FORGE: Foundational Optimization Representations from Graph Embeddings

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Combinatorial optimization problems are ubiquitous in science and engineering, yet learning-based approaches to accelerate their solution often require solving a large number of hard-to-solve optimization instances to collect training data, incurring significant computational overhead. Existing methods require training dedicated models for each problem distribution for each downstream task, limiting their scalability and generalization. In this work, we introduce FORGE, a method of pre-training a vector-quantized graph autoencoder on a large and diverse corpus of mixed-integer programming (MIP) instances in an unsupervised fashion without dependency on their solution. The vector quantization creates discrete code assignments that act as a vocabulary to represent MIP instances. We evaluate FORGE in both supervised and unsupervised settings. For the unsupervised setting, we show that FORGE embeddings effectively differentiate and cluster unseen instances. For the supervised setting, we fine-tune FORGE and show that *a single model* predicts both the variables for warm-starts and integrality gaps for cut-generation across multiple problem type distributions. Both predictions help improve performance of a SOTA, commercial optimization solver. Finally, we release our code and pre-trained FORGE weights to encourage further research and practical use of instance-level MIP embeddings.

1 Introduction

Combinatorial Optimization (CO) problems are fundamental in science and engineering with applications in multiple domains, like logistics, energy systems, network design, and recommendations [1, 2, 3]. Traditionally, CO problems have been addressed using carefully designed meta-heuristics and sophisticated solvers. These classical approaches often demand significant domain expertise and computational resources, especially as problem size and complexity scales.

Recent advances in Machine Learning (ML) have introduced promising alternatives for solving CO problems. The approaches for ML-guided CO fall mainly under two categories: (1) end-to-end models that attempt to predict solutions (or objective) without depending on meta-heuristics or solvers, and (2) hybrid methods that (2a) replace computationally intensive components of traditional solvers with learned models, e.g., learning to predict strong branching heuristics [4], and (2b) guide meta-heuristics with learning from feedback [5]. For a comprehensive overview of ML-guided CO, we refer readers to our Related Works in Section A.6 and the survey by Bengio et. al. [6].

Motivated by the successes of ML in other modalities like Natural Language Processing (NLP) and Computer Vision (CV), a natural question arises for CO: can we leverage the large number of publicly available MIP instances to develop a pre-trained, general-purpose, foundational model for MIP instances that is useful across multiple CO tasks across varying sizes and problem types? The current successes of ML for CO makes this direction imminent (see §A.6). However, many of these

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

ML-based methods face practical limitations - a significant drawback being their heavy dependency on computationally costly training that depends on carefully curating training datasets with desired properties and distributions of the underlying CO instances - leading to limited generalization. Moreover, training often depends on using optimization solvers in the first place to create labeled datasets, which defeats the purpose of improving solving for very hard instances that these solvers cannot deal with today. Adapting ML methods to new distributions and domains remains a challenge, hence, *designing foundational optimization representations in an unsupervised fashion applicable to multiple CO tasks* are a much-needed alternative. This is what we study in this paper.

CO problems, while having a large amount of data, span highly heterogeneous problem types (e.g., Set Cover vs. Combinatorial Auction) and present significant variability within each problem type. Most ML approaches to CO rely on Graph Neural Networks (GNN), as proposed in [4], which are effective at capturing local variable- and constraint-level information but struggle to capture meaningful global information at the instance level due to their inherent locality bias [7]-a critical limitation for CO problem solving. As such, while foundational models for other fields like NLP and CV exist, *to date, there exists no general-purpose MIP embeddings at the instance-level.*

With this in mind, we propose FORGE: **F**oundational **O**ptimization **R**epresentations from **G**raph **E**mbeddings. FORGE is a foundational model to generate MIP embeddings through a pre-training framework that learns *structural representations at the instance level* from a broad distribution of MIP instances without requiring access to their solutions. To achieve this, we incorporate two key ideas, one from NLP and the other from CV. As in NLP, we build a *vocabulary* to represent the latent space of CO problems, enabling instance-level representations. As in CV, we leverage *vector quantization* to preserve global information, which overcomes the issue of GNNs used for CO in previous works. By extending these crucial insights into the CO context, we make the following contributions:

- We propose FORGE, a foundational model to generate MIP embeddings. We show that FORGE effectively captures both local and global structures critical to CO. Unlike previous work, a *single* FORGE model, provides both instance-level representations, i.e., a single vector representation per MIP instance, and fine-grained variable- and constraint-embeddings.
- For the unsupervised setting, we show that the instance level MIP embeddings clusters previously unseen instances across multiple problem types with high accuracy.
- For the supervised setting, we show that pre-trained FORGE embeddings can be fine-tuned for various downstream tasks using a small number of labeled data. We evaluate FORGE on two radically different tasks to showcase its versatility: predicting variables for warm-starts and predicting integrality gap for cut-generation. Notably, a single pre-trained FORGE model is fine-tuned and used across diverse problem types like Combinatorial Auctions (CA), Set Covering (SC), Generalized Independent Set (GIS), and Minimum Vertex Cover (MVC).
- To enhance traditional solvers, we integrate predictions from FORGE into Gurobi [8] and to enhance ML-based CO methods, we incorporate FORGE embeddings into PS-Gurobi [9]. Across both, we show consistently lower primal gaps for a range of problem types and sizes.

2 FORGE: Unsupervised Approach to Learn MIP Embeddings

MIP instances are typically encoded as variable–constraint bipartite graphs with node features [4, 10, 11, 12]. GNNs are then trained for a single downstream task on a specific problem type—for example, warm-start variable prediction on Set Cover and Independent Set [9], or integrality-gap estimation [13]. Numerous variants follow this template (see §A.6). However, *all of these methods require supervision* and do not yield a general-purpose MIP embedding at the instance level. Hence, our goal is to learn the *structure* of MIP instances in an unsupervised manner. Figure 1 presents our overall architecture, which is composed of these main building blocks:

A) MIP-to-BP: Given a MIP instance, we generate its bipartite (BP) representation. Each node in the BP graph represents a constraint or a variable, with edges connecting variables to constraints they belong to. Each node is associated with node features and each edge is weighted by the coefficient of the variable in the constraint. FORGE uses only basic properties of the input instance - for each constraint node, it uses four features composed of its sense (i.e., $>$, $<$ or $=$) and the RHS value. For each variable node, it uses six features composed of its type (integer, binary, continuous), upper/lower bound, and the coefficient in the objective function (Figure 1(A)).

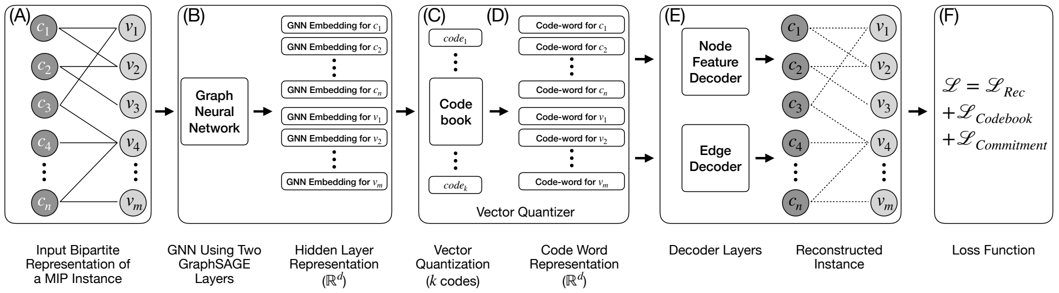

Figure 1: FORGE: The overall unsupervised approach for learning MIP embeddings. FORGE uses a vector quantized graph autoencoder (C-D) to reconstruct node features and edges and is pre-trained across a diverse set of problem types and difficulty levels, enabling it to learn generalizable structural representations without requiring access to optimal solutions.

B) BP-to-GNN: This bipartite graph is then passed into a GNN, to generate embeddings for each constraint and variable node. More specifically, FORGE uses two GRAPHSAGE [14] layers that project each input node into a d dimensional embedding space (Figure 1(B)).

C) Vector Quantized Codebook: As discussed earlier, while GNNs are good at capturing local variable- and constraint-level information, they struggle to capture meaningful global information at the instance level due to their inherent locality bias [7]. Preserving global structure is important in CO problems, especially to generalize across problem types. To do so, we introduce a vector quantized codebook with k discrete codes. These discrete codes act as a "vocabulary", akin to language models, across MIP instances of various domains and difficulties, thereby preserving global structure. The design follows the approaches developed in computer vision [15, 16, 17] and the structure-aware graph tokenizer extension as proposed in [18] (Figure 1(C)).

D) GNN-to-CW: GNN embeddings are passed into a vector quantizer with a codebook with k discrete codes. The codebook maps each variable and constraint node to a discrete code. Each code is then mapped into a d dimensional codeword (CW), yielding constraint and variable CW representations. These representations are the same dimension as the hidden GNN layers (Figure 1(D)).

E) CW-to-BP: We use the CW corresponding to each constraint and variable node to reconstruct the original bipartite representation of the MIP instance. These CW are passed into a linear node feature decoder and a linear edge decoder to reconstruct the input bipartite graph. By doing so, we obtain an *unsupervised method* that learns from the structure of MIP instances (Figure 1(E)).

F) Loss Function: The overall loss function minimizes the edge reconstruction loss, the node feature reconstruction loss as well as losses related to the vector quantization as shown in Figure 1(F). A detailed description of this loss function is given in the Appendix in Section A.1.

Once FORGE is trained in this unsupervised manner across a corpus of MIP instances, we obtain - (1) **Local representations**, where each node in the bipartite graph of the MIP instance is assigned a discrete code. Each code is further mapped to a codeword which becomes the variable and/or constraint embedding and (2) **Global representations**, where we leverage the distribution of codes for the given MIP instance. Each MIP instance can be represented with an embedding of size $vocabulary = |codebook|$ where each value denotes the frequency of codes present in the instance. This process is explained in Section A.2 in the Appendix.

Note that this form of unsupervised pre-training aims at learning the *structure* of MIP instances and is different from methods which aim to cast the discrete objective function of a MIP instance as a differentiable one and learn the solution using gradient descent in an end to end manner [19, 20, 21].

3 Initial Analysis of FORGE Embeddings

Given the absence of methods in the literature that can provide general-purpose MIP embeddings at the instance level, with the exception of earlier works that depend on creating hand-crafted features to classify MIP instances (e.g., [22]), we begin our analysis with a comparison of FORGE against two (ablation) baselines when clustering unseen instances. We investigate both the accuracy of the

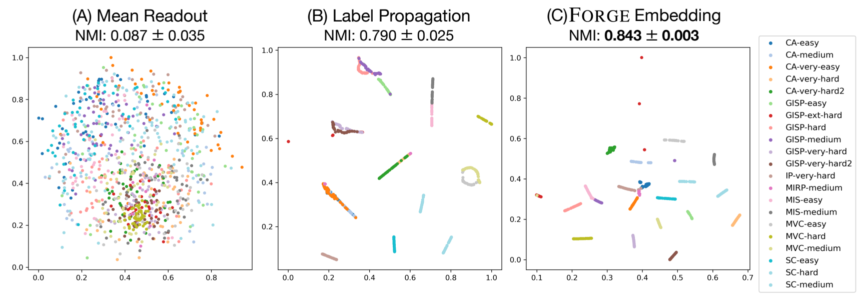

Figure 2: 2-D PaCMAP visualization of D-MIPLIB instances using MIP instance embeddings from (a) the mean readout of node embeddings from the GNN in FORGE, (b) the two-hop label propagation of input node features, and (c) the distribution of discrete codes produced by FORGE. Normalized Mutual Information (NMI), averaged over 10 runs of k-means clustering, is reported for each method.

clustering (quantitative) as well as visual inspection (qualitative). We train FORGE on a set of 600 MIP instances from MIPLIB [23], and test it on 1,050 unseen instances from Distributional MIPLIB (D-MIPLIB) [24] split across 21 problem type-difficulty pairs as shown in Figure 2. While MIPLIB is a mixed dataset, D-MIPLIB is categorized into different problem types and difficulties that serves as ground truth labels where each problem type-difficulty pair is a cluster to evaluate our embeddings.

Training: We use 600 instances from MIPLIB, sorted by size to ensure the resulting bipartite graphs fit on GPU memory. For additional training data, we generate two instances from each MIPLIB instance by randomly dropping 5% and 10% of constraints (note that dropping constraints does not impact feasibility). In total, we obtain 1,800 MIP instances to train FORGE. We use two GraphSage layers with $d = 1024$ dimensions and a *codebook* with $k = 5000$ codes (the size of the vocabulary)

Testing: These 1,050 instances are passed through FORGE to generate one embedding vector per MIP instance (as shown in Section A.2). As a baseline approach, we use the Mean Readout utilizing the GNN embeddings of the trained FORGE model. The Mean Readout method generates the MIP embedding of an instance by averaging all node features from the GNN hidden layer representation from Figure 1(B). This baseline also behaves as an ablation for FORGE without vector quantization. Given the weakness of GNN embeddings in capturing global structure [7], we expect this method to perform poorly. Alternatively, starting from the input static node features of the bipartite graph (Fig. 1(A)), we perform two-hop label propagation [25] and average the resulting node features.

Clustering Visualization and Accuracy: Figure 2 visualizes these embeddings, projected into two dimensions using PaCMAP [26]. Each dot represents an instance colored by its problem type. For quantitative evaluation, we run k-means clustering with 21 clusters, expecting one cluster for each problem type. We calculate the normalized mutual information score (NMI) between the ground truth problem type assignment and the k-means clusters generated by each method. The NMI scores are averaged over 10 runs of k-means clustering. As expected, Mean Readout over-smooths node features and loses global structure, yielding arbitrary clusters (NMI = 0.087). Label Propagation, working on sparse input node features only, separates domains better (NMI = 0.790) but still mixes some problem types. FORGE however, cleanly splits both problem type and difficulty (NMI = 0.843) despite zero exposure to D-MIPLIB instances, showing that it can successfully preserve global structure.

4 Experiments and Results

Given that FORGE embeddings reliably cluster unseen instances from diverse problem classes, we look at supervised settings directly aimed at improving MIP solving. We design experiments on downstream tasks that are (1) commonly applied in the literature to enable us with a fair comparison (2) radically different from each other, where we use the *same* FORGE model, to validate its general applicability across tasks, problems, and sizes and (3) are agnostic to the underlying MIP solver, i.e., we do not depend on internal access to specific solver procedures (e.g., within the branch-and-bound tree). With this motivation, we study two different downstream tasks: predicting the integrality gap of a MIP instance, as used in [13], and predicting variables for warm-starts, as used in [9] (Sections § 4.1 and § 4.2 respectively). Both tasks serve as primal heuristics to enhance MIP solving by obtaining better solutions faster. The integrality gap is used to generate a pseudo-cut added to the original problem formulation to tighten its bound. The warm-start variables are used to provide hints to the

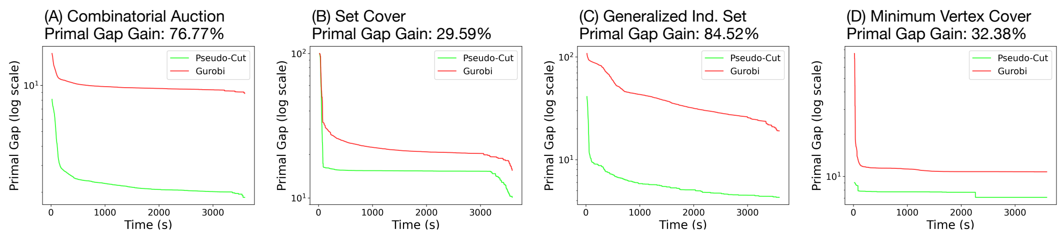

Figure 3: Gurobi vs. Gurobi + Pseudo-Cuts generated by FORGE. Each subplot shows the primal gap reduction averaged across 50 'very-hard' instances per problem type (lower is better).

solver to start the solving process from a promising initial assignment. The critical aspect of these sets of experiments is that we use **the same pre-trained FORGE model** to obtain general-purpose MIP embeddings that are then further fine-tuned on a small number of labeled data to learn prediction heads for **completely different tasks**. This is shown in Figure 7(A) in the Appendix.

4.1 Integrality Gap Prediction

The label collection, training and inference processes for integrality gap prediction are described in the Appendix in Section A.4. We summarize our results below.

Test Instances & Setup: The pre-trained FORGE model is fine-tuned with a prediction head using 450 labeled instances of 'easy', 'medium' and 'hard' difficulties for CA, SC and GIS problems, and predicts the integrality gap of 50 'very-hard' instances. Note that MVC instances are not part of the train set. A pseudo-cut is generated by adjusting the initial LP relaxation objective, specifically, by increasing it for minimization problems (or decreasing it for maximization problems) by the predicted gap. This pseudo-cut is incorporated into the original formulation as an additional constraint, enforcing that the integral objective must exceed (or fall below) the computed pseudo-cut value.

Comparison with the Commercial MIP Solver: We compare the default GUROBI solver on these 'very-hard' instances with and without our predicted pseudo-cut. Figure 3 shows the primal gap averaged over 50 'very-hard' instances each of CA, SC, GIS and MVC with a time limit of 3600 seconds. Without an exception, across all problem types, the use of pseudo-cuts generated by FORGE embeddings consistently results in significantly lower primal gaps over time. The gains range from 30% to 85%.

Comparison with SOTA ML Methods: To further evaluate generalization across problem types and sizes, we compare against the setup described in [13], where a GNN is trained on 38,256 instances from 643 generated problem types and tested on 11,584 instances spanning 157 problem types. We use Forge as is, *without* any additional training instances, and test on 17,500 previously unseen instances spanning 400 generated problem types. FORGE achieves a mean deviation of 18.63% in integrality gap prediction, outperforming the reported 20.14% deviation by almost 10%.

4.2 Warm-Start Prediction

The label collection, training and inference processes are described in the Appendix in Section A.5. We summarize our results below.

Test Instances & Setup: We start with the pre-trained FORGE model, fine-tuned with a prediction head using the labeled data, and then predict warm start variables of 50 'medium' instances each of CA, SC, GIS, and MVC. Note that MVC instances are not part of the train set.

Comparision with the Commercially MIP Solver: We compare GUROBI on these 'medium' instances with and without our predicted warm start variables. Figure 4 shows primal gaps averaged over 50 instances for each of CA, SC, GIS and MVC using Gurobi versus Gurobi warm started with predicted variables with a time limit of 3600 seconds. The warm-started variants consistently achieve lower primal gaps and converge to optimal solutions faster, demonstrating the practical utility of FORGE in accelerating MIP solving.

Comparison with SOTA ML Methods: As an additional validation of FORGE embeddings, we augment PS-Gurobi [9], a SOTA ML-based method that demonstrates strong performance relative

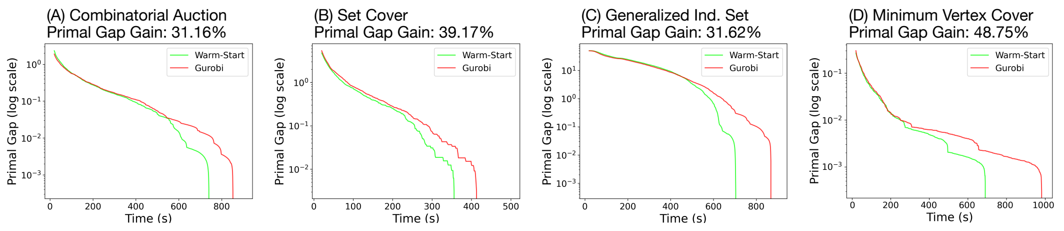

Figure 4: Gurobi vs. Gurobi + Warm Starts generated by FORGE. Each subplot shows the primal gap reduction averaged across 50 'medium' instances per problem type (lower is better). Gurobi + Warm Starts also converges to the optimal solution significantly faster.

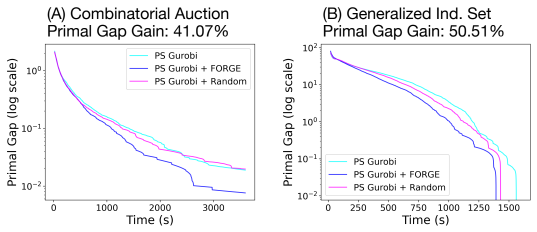

Figure 5: Performance comparison of three variants of PS-Gurobi: the original model, a version augmented with 64-dimensional random vectors, and a version augmented with FORGE embeddings (reduced to 64 dimensions via PCA). Evaluation is conducted on 50 'medium' instances each of CA and GIS problems. The FORGE-augmented PS-Gurobi consistently outperforms the baseline.

to other ML approaches. We concatenate the input node features for variables and constraints in PS-Gurobi with *unsupervised* embeddings generated by FORGE. We apply Principal Component Analysis (PCA) to reduce the dimensionality of FORGE embeddings from 1024 to 64 dimensions, thereby minimizing architectural modifications to the original model. To ensure that performance gains are attributed to the semantic content of the FORGE embeddings rather than increased model capacity, we also conduct a control experiment by augmenting PS-Gurobi with 64-dimensional random vectors. This comparison allows us to isolate the contribution of the learned embeddings to the overall performance. This is evaluated on the same 50 instances of the CA and GIS problems - two problem types that are common to both FORGE and PS-Gurobi. Results are shown in Figure 5 where PS-Gurobi + FORGE consistently outperforms the standalone PS-Gurobi with the gain in primal gaps being 41.07% and 50.51% for CA and GIS respectively. PS-Gurobi with random vectors occasionally surpasses the original model, likely due to the significant increase in input dimensionality - from 4 and 6 features for constraints and variables, respectively, to 68 and 70 - resulting in a larger model capacity that can yield marginal performance gains.

5 Conclusion & Future Work

We introduce FORGE, an unsupervised framework for learning structural representations of MIP instances without requiring access to their solutions. By using a vector-quantized graph autoencoder, FORGE effectively captures both local and global structural properties - an essential capability for addressing CO problems. The discrete codes produced by the vector quantization process behaves as a vocabulary of MIP instances. We demonstrate that these codes can meaningfully cluster previously unseen MIP instances and can be fine-tuned for diverse downstream tasks, including pseudo-cut generation via integrality gap prediction and warm-start variable prediction. Remarkably, a single model with only 3.25 million parameters is used across both tasks and multiple problem types and difficulty levels. We further show that the embeddings produced by FORGE can be seamlessly integrated into existing machine learning pipelines for CO, yielding improvements in solver performance. Looking ahead, the instance-level embeddings generated by FORGE offer promising opportunities for reinforcement learning approaches, such as guiding subproblem selection in Large Neighborhood Search frameworks. Exploring such integrations represents a compelling direction for future work. To support reproducibility and encourage further research, we release all code and pre-trained models at `https://anonymous.4open.science/r/forge-56FD`.

References

[1] Teodor Gabriel Crainic, Michel Gendreau, and Bernard Gendron. *Network Design with Applications to Transportation and Logistics*. Springer, 2021.

[2] Benjamin A Miller, Zohair Shafi, Wheeler Ruml, Yevgeniy Vorobeychik, Tina Eliassi-Rad, and Scott Alfeld. Attacking shortest paths by cutting edges. *ACM Transactions on Knowledge Discovery from Data*, 18(2):1–42, 2023.

[3] Serdar Kadıoğlu, Bernard Kleynhans, and Xin Wang. Integrating optimized item selection with active learning for continuous exploration in recommender systems. *Annals of Mathematics and Artificial Intelligence*, 92(6):1585–1607, 2024.

[4] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Neural Information Processing Systems*, 32:15554–15566, 2019.

[5] Junyang Cai, Serdar Kadioglu, and Bistra Dilkina. Balans: Multi-armed bandits-based adaptive large neighborhood search for mixed-integer programming problem. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-25*. International Joint Conferences on Artificial Intelligence Organization, 2025.

[6] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[7] Shengyu Feng, Weiwei Sun, Shanda Li, Ameet Talwalkar, and Yiming Yang. A comprehensive evaluation of contemporary ml-based solvers for combinatorial optimization. *arXiv preprint arXiv:2505.16952*, 2025.

[8] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024.

[9] Qingyu Han, Linxin Yang, Qian Chen, Xiang Zhou, Dong Zhang, Akang Wang, Ruoyu Sun, and Xiaodong Luo. A gnn-guided predict-and-search framework for mixed-integer linear programming. In *International Conference on Learning Representations*, 2023.

[10] Aaron M. Ferber, Jialin Song, Bistra Dilkina, and Yisong Yue. Learning pseudo-backdoors for mixed integer programs. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, volume 13292, pages 91–102, 2022.

[11] Morris Yau, Nikolaos Karalias, Eric Lu, Jessica Xu, and Stefanie Jegelka. Are graph neural networks optimal approximation algorithms? *Neural Information Processing Systems*, 37:73124–73181, 2024.

[12] Ziang Chen, Jialin Liu, Xinshang Wang, and Wotao Yin. On representing linear programs by graph neural networks. In *International Conference on Learning Representations*, 2023.

[13] Sirui Li, Janardhan Kulkarni, Ishai Menache, Cathy Wu, and Beibin Li. Towards foundation models for mixed integer linear programming. In *International Conference on Learning Representations*, 2025.

[14] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Neural Information Processing Systems*, pages 1024–1034, 2017.

[15] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[16] Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. Vector-quantized image modeling with improved vqgan. In *International Conference on Learning Representations*, 2022.

[17] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *Conference on Computer Vision and Pattern Recognition*, pages 11513–11522, 2022.

[18] Ling Yang, Ye Tian, Minkai Xu, Zhongyi Liu, Shenda Hong, Wei Qu, Wentao Zhang, Bin Cui, Muhan Zhang, and Jure Leskovec. Vqgraph: Rethinking graph representation space for bridging gnns and mlps. In *International Conference on Learning Representations*, 2024.

[19] Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. In *International Conference on Machine Learning*, pages 43346–43367, 2024.

[20] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Neural Information Processing Systems*, 33:6659–6672, 2020.

[21] Fanchen Bu, Hyeonsoo Jo, Soo Yong Lee, Sungsoo Ahn, and Kijung Shin. Tackling prevalent conditions in unsupervised combinatorial optimization: Cardinality, minimum, covering, and more. In *International Conference on Machine Learning*, pages 4696–4729, 2024.

[22] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC - instance-specific algorithm configuration. In Helder Coelho, Rudi Studer, and Michael J. Wooldridge, editors, *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 751–756. IOS Press, 2010.

[23] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021.

[24] Weimin Huang, Taoan Huang, Aaron M Ferber, and Bistra Dilkina. Distributional miplib: a multi-domain library for advancing ml-guided milp methods. *arXiv preprint arXiv:2406.06954*, 2024.

[25] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. *ProQuest number: information to all users*, 2002.

[26] Yingfan Wang, Haiyang Huang, Cynthia Rudin, and Yaron Shaposhnik. Understanding how dimension reduction tools work: an empirical approach to deciphering t-sne, umap, trimap, and pacmap for data visualization. *Journal of Machine Learning Research*, 22(201):1–73, 2021.

[27] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[28] Jiayi Zhang, Chang Liu, Xijun Li, Hui-Ling Zhen, Mingxuan Yuan, Yawen Li, and Junchi Yan. A survey for solving mixed integer programming via machine learning. *Neurocomputing*, 519:205–217, 2023.

[29] Abdelrahman Hosny and Sherief Reda. Automatic milp solver configuration by learning problem similarities. *Annals of Operations Research*, 339(1):909–936, 2024.

[30] Léo Boisvert, Hélène Verhaeghe, and Quentin Cappart. Towards a generic representation of combinatorial problems for learning-based approaches. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 99–108, 2024.

[31] Junyang Cai, Taoan Huang, and Bistra Dilkina. Multi-task representation learning for mixed integer linear programming. In Guido Tack, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 134–151, Cham, 2025. Springer Nature Switzerland.

[32] Darko Drakulic, Sofia Michel, and Jean-Marc Andreoli. Goal: A generalist combinatorial optimization agent learning. *arXiv preprint arXiv:2406.15079*, 2024.

[33] Ziang Chen, Jialin Liu, Xiaohan Chen, Wang Wang, and Wotao Yin. Rethinking the capacity of graph neural networks for branching strategy. *Neural Information Processing Systems*, 37:123991–124024, 2024.

[34] Furkan Cantürk, Taha Varol, Reyhan Aydoğan, and Okan Örsan Özener. Scalable primal heuristics using graph neural networks for combinatorial optimization. *Journal of Artificial Intelligence Research*, 80:327–376, 2024.

[35] Junyang Cai, Taoan Huang, and Bistra Dilkina. Learning backdoors for mixed integer linear programs with contrastive learning. In *European Conference on Artificial Intelligence*, volume 392, pages 2418–2425, 2024.

[36] Taoan Huang, Aaron M. Ferber, Yuandong Tian, Bistra Dilkina, and Benoit Steiner. Searching large neighborhoods for integer linear programs with contrastive learning. In *International Conference on Machine Learning*, volume 202, pages 13869–13890, 2023.

[37] Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Learning large neighborhood search policy for integer programming. *Neural Information Processing Systems*, 34:30075–30087, 2021.

[38] Jialin Song, Yisong Yue, Bistra Dilkina, et al. A general large neighborhood search framework for solving integer linear programs. *Neural Information Processing Systems*, 33:20012–20023, 2020.

[39] Elias B. Khalil, Bistra Dilkina, George L. Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *International Joint Conference on Artificial Intelligence*, pages 659–666, 2017.

[40] Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Towards omni-generalizable neural methods for vehicle routing problems. In *International Conference on Machine Learning*, pages 42769–42789, 2023.

[41] Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, André Hottung, Niels Wouda, Leon Lan, Kevin Tierney, and Jinkyoo Park. Routefinder: Towards foundation models for vehicle routing problems. In *ICML 2024 Workshop on Foundation Models in the Wild*, 2024.

[42] Jan Tönshoff, Berke Kisin, Jakob Lindner, and Martin Grohe. One model, any CSP: graph neural networks as fast global search heuristics for constraint satisfaction. In *International Joint Conference on Artificial Intelligence*, pages 4280–4288, 2023.

[43] Haonan Duan, Pashootan Vaezipoor, Max B Paulus, Yangjun Ruan, and Chris Maddison. Augment with care: Contrastive learning for combinatorial problems. In *International Conference on Machine Learning*, pages 5627–5642, 2022.

[44] Zohair Shafi, Benjamin A Miller, Tina Eliassi-Rad, and Rajmonda S Caceres. Accelerated discovery of set cover solutions via graph neural networks. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 191–208, 2025.

A Appendix

A.1 Loss Function

The overall loss function minimizes the edge reconstruction loss, the node feature reconstruction loss as well as losses related to the vector quantization. Specifically, the loss function is given by:

$$\mathcal{L} = \mathcal{L}_{Rec} + \mathcal{L}_{Codebook} + \mathcal{L}_{Commitment} \tag{1}$$

where given N nodes, input node feature $v_i \; \forall i \in N$, the adjacency matrix A and a matrix $\hat{X}$ composed of reconstructed input features v_i, the reconstruction loss, $\mathcal{L}_{Rec}$, the codebook loss, $\mathcal{L}_{Codebook}$, and commitment loss, $\mathcal{L}_{Commitment}$ are given by:

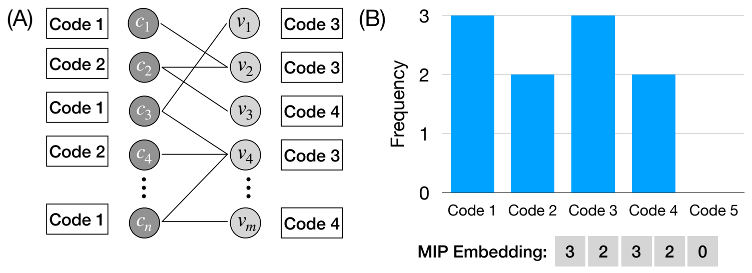

Figure 6: (A) Each node in the bipartite graph representation of the MIP instance is assigned a discrete code. (B) The distribution of these assigned codes yields the embedding of the MIP instance. The embedding dimension is the size of the codebook, i.e., $vocabulary = |codebook|$. This example uses 5 codes leading to the MIP embedding $\vec{emb} = [3, 2, 3, 2, 0]$.

$$\mathcal{L}_{Rec} = (A - \hat{X}\hat{X}^T)^2 + \frac{1}{N}\sum_{i=1}^{N}(\hat{v_i} - v_i)^2 \tag{2}$$

$$\mathcal{L}_{Codebook} = \frac{1}{N}\sum_{i=1}^{N}\|sg[h_i] - cw_i\|_2^2 \tag{3}$$

$$\mathcal{L}_{Commitment} = \frac{\alpha}{N}\sum_{i=1}^{N}\|sg[cw_i] - h_i\|_2^2 \tag{4}$$

Here, $sg[.]$ is the stop-gradient operator, h_i is the hidden layer representation of node i after the GNN forward pass and cw_i is the codeword corresponding to the code that node i has been assigned.

Intuitively, the codebook loss in Eq. 3 can be interpreted as k-means clustering, where the codewords cw_i (akin to cluster centroids) are updated to move closer to the node embeddings h_i and the node embeddings h_i are fixed in place due to the stop-gradient operator. Conversely, the commitment loss in Eq. 4 fixes the codewords cw_i using the stop-gradient operator, and instead, updates the embeddings h_i to move towards the codewords. The hyperparameter α weighs the importance of the commitment loss.

A.2 Generating MIP Instance Level Embeddings

To generate MIP instance level embeddings, we feed the input bipartite graph into FORGE, record the discrete codes assigned to every node, and treat the resulting code-frequency distribution as the instance embedding. This process is shown in Figure 6 with a codebook of size 5 (for brevity) and the resulting MIP embedding $\vec{emb} = [3, 2, 3, 2, 0]$ from the bipartite graph.

A.3 Supervised Fine Tuning Setup for the FORGE Model

The setup for training FORGE is identical to the setting in the initial clustering analysis, with two GraphSage layers with $d = 1,024$ dimensions and $k = 5000$ codes. The only difference is that we pre-train on the structure of the 1,800 MIPLIB instances as well as the 1,050 D-MIPLIB instances. In total, FORGE is trained on a corpus of 2,850 MIP instances.

For these experiments, we train on the ml.g5.xlarge AWS instance with a GPU with 24 GB of memory. Inference experiments were run on the ml.c5.12xlarge instance with 48 cores and 96 GB of RAM. To ensure consistency and fairness, all experiments were executed with GUROBI, a state-of-the-art commercial MIP solver [8], restricted to a single thread and a time limit of 3600 seconds. Unsupervised pre-training and integrality gap fine-tuning are run for 10 epochs with a learning rate of 10^{-4}, while the warm-start prediction task is trained for 25 epochs using a learning rate of 10^{-5}.

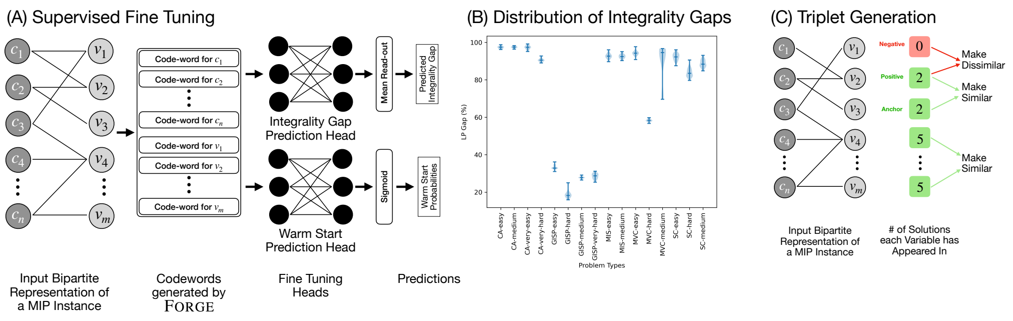

Figure 7: (A) The supervised fine tuning flow for FORGE. Integrality gap predictions are generated by using a mean readout and warm start probabilities are generated for each variable using a sigmoid function. (B) Distribution of integrality gaps for different problem type-difficulty pairs. (C) Procedure for constructing triplets for the triplet loss for warm start prediction. Each MIP instance is solved to obtain the top 5 solutions. Nodes appearing in the same number of solutions are paired as anchor and positive, while nodes absent from all solutions are designated as negatives.

A.4 Integrality Gap Prediction & Pseudo-Cut Generation

Integrality Gap Prediction: The integrality gap measures the ratio between the optimal solution of the LP relaxation and the optimal solution of the original integer program. Intuitively, this gap quantifies the quality of the approximation offered by the LP relaxation. A smaller gap means the LP relaxation is a good approximation and is close to value of the best integral solution. Here, we are interested in predicting the integrality gap of a given MIP instance, without having access to or solving for its best integral solution value, as also studied in [13].

Pseudo-Cut Generation: If we can predict the integrality gap, then we can generate a *pseudo-cut* and add it as an additional constraint to the model to immediately bound the optimal objective value from the initial LP relaxation. Note that, this cut is not guaranteed to be a valid cut at all times, hence it is a *pseudo-cut*. If the integrality gap prediction is incorrect, it risks over (or under) estimating the best objective value. This makes integrality gap prediction a challenging problem.

Training Instances: We do not expect this task to generalize between problem classes and/or sets of varying complexities of a given problem class. For instance, there is no reason for an LP gap of 70% to be the same between easy vs. hard Set Cover instances. Figure 7(B) shows the distribution of integrality gap across different problem types and difficulties. As we can see, the integrality gap can be anywhere from 5% to 95% with wide distributions in between. As such, there is no magic constant that one could use at all times heuristically, which makes integrality gap prediction a deliberate learning task. For training, we consider Combinatorial Auction (very-easy, easy, medium), Set Cover (easy, medium, hard), and Generalized Independent Set (easy, medium, hard) with 50 instances for each. In total, we obtain a total of 450 training instances. Note that this is a considerably smaller training set than initially used for training FORGE embeddings.

Label Collection: To generate training labels, each training instance is solved using Gurobi with a time limit of 120 seconds. Notice that, when creating this supervised dataset, we do not require solving the instance to optimality which can be computationally challenging. The numeric label is defined as the ratio (or gap) between the integer program solution at the time-out value and the LP relaxation solution. As mentioned, overestimating the integrality gap (or underestimating for maximization problems) can lead to suboptimal solutions, as the solver may terminate prematurely. In contrast, underestimating the gap is generally acceptable, as it still facilitates faster solve times without compromising solution quality. To account for this asymmetry, we adopt a conservative labeling strategy by setting a 120-second timeout when collecting labels. This often results in an underestimation of the true integrality gap, particularly for more challenging instances. By doing so, we reduce the risk of the model overestimating the gap during inference, thereby improving the reliability of the predicted cuts.

Supervised Fine-Tuning: Given the labeled data, to predict the integrality gap, a dense prediction head is added to pre-trained FORGE that takes as input the codewords assigned to each node and outputs a real number (Figure 7(A)). The mean readout across all nodes is used as the predicted integrality gap. As a regression task, this is trained with the mean absolute error loss in an end-to-end manner.

Additional Results - Prediction Accuracy: We start with the accuracy of our predictions on the test instances. We measure the deviation in terms of mean absolute error between the true integrality gap and the gap predicted by FORGE. On the 'very-hard' instances, FORGE achieves a deviation of 15.42%, 13.55%, 12.03% and 19.077% for CA, SC, GIS and MVC instances respectively. Note that MVC instances were not part of the train set.

While predicting integrality gaps may appear straightforward, training FORGE from scratch for this task led to a $\approx 33\%$ increase on average in mean absolute error across all problem types compared to fine-tuning a pre-trained model. This highlights the importance of unsupervised pre-training in capturing transferable structural patterns across diverse MIP instances.

A.5 Warm-Start Prediction & Guiding The Search

Warm Start Prediction: Recall that FORGE generates embedding vectors per instance and per variable/constraint. We fine tune the variable embeddings to predict which variables are likely to be in the optimal solution. These predictions are then passed into GUROBI as variable hints. A straightforward method of predicting which variables are likely to be part of the solution is to treat it as a binary classification problem and use binary cross entropy (BCE) loss. However, this poses challenges due to the large class imbalance where most variables are not part of the solution.

To address this issue, we produce 5 feasible solutions for a given instance. Variables from the instance are grouped based on how many of the 5 solutions each variable was repeated in. A variable is considered a "negative" variable if and only if it appears in *none* of the 5 solutions. Next, we use triplet loss [27] to fine tune the embeddings of the variables, where the triplet loss is given by

$$L(a, p, n) = max\{d(a_i, p_i) - d(a_i, n_i) + margin, 0\} \tag{5}$$

Here, a is the 'anchor' node, p is the 'positive' node, n is the 'negative' node and d is a distance function (euclidean distance in our case). The goal of triplet loss is to minimize the distance between the 'anchor' and 'positive' nodes while ensuring that the 'negative' node is at least 'margin' distance away from the 'anchor' node (margin is set to 2 in our case). This is shown in Figure 7(C).

In our setting, all variables appearing in the *same number of solutions* are treated as 'positive' and 'anchor' pairs. A key challenge in using triplet loss is to find good negative nodes, as picking nodes that are trivially negative does not aid learning. To pick negative nodes, we pick variables that have not appeared in any solution but are **closest** to the positive node in the **unsupervised** embedding space. Finally, the FORGE model is fine-tuned to predict warm start variables using a combination of triplet loss and BCE loss.

Training Instances: We collect 100 instances each from Combinatorial Auction (easy, medium), Set Cover (easy, medium, hard) and Generalized Independent Set (easy, medium) for a total of 700 training instances.

Label Collection: To generate training labels, we solve a given instance using Gurobi and a solution pool of 5. Here, we set a larger time limit of 300 seconds since we would like to get closer to the optimal solution.

Supervised Fine Tuning: The triplet loss is used to fine tune the GNN within FORGE to encourage meaningful separation in the embedding space. In parallel, a dense prediction head is added to FORGE that takes as input the codewords assigned to each node (Figure 7(C)). The model is trained using both the triplet loss and binary cross entropy loss on the prediction head.

To predict the warm start variables, we begin by generating a feasible solution using Gurobi with a 1-second timeout. The variables included in this solution are designated as *seed* variables. Since fine-tuning with triplet loss encourages nodes corresponding to solution variables to cluster in the embedding space, we identify additional variables whose embeddings lie within a fixed radius (set to 0.1) of the seed variables. Simultaneously, we select variables ranked in the top 5th percentile and bottom 10th percentile by the prediction head as positive and negative candidates, respectively. The intersection of variables identified by the embedding-based proximity search and the prediction head are then passed to Gurobi as solution hints.

A.6 Related Works

Recent advances in applying machine learning to combinatorial optimization, particularly Mixed Integer Linear Programming (MILP or MIP), have led to a diverse set of approaches. These methods can be broadly categorized based on their focus areas, such as solver configuration, branching strategies, heuristic design, and generalization across problem types.

Learning-Based Enhancements to MILP Solvers Several works have explored integrating supervised and reinforcement learning into traditional MILP solving pipelines. The survey by Zhang et. al. [28] categorizes these methods into two main groups: those enhancing the Branch-and-Bound process (e.g., branching variable prediction, cutting plane selection, node selection), and those improving heuristic algorithms like Large Neighborhood Search (LNS), Feasibility Pump, and Predict-and-Pick. These models are typically trained end-to-end, with reinforcement learning often relying on imitation learning.

In the context of solver configuration, Hosny et. al. [29] propose predicting solver parameters by leveraging similarities between problem instances. Their key assumption is that instances with similar costs under one configuration will behave similarly under others. Features include pre-solve statistics and tree-based metrics, with a triplet loss guiding the learning process using solved instance objectives. Kadiogluet. al. [22] propose the ISAC framework that takes a clustering-based approach to instance-specific algorithm configuration. It uses G-means to cluster problem instances and assigns configurations based on cluster membership. The method considers domain-specific features like cost-density ratios and root cost metrics for problems like SCP, MIP, and SAT.

To improve generalization, Boisvert et. al. [30] propose a generic representation for combinatorial problems using abstract syntax trees with 5 node types - variables, constraints, values, operators and a model node. While expressive, this approach results in large, computationally expensive graphs.

Multi-Task and Generalist Models Efforts to unify learning across tasks and problem types have also emerged. Cai et. al. [31] introduce a multi-task representation learning framework for MILP, training a shared backbone across tasks such as backdoor prediction and solver configuration prediction, followed by fine-tuning for specific problem types. Their method uses a similar bipartite graph representation, Graph Attention Networks (GAT), and contrastive loss, and is evaluated on problems like Combinatorial Auctions, Minimum Vertex Cover (MVC), and Maximum Independent Set (MIS). Similarly, Drakulic et. al. [32] present GOAL, a generalist agent for combinatorial optimization. It avoids GNNs, instead using mixed attention over edge and node matrices derived from bipartite graphs. Li et. al. [13] propose an LLM-based evolutionary framework that can generate a large set of diverse MILP classes and can be fine tuned to predict integrality gaps and branching nodes.

Graph Neural Networks for Branching and Heuristics Graph-based representations have become standard for encoding MILP instances. One of the earliest work by Gasse et. al. [4] uses GCNs to learn strong branching policies, introducing a bipartite graph structure and dual half-convolutions to facilitate message passing between constraints and variables. Chen et. al. [33] revisit GNN for MIPs and show that higher-order GNNs can overcome limitations identified via the 1-Weisfeiler-Lehman test, making all instances tractable for message passing. Canturk et. al. [34] introduce improvements to the standard GNN workflow for CO problems so that they generalize on instances of a larger scale than those used in training and proposes a two-stage primal heuristic strategy based on uncertainty quantification to automatically configure how solution search relies on the predicted decision values.

Along the lines of Backdoor learning, Cai et. al. [35] use Monte Carlo Tree Search to identify effective backdoors, training a GAT to score variables. Ferber et. al. [10] propose pseudo-backdoors, using one model that characterizes if a subset of variables is a good backdoor and another model to predict whether prioritizing this subset would lead to a smaller run time.

Learning Heuristics and Large Neighborhood Search (LNS) Policies A growing body of work focuses on learning heuristics, particularly for LNS. Huang et. al. [36] use expert hueuristics to create training data followed by random perturbations to create ‘negative’ samples. Then contrastive learning is used to train GATs to predict node probabilities. Other works by Wu et. al. [37] and Song et. al. [38] use deep reinforcement learning to learn destroy operators or decompositions, with rewards based on objective improvements. Khalil et. al. [39] model the success of heuristics at specific nodes by examining instance based characteristics and use logistic regression over a rich feature set, including LP relaxation and scoring metrics. Concretely, they learn a binary classifier to answer whether some heuristic H will find an incumbent solution at a given node N.

Problem Specific Solutions The vehicle routing problem (VRP) has garnered special attention from the community. Zhou et. al.[40] introduce a meta-learning framework for VRPs, enabling generalization across problem sizes and distributions. Berto et. al. [41] explore ML solutions for different kinds of VRPs like those including backhauls, multi-depots, duration limits, mixed backhaul, line hauls etc.. They use a common encoder for all VRP types with global attributes for problem type and local node attributes to capture customer specific attributes like location and demands.

Another problem that has garnered attention is the constraint satisfaction problem. Tonshoff et. al. [42] use GNNs to predict soft assignments, with reinforcement learning rewards based on constraint satisfaction improvements. Duan et. al. [43] propose a contrastive learning framework that generates label-preserving augmentations for SAT problems. These include techniques like unit clause propagation, pure literal elimination, and clause resolution, ensuring that the satisfiability of the instance remains unchanged while enhancing the model's robustness. Shafi et. al. [44] introduce Graph-SCP, a method that leverages features extracted from both bipartite and hypergraph representations of SCP instances. A GNN is then trained with these features to predict a promising subproblem where the optimal solution is likely to reside. This predicted subproblem is passed to a solver, effectively accelerating the overall solution process.

Unsupervised Approaches Unsupervised learning has also been explored in various forms. Karalias et. al. [20] introduce a framework that learns a probability distribution over nodes, optimizing a loss that bounds the probability of finding a solution. These are then decoded using a derandomization process. Bu et. al. [21] build on the work in [20] by formalizing objective construction and derandomization strategies. They derive explicit formulations tailored to a range of combinatorial problems, including facility location, maximum coverage, and robust graph coloring. Sanokowski et. al. [19] provide an approach for solving combinatorial optimization problems without labeled data by leveraging diffusion models to sample from complex discrete distributions. Their method avoids the need for exact likelihoods by optimizing a loss that upper bounds the reverse KL divergence. While FORGE falls in the domain of unsupervised approaches, we differ in that our goal is to not solve a given instance in an unsupervised manner, but rather to learn the graphical structure of various MIP instances in an unsupervised manner followed by supervised fine-tuning to aid in finding the solution.