [31, 75, 80], vehicle routing [45, 55], boolean satisfiability [66, 4], general graph optimization problems [41, 53] and multi-agent path finding [33, 34, 79, 77].

B Network Architecture

We give full details of the GAT architecture described in Section 4.2. The policy takes as input the state $\boldsymbol{s}^t$ and output a score vector $\pi_{\boldsymbol{\theta}}(\boldsymbol{s}^t) \in [0,1]^n$, one score per variable. We use 2-layer MLPs with 64 hidden units per layer and ReLU as the activation function to map each node feature and edge feature to $\mathbb{R}^d$ where $d = 64$.

Let $\mathbf{v}_j, \mathbf{c}_i, \mathbf{e}_{i,j} \in \mathbb{R}^d$ be the embeddings of the j-th variable, i-th constraint and the edge connecting them output by the embedding layers. We perform two rounds of message passing through the GAT. In the first round, each constraint node $\mathbf{c}_i$ attends to its neighbors $\mathcal{N}_i$ using an attention stucture with $H = 8$ attention heads:

$$\mathbf{c}'_i = \frac{1}{H}\sum_{h=1}^{H}\left(\alpha_{ii,1}^{(h)}\boldsymbol{\theta}_{c,1}^{(h)}\mathbf{c}_i + \sum_{j\in\mathcal{N}_i}\alpha_{ij,1}^{(h)}\boldsymbol{\theta}_{v,1}^{(h)}\mathbf{v}_j\right)$$

where $\boldsymbol{\theta}_{c,1}^{(h)} \in \mathbb{R}^{d\times d}$ and $\boldsymbol{\theta}_{v,1}^{(h)} \in \mathbb{R}^{d\times d}$ are learnable weights. The updated constraint embeddings $\mathbf{c}'_i$ are averaged across H attention heads using attention weights [9]

$$\alpha_{ij,1}^{(h)} = \frac{\exp(\boldsymbol{w}_1^\intercal \rho([\boldsymbol{\theta}_{c,1}^{(h)}\mathbf{c}_i, \boldsymbol{\theta}_{v,1}^{(h)}\mathbf{v}_j, \boldsymbol{\theta}_{e,1}^{(h)}\mathbf{e}_{i,j}]))}{\sum_{k\in\mathcal{N}_i} \exp(\boldsymbol{w}_1^\intercal \rho([\boldsymbol{\theta}_{c,1}^{(h)}\mathbf{c}_i, \boldsymbol{\theta}_{v,1}^{(h)}\mathbf{v}_k, \boldsymbol{\theta}_{e,1}^{(h)}\mathbf{e}_{i,k}]))}$$

where the attention coefficients $\boldsymbol{w}_1 \in \mathbb{R}^{3d}$ and $\boldsymbol{\theta}_{e,1}^{(h)} \in \mathbb{R}^{d\times d}$ are both learnable weights and $\rho(\cdot)$ refers to the LeakyReLU activation function with negative slope 0.2. In the second round, similary, each variable node attends to its neighbors to get updated variable node embeddings

$$\mathbf{v}'_j = \frac{1}{H}\sum_{h=1}^{H}\left(\alpha_{jj,2}^{(h)}\boldsymbol{\theta}_{v,2}^{(h)}\mathbf{v}_j + \sum_{i\in\mathcal{N}_j}\alpha_{ji,2}^{(h)}\boldsymbol{\theta}_{c,2}^{(h)}\mathbf{c}'_i\right)$$

with attention weights

$$\alpha_{ji,2}^{(h)} = \frac{\exp(\boldsymbol{w}_2^\intercal \rho([\boldsymbol{\theta}_{c,2}^{(h)}\mathbf{c}'_i, \boldsymbol{\theta}_{v,2}^{(h)}\mathbf{v}_j, \boldsymbol{\theta}_{e,2}^{(h)}\mathbf{e}_{i,j}]))}{\sum_{k\in\mathcal{N}_j} \exp(\boldsymbol{w}_2^\intercal \rho([\boldsymbol{\theta}_{c,2}^{(h)}\mathbf{c}'_i, \boldsymbol{\theta}_{v,2}^{(h)}\mathbf{v}_j, \boldsymbol{\theta}_{e,2}^{(h)}\mathbf{e}_{i,k}]))}$$

where $\boldsymbol{w}_2 \in \mathbb{R}^{3d}$ and $\boldsymbol{\theta}_{c,2}^{(h)}, \boldsymbol{\theta}_{v,2}^{(h)}, \boldsymbol{\theta}_{e,2}^{(h)} \in \mathbb{R}^{d\times d}$ are learnable weights. After the two rounds of message passing, the final representations of variables $\mathbf{v}'$ are passed through a 2-layer MLP with 64 hidden units per layer to obtain a scalar value for each variable. Finally, we apply the sigmoid function to get a score between 0 and 1.

B.1 Features

We use features proposed in Gasse et al. [21] for node features and edge features in the bipartite graph and also include a fixed-size window of most recent incumbent values as variable node features with the window size set to 3 in experiments. In addition, we include features proposed in Khalil et al. [40] computed at the root node of BnB to make it a richer set of variable node features. The full list of features can be found in Table 2 in Appendix of Gasse et al. [21] and Table 1 in Khalil et al. [40]. In our implementation, we compute them using the APIs provided by the Ecole library [62][1].

C Additional Details of Instance Generation

We present the ILP formulations for the minimum vertex cover (MVC), maximum independent set (MIS), set covering (SC) and combinatorial auction (CA) problems. For each test set, Table 3 shows its average numbers of variables and constraints.

Table 3: Names and the average numbers of variables and constraints of the test instances.

	Small Instances				Large Instances			
Name	MVC-S	MIS-S	CA-S	SC-S	MVC-L	MIS-L	CA-L	SC-L
#Variables	1,000	6,000	4,000	4,000	2,000	12,000	8,000	8,000
#Constraints	65,100	23,977	2,675	5,000	135,100	48,027	5,353	5,000

C.1 MVC

In an MVC instance, we are given an undirected graph $G = (V, E)$. The goal is to select the smallest subset of nodes such that at least one end point of every edge in the graph is selected:

$$\begin{aligned} &\min \textstyle\sum_{v\in V} x_v \\ \text{s.t.}\quad & x_u + x_v \geq 1,\ \forall (u,v) \in E, \\ & x_v \in \{0,1\},\ \forall v \in V. \end{aligned}$$

[1]More details and the source code can be found at `https://doc.ecole.ai/py/en/stable/reference/observations.html`.

C.2 MIS

In an MIS instance, we are given an undirected graph $G = (V, E)$. The goal is to select the largest subset of nodes such that no two nodes in the subsets are connected by an edge in G:

$$\begin{aligned} &\min -\textstyle\sum_{v\in V} x_v \\ \text{s.t.} \quad & x_u + x_v \leq 1,\ \forall (u, v) \in E, \\ & x_v \in \{0, 1\},\ \forall v \in V. \end{aligned}$$

C.3 SC

In an SC instance, we are given m elements and a collection S of n sets whose union is the set of all elements. The goal is to select a minimum number of sets from S such that the union of the selected set is still the set of all elements:

$$\begin{aligned} &\min \textstyle\sum_{s\in S} x_s \\ \text{s.t.} \quad & \textstyle\sum_{s\in S: i\in s} x_s \geq 1,\ \forall i \in [m], \\ & x_s \in \{0, 1\},\ \forall s \in S. \end{aligned}$$

C.4 CA

In a CA instance, we are given n bids $\{(B_i, p_i) : i \in [n]\}$ for m items, where B_i is a subset of items and p_i is its associated bidding price. The objective is to allocate items to bids such that the total revenue is maximized:

$$\begin{aligned} &\min -\textstyle\sum_{i\in[n]} p_i x_i \\ \text{s.t.} \quad & \textstyle\sum_{i: j\in B_i} x_i \leq 1,\ \forall j \in [m], \\ & x_i \in \{0, 1\},\ \forall i \in [n]. \end{aligned}$$

D Additional Details on Hyperparameter Tuning

For RL-LNS, we use all the hyperparameters provided in their code [74] in our experiments. For the other LNS methods, all hyperparameters used in experiments are fine-tuned on the validation set and the hyperparameter tunings are described in the following.

For β, which upper bounds the neighborhood size, we tried values from $\{0.25, 0.5, 0.6, 0.7\}$. $\beta = 0.25$ is the worst for all approaches, resulting in the highest gap. For LB-RELAX, IL-LNS and CL-LNS, all values perform similarly (because they select effective neighborhoods early in the search and their neighborhood sizes either do not reach the upper bound 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, we observe that the best values are sensitive for approaches that need longer runtime to select variables, such as LB-RELAX, IL-LNS and CL-LNS, thus they need the right k^0 from the beginning and we fine-tune it for them. For RANDOM and GRAPH, their runtime for selecting 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. Despite the differences being small, we still use the best k^0 for them.

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

For the runtime limit for each repair operation, we tried different limits of 0.5, 1, 2 and 5 minutes. All approaches are not sensitive to it since most repairs are finished within 20 seconds. Except for IL-LNS on the SC instances, it selects neighborhoods that require a longer time to repair and a 2-minute runtime limit is necessary. Therefore, we use 2 minutes consistently.

Table 4: Hyperparameters with their notations and values used.

Hyperparameter	Notation	Value
Suboptimality threshold to determine positive samples	α_p	0.5
Upper bound on the number of positive samples	u_p	10
Suboptimality threshold to determine negative samples	α_n	0.05
Ratio between the numbers of positive and negative samples	κ	9
Feature embedding dimension	d	64
Window size of the most recent incumbent values in variable features		3
Number of attention heads in the GAT	H	8
Temperature parameter in the contrastive loss	τ	0.07
Rate at which k^t increases	γ	1.02
Upper bound on k^t as a fraction of number of variables	β	0.5
Temperature parameter for sampling variables in IL-LNS	η	0.5
Initial neighborhood size	k^0	Fine-tuned for each case
Runtime for finding initial solution		10 seconds
Runtime limit for each reoptimization		2 minutes
Learning rate (CL-LNS and IL-LNS)		10^{-3}
Batch size (CL-LNS and IL-LNS)		32
Number of training epochs (CL-LNS and IL-LNS)		30

For BnB, the aggressive mode is fine-tuned for each problem on the validation set. With the aggressive mode turned on, BnB (SCIP) does not always deliver better anytime performance than having it turned off. Based on the validation results, the aggressive mode is turned on for MVC and SC instances and turned off for CAT and MIS instances.

For IL-LNS, it uses the same training dataset as CL-LNS but uses only the positive samples. We fine-tune its hyperparameters for each problem on the validation set, resulting in a different k^0 on the SC instance from CL-LNS. Also in Sonnerat et al. [69], they use sampling methods to select variables when using the learned policy. For the temperature parameter η in the sampling method, we tried values from $\{1/2, 2/3, 1\}$ and $\eta = 0.5$ performs the best overall. However, in our experiment, we observe that our greedy method described in Section 4.4 works better for IL-LNS on SC and MIS instances, thus, CL-LNS is compared against the corresponding results on SC and MIS instances.

For LB-RELAX, there are three variants of it presented in Huang et al. [36]. We present only the best of the three variants for each problem in the paper for simplicity.

In Table 4, we summarize all the hyperparameters with their notations and values used in our experiments.

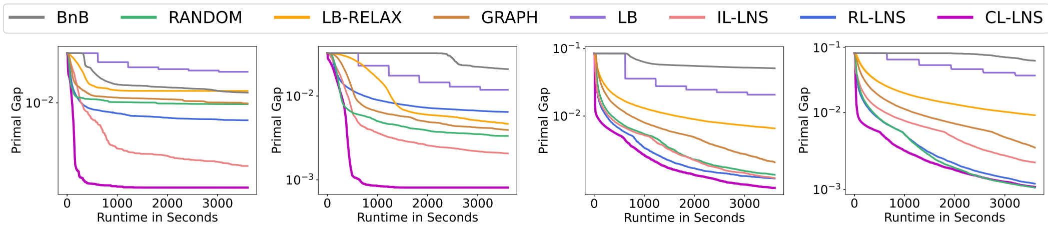
E Additional Experimental Results

In this section, we add two more baselines and evaluate all approaches on one more metric. We show that CL-LNS outperforms all approaches in terms of all metrics.

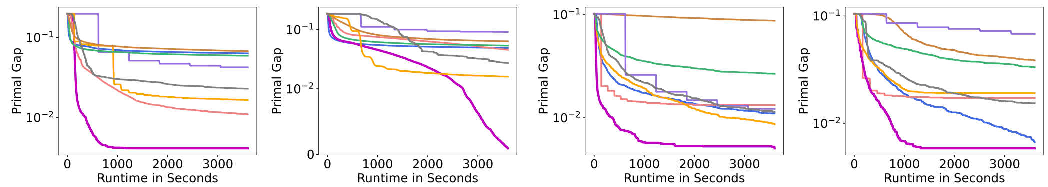
We establish two additional baselines:

- LB: LNS which selects the neighborhood with the LB heuristics. We set the time limit to 10 minutes for solving the LB ILP in each iteration;
- GRAPH: LNS which selects the neighborhood based on the bipartite graph representation of the ILP similar to GINS [56]. 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.

Figure 7 shows the full results on the primal gap as a function of runtime. Figure 8 shows the full results on the survival rate as a function of runtime. Figure 9 shows the full results on the primal bound as a function of runtime. Tables 5, 6, 7 and 8 present the average primal bound, primal gap and primal integral at 15, 30, 45 and 60 minutes runtime cutoff, respectively, on the small instances. Tables 9, 10, 11 and 12 present the average primal bound, primal gap and primal integral at 15, 30, 45 and 60 minutes runtime cutoff, respectively, on the large instances.

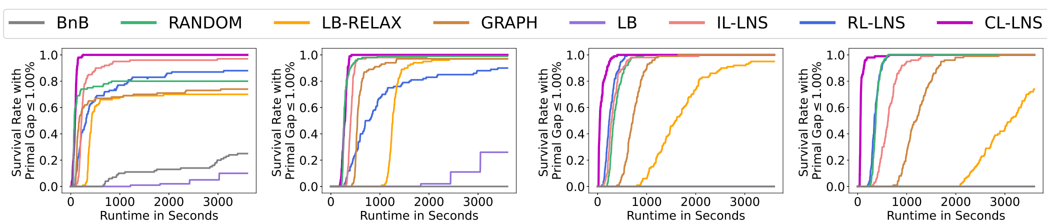


(a) MVC-S (left) and MVC-L (right). (b) MIS-S (left) and MIS-L (right).

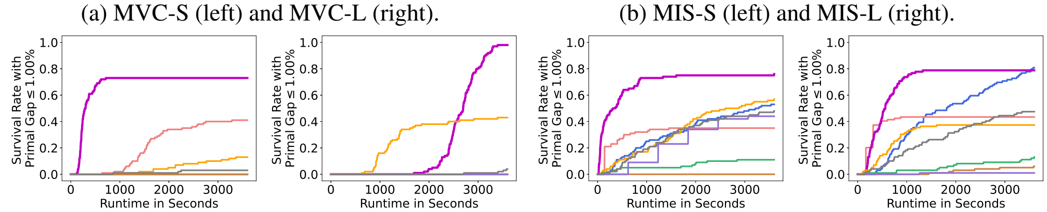


(c) CA-S (left) and CA-L (right). (d) SC-S (left) and SC-L (right).

Figure 7: The primal gap (the lower the better) as a function of time, averaged over 100 instances. For ML approaches, the policies are trained on only small training instances but tested on both small and large test instances.



(a) MVC-S (left) and MVC-L (right). (b) MIS-S (left) and MIS-L (right).

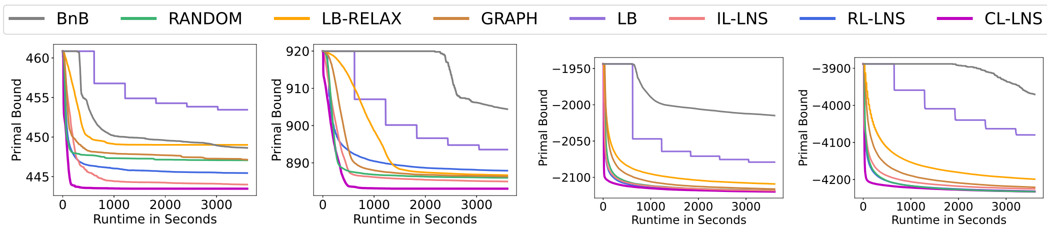


(c) CA-S (left) and CA-L (right). (d) SC-S (left) and SC-L (right).

Figure 8: The survival rate (the higher the better) over 100 instances as a function of time to meet primal gap threshold 1.00%. For ML approaches, the policies are trained on only small training instances but tested on both small and large test instances.

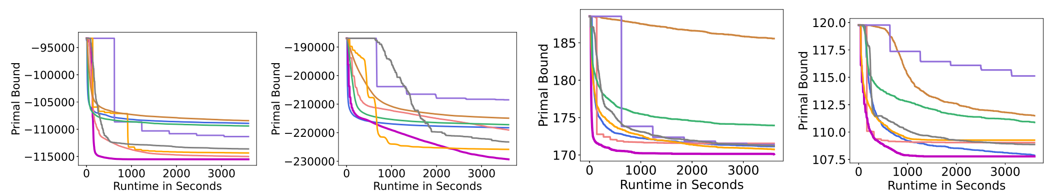
Next, we evaluate the performance with one additional metric: The *gap to virtual best* at time q for an approach is the normalized difference between its best primal bound found up to time q and the best primal bound found up to time q by any approach in the portfolio.

Figure 10 shows the full results on the best performing rate as a function of runtime. Figure 11 shows the full results on the gap to virtual best as a function of runtime.



(a) MVC-S (left) and MVC-L (right).

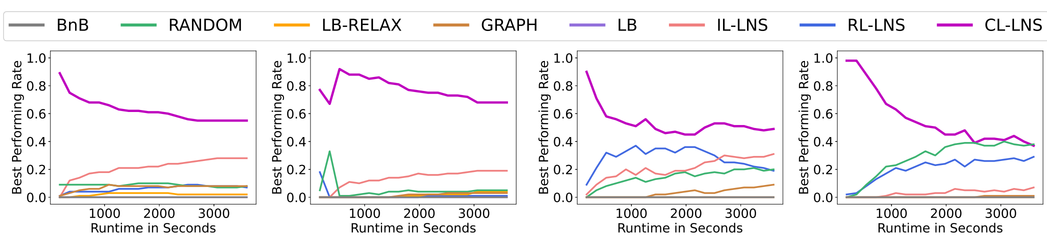
(b) MIS-S (left) and MIS-L (right).



(c) CA-S (left) and CA-L (right).

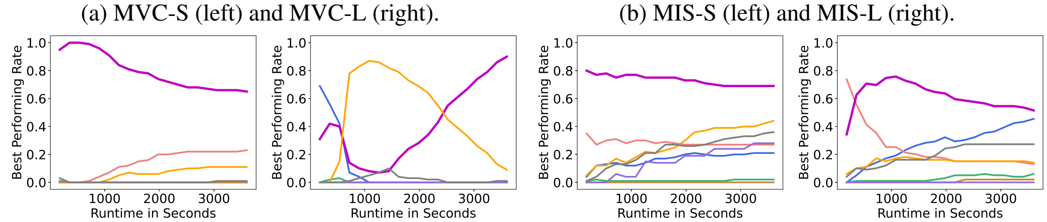
(d) SC-S (left) and SC-L (right).

Figure 9: The primal bound (the lower the better) as a function of time, averaged over 100 instances. For ML approaches, the policies are trained on only small training instances but tested on both small and large test instances.



(a) MVC-S (left) and MVC-L (right).

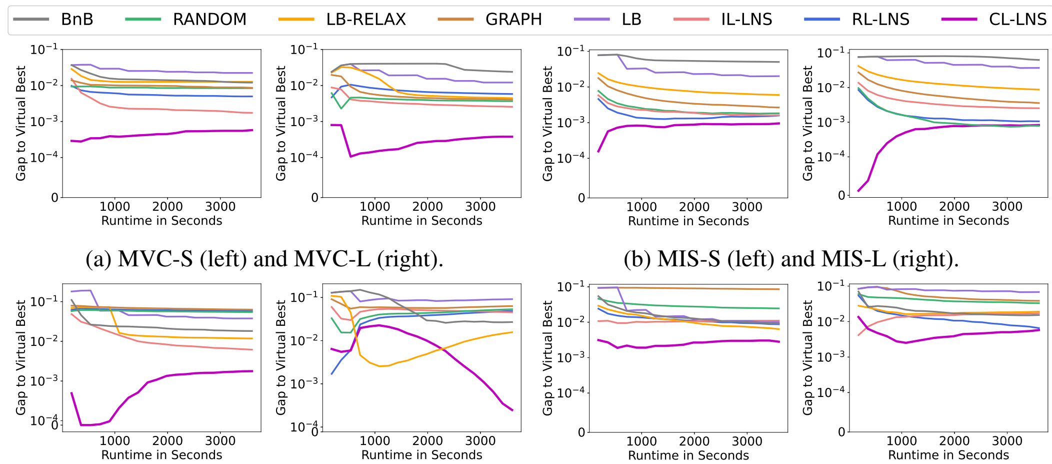
(b) MIS-S (left) and MIS-L (right).



(c) CA-S (left) and CA-L (right).

(d) SC-S (left) and SC-L (right).

Figure 10: The best performing rate (the higher the better) as a function of runtime over 100 test instances. For ML approaches, the policies are trained on only small training instances but tested on both small and large test instances.



(a) MVC-S (left) and MVC-L (right).

(b) MIS-S (left) and MIS-L (right).

(c) CA-S (left) and CA-L (right).

(d) SC-S (left) and SC-L (right).

Figure 11: The gap to virtual best (the lower the better) as a function of runtime, averaged over 100 test instances. For ML approaches, the policies are trained on only small training instances but tested on both small and large test instances.

Table 5: Test results on small instances: Primal bound (PB), primal gap (PG) (in percent), primal integral (PI) at 15 minutes time cutoff, averaged over 100 instances and their standard deviations.

	PB	PG (%)	PI	PB	PG (%)	PI
	MVC			MIS		
BnB	450.41±9.85	1.71±0.48	25.7±3.3	-1,981.72±23.49	6.66±0.89	74.2±4.4
LB	456.78±11.22	3.07±1.00	32.9±5.1	-2,047.01±18.76	3.58±0.60	62.4±3.8
RANDOM	447.33±11.33	1.02±1.28	11.5±11.3	-2,110.73±11.86	0.58±0.19	12.8±1.6
GRAPH	447.98±11.30	1.16±1.28	14.0±10.6	-2,104.62±12.23	0.87±0.17	18.5±1.7
LB-RELAX	449.23±11.49	1.43±1.51	19.6±10.9	-2,093.80±12.07	1.38±0.23	22.9±2.1
IL-LNS	444.50±9.69	0.40±0.28	10.2±5.5	-2,111.49±12.10	0.54±0.20	10.5±1.8
RL-LNS	446.12±10.10	0.76±0.36	11.9±2.9	-2,113.48±11.72	0.45±0.17	9.5±1.7
CL-LNS	**443.51±9.58**	**0.18±0.10**	**4.0±2.1**	**-2,114.66±12.42**	**0.39±0.19**	**6.4±1.6**
	CA			SC		
BnB	-112,703±1,682	3.06±0.70	67.4±16.6	173.26±13.00	2.28±1.34	45.9±13.0
LB	-108,647±2,227	6.55±1.42	140.7±9.9	173.83±12.93	2.60±1.31	70.6±15.6
RANDOM	-108,576±1,709	6.61±1.12	69.1±8.5	175.61±12.76	3.60±1.44	43.6±13.8
GRAPH	-107,189±1,977	7.81±1.15	84.7±9.8	187.69±14.24	9.77±2.17	89.9±19.9
LB-RELAX	-107,133±1,816	7.86±0.76	89.5±6.2	172.79±12.76	2.02±1.21	30.0±11.4
IL-LNS	-113,501±1,611	2.38±0.66	52.4±10.9	171.72±12.42	1.43±1.00	26.9±9.2
RL-LNS	-108,120±1,906	7.01±1.10	71.8±9.3	172.35±12.45	1.79±0.96	41.4±8.2
CL-LNS	**-115,499±1,626**	**0.66±0.33**	**33.3±6.8**	**170.27±12.21**	**0.59±0.67**	**11.7±7.4**

Table 6: Test results on small instances: Primal bound (PB), primal gap (PG) (in percent), primal integral (PI) at 30 minutes time cutoff, averaged over 100 instances and their standard deviations.

	PB	PG (%)	PI	PB	PG (%)	PI
	MVC			MIS		
BnB	449.67±9.69	1.55±0.44	40.2±6.6	-2,004.24±26.21	5.60±1.00	127.1±12.4
LB	454.89±11.55	2.66±1.16	58.2±14.1	-2,064.30±16.40	2.77±0.51	89.9±7.3
RANDOM	447.16±11.22	0.98±1.26	20.6±22.5	-2,115.23±11.82	0.37±0.16	16.9±2.7
GRAPH	447.75±11.39	1.11±1.30	24.2±22.1	-2,111.84±12.06	0.53±0.16	24.4±2.7
LB-RELAX	449.02±11.53	1.38±1.51	32.1±24.2	-2,102.85±11.97	0.95±0.19	33.0±3.6
IL-LNS	444.27±9.61	0.35±0.25	13.5±6.9	-2,115.30±12.04	0.36±0.18	14.4±3.2
RL-LNS	445.71±9.98	0.67±0.35	18.2±5.7	-2,116.64±11.53	0.30±0.15	12.7±2.9
CL-LNS	**443.48±9.56**	**0.17±0.09**	**5.5±3.6**	**-2,117.58±11.86**	**0.26±0.17**	**9.3±3.0**
	CA			SC		
BnB	-113,068±1,595	2.75±0.62	93.5±18.6	172.09±12.65	1.63±1.20	62.9±22.5
LB	-110,303±2,001	5.13±1.08	191.6±16.9	172.37±12.71	1.79±1.11	89.4±22.3
RANDOM	-109,040±1,685	6.21±1.05	126.8±17.6	174.70±12.75	3.10±1.38	73.4±24.6
GRAPH	-107,802±1,892	7.28±1.07	152.2±18.9	186.79±14.13	9.33±2.28	175.7±38.8
LB-RELAX	-114,103±1,521	1.86±0.57	109.5±9.4	171.60±12.43	1.36±1.02	44.6±19.3
IL-LNS	-114,621±1638	1.41±0.58	68.1±13.9	171.59±12.45	1.35±1.00	39.3±17.4
RL-LNS	-108,562±1,854	6.63±1.05	132.9±18.2	171.70±12.30	1.42±0.88	55.7±15.6
CL-LNS	**-115,513±1,621**	**0.65±0.32**	**39.1±11.6**	**170.16±12.13**	**0.53±0.63**	**16.7±12.3**

Table 7: Test results on small instances: Primal bound (PB), primal gap (PG) (in percent), primal integral (PI) at 45 minutes time cutoff, averaged over 100 instances and their standard deviations.

	PB	PG (%)	PI	PB	PG (%)	PI
	MVC			MIS		
BnB	449.28±9.77	1.46±0.42	53.7±9.9	-2,010.68±21.72	5.29±0.79	176.0±19.7
LB	453.84±11.65	2.44±1.26	80.7±24.6	-2,075.43±14.84	2.24±0.46	111.6±10.5
RANDOM	447.09±11.21	0.96±1.26	29.4±33.6	-2,116.96±11.54	0.29±0.15	19.8±3.9
GRAPH	447.42±11.19	1.04±1.27	33.9±33.4	-2,114.42±11.74	0.41±0.16	28.6±3.8
LB-RELAX	449.01±11.53	1.38±1.51	44.6±37.6	-2,106.88±11.40	0.76±0.20	40.6±5.0
IL-LNS	444.13±9.68	0.32±0.26	16.5±8.5	-2,117.43±11.79	0.26±0.17	17.2±4.5
RL-LNS	445.54±9.98	0.63±0.34	24.0±8.6	-2,117.79±11.34	0.25±0.14	15.2±4.1
CL-LNS	**443.48±9.56**	**0.17±0.09**	**7.1±5.1**	**-2,119.04±11.98**	**0.19±0.16**	**11.3±4.2**
	CA			SC		
BnB	-113,421±1,599	2.45±0.62	116.3±22.0	171.47±12.67	1.27±1.01	75.9±30.6
LB	-111,113±1,835	4.43±0.81	233.3±22.3	171.54±12.85	1.30±0.98	102.4±28.5
RANDOM	-109,253±1,697	6.03±1.02	181.9±26.2	174.15±12.94	2.78±1.30	99.8±35.3
GRAPH	-108,169±1,834	6.96±1.06	216.2±27.8	186.12±14.24	9.00±2.23	258.1±58.1
LB-RELAX	-114,268±1,512	1.72±0.57	125.3±13.6	170.98±12.38	1.00±0.88	54.8±25.6
IL-LNS	-114,871±1,602	1.20±0.56	79.7±17.3	171.55±12.47	1.33±0.97	51.2±25.7
RL-LNS	-108,776±1,813	6.44±1.04	191.7±27.0	171.35±12.29	1.22±0.85	67.5±22.6
CL-LNS	**-115,513±1,621**	**0.65±0.32**	**44.9±17.0**	**170.15±12.12**	**0.53±0.62**	**21.5±17.5**

Table 8: Test results on small instances: Primal bound (PB), primal gap (PG) (in percent), primal integral (PI) at 60 minutes time cutoff, averaged over 100 instances and their standard deviations.

	PB	PG (%)	PI	PB	PG (%)	PI
	MVC-S			MIS-S		
BnB	448.63±9.58	1.32±0.43	66.1±13.1	-2,014.85±20.04	5.10±0.69	222.8±25.9
LB	453.45±11.81	2.35±1.30	102.2±35.9	-2,079.07±14.34	2.07±0.44	130.9±13.6
RANDOM	447.06±11.21	0.96±1.26	38.0±44.8	-2,117.92±11.31	0.24±0.14	22.1±5.0
GRAPH	447.14±10.83	0.98±1.20	42.9±44.0	-2,116.15±11.58	0.32±0.15	31.8±5.0
LB-RELAX	449.01±11.53	1.38±1.51	57.0±51.2	-2,109.17±11.17	0.65±0.20	46.9±6.5
IL-LNS	444.00±9.73	0.29±0.23	19.2±10.2	-2,118.38±11.77	0.22±0.17	19.4±5.8
RL-LNS	445.45±9.99	0.61±0.34	29.6±11.5	-2,118.44±11.36	0.22±0.14	17.2±5.2
CL-LNS	**443.48±9.56**	**0.17±0.09**	**8.7±6.7**	**-2,119.78±12.14**	**0.15±0.15**	**12.8±5.4**
	CA-S			SC-S		
BnB	-113,608±1,611	2.28±0.59	137.4±25.9	171.22±12.50	1.13±0.95	86.7±37.9
LB	-111,342±1,732	4.23±0.75	272.1±26.9	171.39±12.81	1.22±0.97	113.7±35.2
RANDOM	-109,397±1,684	5.90±1.02	235.6±34.9	173.95±12.98	2.67±1.29	124.3±45.4
GRAPH	-108,422±1,775	6.74±1.03	277.7±36.5	185.57±14.17	8.74±2.13	337.8±76.4
LB-RELAX	-114,348±1,516	1.65±0.57	140.5±18.3	170.74±12.35	0.86±0.83	63.2±31.6
IL-LNS	-115,001±1,564	1.09±0.51	90.0±20.8	171.55±12.47	1.33±0.97	63.2±34.3
RL-LNS	-108,920±1,816	6.32±1.03	249.2±35.9	171.14±12.30	1.10±0.77	77.8±28.9
CL-LNS	**-115,513±1,621**	**0.65±0.32**	**50.7±22.7**	**170.11±12.10**	**0.50±0.58**	**26.2±12.8**

Table 9: Generalization results on large instances: Primal bound (PB), primal gap (PG) (in percent), primal integral (PI) at 15 minutes time cutoff, averaged over 100 instances and their standard deviations.

	PB	PG (%)	PI	PB	PG (%)	PI
	MVC			MIS		
BnB	919.96±12.38	4.06±0.38	36.8±3.4	-3,888.39±20.62	8.24±0.31	76.3±2.8
LB	907.06±12.46	2.69±0.36	32.7±3.2	-3,959.15±59.75	6.57±1.34	70.0±3.6
RANDOM	886.97±12.69	0.49±0.25	11.5±2.0	-4,215.32±15.86	0.52±0.12	12.4±1.0
GRAPH	888.28±12.61	0.64±0.26	18.0±2.3	-4,185.96±17.29	1.22±0.17	23.2±1.5
LB-RELAX	901.37±12.66	2.08±0.30	30.1±2.8	-4,148.06±19.51	2.11±0.20	33.2±1.8
IL-LNS	886.32±12.63	0.42±0.26	12.6±1.8	-4,203.74±16.80	0.80±0.17	14.8±1.7
RL-LNS	890.78±12.34	0.92±0.30	18.7±2.5	-4,215.17±15.97	0.53±0.14	11.5±1.2
CL-LNS	**883.18±12.52**	**0.06±0.05**	**7.7±1.5**	**-4,220.96±15.68**	**0.39±0.14**	**6.8±1.5**
	CA			SC		
BnB	-194,128±14,403	15.43±6.20	164.4±11.8	110.42±7.44	2.92±1.49	63.3±12.2
LB	-203,872±4,522	11.18±1.72	149.9±8.6	117.36±8.84	8.58±2.85	89.3±19.3
RANDOM	-215,183±2,670	6.26±0.74	75.8±6.0	112.91±7.72	5.04±2.03	59.9±16.8
GRAPH	-210,157±2,697	8.44±0.85	108.8±6.9	116.28±7.84	7.81±1.86	89.2±19.6
LB-RELAX	**-222,638±4,846**	**3.01±1.78**	102.5±12.3	109.66±7.24	2.25±1.51	36.2±13.3
IL-LNS	-211,938±3,323	7.67±1.22	89.9±8.9	109.12±6.97	1.79±1.26	32.4±10.7
RL-LNS	-216,788±2,730	5.56±0.85	**58.1±6.9**	109.38±6.89	2.03±1.08	83.6±8.8
CL-LNS	-218,510±2,989	4.81±0.81	61.3±7.1	**107.95±6.78**	**0.73±0.57**	**23.1±8.6**

Table 10: Generalization results on large instances: Primal bound (PB), primal gap (PG) (in percent), primal integral (PI) at 30 minutes time cutoff, averaged over 100 instances and their standard deviations.

	PB	PG (%)	PI	PB	PG (%)	PI
	MVC			MIS		
BnB	919.96±12.38	4.06±0.38	73.4±6.8	-3,888.39±20.62	8.24±0.31	150.5±5.6
LB	900.15±12.32	1.95±0.35	52.6±6.0	-4,009.23±71.94	5.39±1.59	123.1±15.1
RANDOM	886.39±12.71	0.43±0.25	15.6±3.9	-4,225.74±15.63	0.28±0.10	15.8±1.8
GRAPH	886.89±12.79	0.48±0.23	22.9±3.9	-4,206.29±16.76	0.74±0.16	31.6±2.7
LB-RELAX	887.64±12.21	0.57±0.23	39.4±4.4	-4,177.14±18.22	1.42±0.16	48.5±3.0
IL-LNS	885.58±12.65	0.33±0.26	15.9±4.0	-4,216.32±17.30	0.50±0.17	20.4±3.0
RL-LNS	888.89±12.64	0.71±0.30	25.8±4.8	-4,224.37±15.79	0.31±0.13	15.1±2.2
CL-LNS	**883.07±12.61**	**0.05±0.04**	**8.1±2.1**	**-4,226.65±15.56**	**0.26±0.13**	**9.7±2.6**
	CA			SC		
BnB	-216,772±13,060	5.58±5.42	257.1±56.4	109.39±7.26	2.02±1.36	84.4±22.2
LB	-206,526±3,750	10.03±1.39	245.1±19.2	116.43±8.97	7.84±2.88	162.6±39.2
RANDOM	-216,326±2,603	5.76±0.74	129.4±12.1	111.71±7.65	4.02±1.86	100.6±32.0
GRAPH	-213,142±2,713	7.14±0.78	177.6±13.2	112.74±7.64	4.91±1.80	141.7±31.1
LB-RELAX	**-225,154±4,366**	**1.91±1.60**	121.9±23.9	109.26±7.07	1.91±1.42	53.9±24.5
IL-LNS	-214,495±3,148	6.56±1.01	154.0±17.9	109.04±6.94	1.72±1.19	48.1±21.3
RL-LNS	-217,600±2,705	5.20±0.84	106.3±14.2	108.66±6.83	1.38±0.99	98.1±15.1
CL-LNS	-223,257±2,667	2.74±0.71	**95.0±12.5**	**107.78±6.64**	**0.58±0.45**	**28.6±12.6**

Table 11: Generalization results on large instances: Primal bound (PB), primal gap (PG) (in percent), primal integral (PI) at 45 minutes time cutoff, averaged over 100 instances and their standard deviations.

	PB	PG (%)	PI	PB	PG (%)	PI
	MVC			MIS		
BnB	907.44±12.77	2.73±0.43	107.2±9.4	-3,913.03±46.93	7.66±1.06	222.6±9.1
LB	894.77±12.41	1.36±0.30	66.3±8.2	-4,063.18±54.80	4.11±1.18	165.2±25.7
RANDOM	886.15±12.71	0.40±0.24	19.2±5.9	-4,230.24±15.56	0.17±0.09	17.8±2.5
GRAPH	886.53±12.72	0.44±0.23	27.0±5.7	-4,215.85±16.16	0.51±0.16	37.1±3.9
LB-RELAX	887.00±12.32	0.49±0.23	44.1±5.8	-4,191.17±17.76	1.09±0.16	59.7±4.2
IL-LNS	885.23±12.65	0.29±0.24	18.7±6.0	-4,222.04±16.64	0.36±0.16	24.2±4.3
RL-LNS	888.25±12.70	0.63±0.31	31.8±7.2	-4,228.78±15.68	0.20±0.12	17.3±3.1
CL-LNS	**883.07±12.61**	**0.05±0.04**	**8.6±2.7**	**-4,230.20±15.19**	**0.17±0.11**	**11.6±3.6**
	CA			SC		
BnB	-221,424±7,149	3.54±2.83	293.0±71.3	109.02±7.39	1.67±1.38	100.7±32.1
LB	-208,294±3,906	9.26±1.42	330.9±27.6	115.67±8.66	7.25±2.68	230.3±60.0
RANDOM	-216,819±2,611	5.54±0.73	180.1±18.1	111.24±7.54	3.63±1.81	134.9±46.8
GRAPH	-214,331±2,641	6.63±0.83	239.2±19.7	111.96±7.60	4.25±1.78	182.5±43.6
LB-RELAX	-225,641±4,235	1.70±1.53	138.1±37.1	109.26±7.07	1.91±1.42	71.1±36.5
IL-LNS	-216,705±3,062	5.59±0.97	208.7±25.7	109.04±6.94	1.72±1.19	63.6±31.8
RL-LNS	-217,987±2,711	5.03±0.81	152.3±21.4	108.22±6.75	0.99±0.87	108.6±21.2
CL-LNS	**-227,235±2,698**	**1.01±0.54**	**111.7±16.6**	**107.78±6.64**	**0.58±0.45**	**33.9±17.6**

Table 12: Generalization results on large instances: Primal bound (PB), primal gap (PG) (in percent) and primal integral (PI) at 60 minutes time cutoff, averaged over 100 instances and their standard deviations.

	PB	PG (%)	PI	PB	PG (%)	PI
	MVC-L			MIS-L		
BnB	904.41±12.95	2.41±0.40	130.2±11.1	-3,970.78±71.54	6.29±1.62	285.1±18.2
LB	893.56±12.62	1.22±0.30	77.8±10.1	-4,079.76±43.09	3.72±0.87	200.7±32.5
RANDOM	886.00±12.74	0.38±0.24	22.7±8.0	**-4,232.68±15.42**	**0.11±0.08**	19.0±3.1
GRAPH	886.34±12.67	0.42±0.23	30.9±7.6	-4,220.89±16.42	0.39±0.15	41.1±5.1
LB-RELAX	886.68±12.33	0.46±0.23	48.4±7.5	-4,199.04±17.54	0.91±0.16	68.6±5.5
IL-LNS	885.00±12.56	0.27±0.23	21.2±8.1	-4,225.28±16.25	0.29±0.15	27.1±5.5
RL-LNS	887.90±12.67	0.59±0.30	37.3±9.6	-4,231.52±15.97	0.14±0.12	18.9±4.1
CL-LNS	**883.07±12.61**	**0.05±0.04**	**9.1±3.4**	-4,232.50±14.86	0.12±0.11	**12.9±4.4**
	CA-L			SC-L		
BnB	-223,225±5,106	2.74±1.87	320.9±83.1	108.87±7.35	1.54±1.33	115.0±42.5
LB	-208,500±3,976	9.17±1.43	414.0±36.9	115.12±8.77	6.80±2.73	293.5±79.7
RANDOM	-217,204±2,612	5.37±0.75	229.2±24.4	110.88±7.55	3.31±1.79	166.4±61.3
GRAPH	-214,926±2,649	6.37±0.86	297.5±26.9	111.49±7.51	3.85±1.74	218.9±56.7
LB-RELAX	-225,848±4,201	1.61±1.50	153.0±50.3	109.26±7.07	1.91±1.42	88.3±48.9
IL-LNS	-219,074±3,278	4.56±0.98	254.2±33.4	109.04±6.94	1.72±1.19	79.1±42.4
RL-LNS	-218,273±2,725	4.91±0.81	197.0±28.5	107.87±6.74	0.66±0.72	116.2±27.1
CL-LNS	**-229,331±2,800**	**0.09±0.10**	**116.1±18.0**	**107.78±6.64**	**0.58±0.45**	**39.2±23.2**