OPT2023: 15th Annual Workshop on Optimization for Machine Learning

Contrastive Predict-and-Search for Mixed Integer Linear Programs

Taoan Huang TAOANHUA@USC.EDU
Aaron Ferber AMF272@CORNELL.EDU
Arman Zharmagambetov ARMANZ@META.COM
Yuandong Tian YUANDONG@META.COM
Bistra Dilkina DILKINA@USC.EDU

Abstract

Mixed integer linear programs (MILP) are flexible and powerful tool for modeling and solving many difficult real-world combinatorial optimization problems. In this paper, we propose a novel machine learning-based framework *ConPaS* that learns to predict solutions to MILPs with contrastive learning. For training, we collect high-quality solutions as positive samples and low-quality or infeasible solutions as negative samples. We then learn to make discriminative predictions by contrasting the positive and negative samples. During test time, we predict assignments for a subset of integer variables of a MILP and then solve the resulting reduced MILP to construct high-quality solutions. Empirically, we show that ConPaS achieves state-of-the-art results compared to other ML-based approaches in terms of the quality of and the speed at which the solutions are found.

1. Introduction

Combinatorial optimization (CO) concerns a wide variety of real-world problems, including resource allocation [20], network design [14] and production planning [24] problems, and the majority of them are NP-hard. Therefore, designing efficient and effective algorithms for CO is important but also challenging. Mixed integer linear programs (MILP) can flexibly encode and solve a broad family of CO and has been popular. It is a mathematical program that optimizes a linear objective subject to linear constraints, with some of the variables constrained to take integer values. In many real-world settings, MILPs from the same application domain often share similar structures and characteristics. The performance of MILP solvers crucially depends on how effective the heuristics are for that application. Recently, there has been an increased interest in data-driven heuristic designs for MILP for various decision-makings in BnB, including which variable to branch on[7, 16], which node to expand [12], which subset of variables to reoptimize [15, 25] and which heuristic to run next [4, 17].

There is another line of research that focuses on heuristics that generate optimal solutions to MILPs. In particular, it focuses on generating partial assignments of high-quality feasible solutions. Previously, [22] propose Neural Diving (ND) where they learn to partially assign values to integer variables and delegate the reduced sub-MILP to a MILP solver, e.g., SCIP. The fraction of variables to assign values to is controlled by a hyperparameter called the coverage rate. A SelectiveNet [9] is trained for each coverage rate that jointly decide which variables to fix and the values to fix to during testing. The main two disadvantages of ND are that: (1) enforcing variables to fixed values lead to low-quality or infeasibile solutions if the predictions are not accurate enough and (2) it requires training multiple SelectiveNet to obtain the appropriate coverage rate which is computationally expensive. To mitigate these issues, [11] propose a Predict-and-Search (PaS) framework that deploys

a search inspired by the trust region method. Instead of fixing variables, PaS searches for high-quality solutions within a pre-defined proximity of the predicted solution, which allows better feasibility and finding higher-quality solutions than ND. For both ND and PaS, the effectiveness (i.e., the quality of the solution found) and efficiency (i.e., the speed at which high-quality solutions are found) depend on the accuracy of the machine learning (ML) prediction and the number of variables (controlled by hyperparameters) whose values to fix.

In this paper, we propose a novel ML-based framework *ConPaS*, **Con**trastive **P**redict-**a**nd-**S**earch for MILPs. ConPaS uses contrastive learning to learn to predict (partial) solutions to MILPs. Similar to both ND [22] and PaS [11], we collect a set of optimal and near-optimal solutions as *positive samples*. Different from their approaches, we also collect infeasible and low-quality solutions as *negative samples*. Instead of using a binary cross entropy loss to penalize the inaccurate predictions for each variables separately, we use a contrastive loss that encourages the model to predicts solutions that are similar to the positive samples but dissimilar to the negative ones. Empirically, we test ConPaS on a variety of MILP benchmarks.We show that ConPaS achieves state-of-the-art anytime performance on finding high-quality solutions to MILPs, significantly outperforming other learning-based methods such as ND and PaS in terms of solution quality and speed.

2. Background

2.1. Mixed Integer Linear Programming (MILP)

A *mixed integer linear program (MILP)* $M = (\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, q)$ is defined as $\min \boldsymbol{c}^\mathsf{T}\boldsymbol{x}$ s.t. $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ and $\boldsymbol{x} \in \{0,1\}^q \times \mathbb{R}^{n-q}$, where $\boldsymbol{x} = (x_1, \ldots, x_n)^\mathsf{T}$ denotes the q binary variables and $n-q$ continuous 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 $\boldsymbol{x}$ is *feasible* if its satisfies all the constraints.

2.2. Neural Diving

Neural Diving (ND) [22] learns to generate a Bernoulli distribution for the solution values of binary variables. It learns the conditional distribution of the solution $\boldsymbol{x}$ given a MILP $M = (\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, q)$ defined as $p(\boldsymbol{x}|M) = \frac{\exp(-E(\boldsymbol{x}|M))}{\sum_{x' \in \mathcal{S}_\mathsf{p}^M} \exp(-E(\boldsymbol{x}'|M))}$, where $\mathcal{S}_\mathsf{p}^M$ is a set of optimal or near-optimal solutions to M and $E(\boldsymbol{x}|M)$ is an energy function of a solution $\boldsymbol{x}$ defined as $\boldsymbol{c}^\mathsf{T}\boldsymbol{x}$ if $\boldsymbol{x}$ is feasible or ∞ otherwise. ND learns $p_{\boldsymbol{\theta}}(\boldsymbol{x}|M)$ to approximate $p(\boldsymbol{x}|M)$ assuming conditional independence between variables $p_\theta(\boldsymbol{x}|M) = \prod_{i \leq q} p_\theta(x_i|M)$. Since the full prediction $p_\theta(\boldsymbol{x}|M)$ might not give a feasible solution, ND predicts only a partial solution controlled by the coverage rates and employs SelectiveNet [9] to learn which variables' values to predict for each coverage rates. ND uses binary cross entropy loss combined with the loss function in SelectiveNet to train the neural network. During testing, the input MILP M is then reduced to solving a smaller MILP after fixing the selected variables.

2.3. Predict-and-Search

Instead of using SelectiveNet to learn to fix variables, PaS searches for near-optimal solutions within a neighborhood based on the prediction. Specifically, given the prediction $p_\theta(x_i|M)$ for each binary variable, PaS greedily selects k_0 binary variables $\mathcal{X}_0$ with the smallest $p_\theta(x_i|M)$ and k_1 binary variables $\mathcal{X}_1$ with the largest $p_\theta(x_i|M)$, such that $\mathcal{X}_0$ and $\mathcal{X}_1$ are disjoint ($k_0 + k_1 \leq q$). PaS fixes all variables in $\mathcal{X}_0$ to 0 and $\mathcal{X}_1$ to 1 in the sub-MILP, but also allows Δ of the fixed variables to be

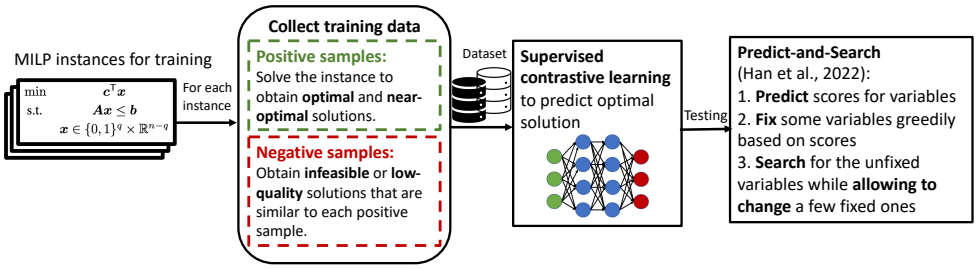


Figure 1: Overview of ConPaS.

flipped when solving it. Formally, let $B(\mathcal{X}_0, \mathcal{X}_1, \Delta) = \{\boldsymbol{x} : \sum_{x_i \in \mathcal{X}_0} x_i + \sum_{x_i \in \mathcal{X}_1} 1 - x_i \le \Delta\}$ and D be the feasible region of the original MILP, PaS solves the following optimization problem:

$$\min \boldsymbol{c}^\top \boldsymbol{x} \quad \text{s.t. } \boldsymbol{x} \in D \cap B(\mathcal{X}_0, \mathcal{X}_1, \Delta). \tag{1}$$

Restricting the solution space to $B(\mathcal{X}_0, \mathcal{X}_1, \Delta)$ can be seen as a generalization of the fixing strategy employed in ND where $\Delta = 0$. Though, ND uses sampling instead of greedy.

3. Solution Predictions with Contrastive Learning

In this section, we introduce our novel framework *ConPaS*, **Con**trastive **P**redict-**a**nd-**S**earch for MILPs. For a given MILP M, our goal is to use contrastive learning to predict the conditional distribution of the solution $p(\boldsymbol{x}|M)$, such that it leads to high-quality solutions fast when it is used to guide downstream MILP solvings. In this paper, we focus on solving optimization problem (1). We employ contrastive learning rather than other learning techniques because it has been theoretically demonstrated to be effective [27] and has empirically outperformed alternative approaches in related combinatorial optimization problems [5, 15, 21]. Figure 1 gives an overview of ConPaS.

3.1. Training Data Collection

In ConPaS, we use contrastive learning to learn to make discriminative predictions of optimal solutions by contrasting positive and negative samples. Since finding good assignments for integer variables is essentially the most challenging part of solving a MILP, we follow previous work to learn $p(\boldsymbol{x}|M)$ approximately as $p_\theta(\boldsymbol{x}|M) = \prod_{i \le q} p_\theta(x_i|M)$ where we mainly focus on predicting $p_\theta(x_i|M)$ for binary variables ($i \le q$). Therefore, our definition of positive and negative samples of solutions mainly concerns the partial solutions of binary variables. Now, we describe how we collect positive and negative samples, which is a crucial step in contrastive learning.

3.1.1. POSITIVE SAMPLES COLLECTION

For a given MILP M, we collect a set of optimal or near-optimal solutions $\mathcal{S}_\mathsf{p}^M$ as our positive samples following previous works [11, 22]. This is done by solving M exhaustively and collecting up to u_p best found solutions with the minimum objective values. In experiments, u_p is set to 50.

3.1.2. NEGATIVE SAMPLES COLLECTION

Negative samples are critical parts of contrastive learning to help distinguish between high-quality and low quality (or even infeasible) solutions. For a given MILP M, we collect a set of u_n negative

samples $\mathcal{S}_{\mathsf{n}}^M$ where $u_{\mathsf{n}} = \kappa|\mathcal{S}_{\mathsf{p}}^M|$ and κ is a hyperparameter to control the ratio between the number of positive and negative samples. We propose two different ways to collect negative samples:

Infeasible Solutions as Negative Samples. For each positive sample $\boldsymbol{x} \in \mathcal{S}_{\mathsf{p}}^M$, we sample κ infeasible solutions by randomly perturbing 10% of the binary variable values (i.e., flipping from 0 to 1 or 1 to 0). If the MILP M contains only binary variables, we validate that the perturbed solutions are indeed infeasible if it violates at least one constraint in M. Otherwise, we fix the binary variables to the values in the perturbed solutions and make sure that no feasible assignment of the continuous variables exists using the MILP solver. If less than κ negative samples are found after validating 2κ randomly perturbed samples, we increase the perturbation rate by 5% and repeat the same process.

Low-Quality Solutions as Negative Samples. For each positive sample $\boldsymbol{x} \in \mathcal{S}_{\mathsf{p}}^M$, we find the worst κ feasible solutions that differ from $\boldsymbol{x}$ in at most 10% of the binary variables. If the MILP $M = (\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, q)$ contains only binary variables, we find negative samples $\boldsymbol{x}'$ by solving the following Local Branching (LB) [6] MILP:

$$\begin{aligned} &\max \boldsymbol{c}^\mathsf{T}\boldsymbol{x}' \\ \text{s.t.} \quad &\boldsymbol{A}\boldsymbol{x}' \leq \boldsymbol{b}, \boldsymbol{x}' \in \{0,1\}^q \times \mathbb{R}^{n-q}, \\ &\textstyle\sum_{i \leq q: x_i = 0} x'_i + \sum_{i \leq q: x_i = 1} (1 - x'_i) \leq k. \end{aligned} \tag{2}$$

The above MILP is essentially solving the same problem as M, but with a negated objective function that tries to find solution x' as low-quality as possible and a constraint that allows changing at most k of the binary variables. After solving it, we consider only solutions as negative samples if they are worse than a given threshold. k is initially set to $10\% \times q$, but if less than κ negative samples are found, we increase the perturbation by 5% and resolve optimization problem (2).

If M contains continuous variables, the goal is to find partial solutions on binary variables, such that we get as low-quality solutions $\boldsymbol{x}'$ as possible when we fix the binary values and optimize for the rest of the continuous variables in M. Formally, solving for the partial solutions on binary variables $x'_1, \ldots, x'_q$ can be written as a maximin optimization problem:

$$\begin{aligned} &\max_{x'_1,\ldots,x'_q} \min_{x'_{q+1},\ldots,x'_n} \boldsymbol{c}^\mathsf{T}\boldsymbol{x}' \\ \text{s.t.} \quad &\boldsymbol{A}\boldsymbol{x}' \leq \boldsymbol{b}, \boldsymbol{x}' \in \{0,1\}^q \times \mathbb{R}^{n-q}, \\ &\textstyle\sum_{i \leq q: x_i = 0} x'_i + \sum_{i \leq q: x_i = 1} (1 - x'_i) \leq k. \end{aligned} \tag{3}$$

Solving the above maximin optimization exactly is prohibitively hard and, to the best of our knowledge, there are no general-purpose solvers for it [2, Chapter 7]. Therefore, we use a heuristic approach based on binary search where we iteratively solve the inner minimization problem and add a constraint $\boldsymbol{c}^\mathsf{T}\boldsymbol{x}' > \boldsymbol{c}^\mathsf{T}\boldsymbol{x}^*$ to enforce the next solution found is strictly better than the current best found solution $\boldsymbol{x}^*$ to the maximin problem.

3.2. Supervised Contrastive Learning

3.2.1. NEURAL NETWORK ARCHITECTURE

Following previous work on learning for MILPs [7, 22], we use a bipartite graph representation to encode the input MILP $M = (\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, q)$. 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. We learn $p_{\boldsymbol{\theta}}(x_i|M)$ represented by a graph convolutional network (GCN) parameterized by learnable weights $\boldsymbol{\theta}$.

3.2.2. TRAINING WITH A CONTRASTIVE LOSS

Given a set of MILP instances $\mathcal{M}$ for training, let $\mathcal{D} = \{(\mathcal{S}_{\mathsf{p}}^M, \mathcal{S}_{\mathsf{n}}^M) : M \in \mathcal{M}\}$ be the set of positive and negative samples for all training instances. A contrastive loss is a function whose value is low when the predicted $p_{\boldsymbol{\theta}}(x_i|M)$ is similar to the positive samples $\mathcal{S}_{\mathsf{p}}^M$ and dissimilar to the negative samples $\mathcal{S}_{\mathsf{n}}^M$. With similarity measured by dot products, we use an alternative form of InfoNCE [13, 23], a supervised contrastive loss, that takes into account the solution qualities of both positive and negative samples:

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

where we let $\frac{1}{\tau(\boldsymbol{x}|M)} \propto -E(\boldsymbol{x}|M)$ if $\boldsymbol{x}$ is feasible to M where is the same energy function used in previous works [11, 22]; otherwise $\tau(\boldsymbol{x}|M)$ is set to a constant τ (we set $\tau = 1$ in experiments). Intuitively, setting $\tau(\boldsymbol{x}|M)$ in this manner encourages the predictions $p_{\boldsymbol{\theta}}(\boldsymbol{x}|M)$ to be more similar to positive samples $\boldsymbol{x}_{\mathsf{p}}$ with better objectives.

3.3. Predict-and-Search

We apply the predicted solution to reduce the search space of the input MILP similarly to previous work. Specifically, we use the same search strategy as PaS [11] where we greedily select $\mathcal{X}_0$ and $\mathcal{X}_1$ based on the prediction and solve the optimization problem defined by Equation (1).

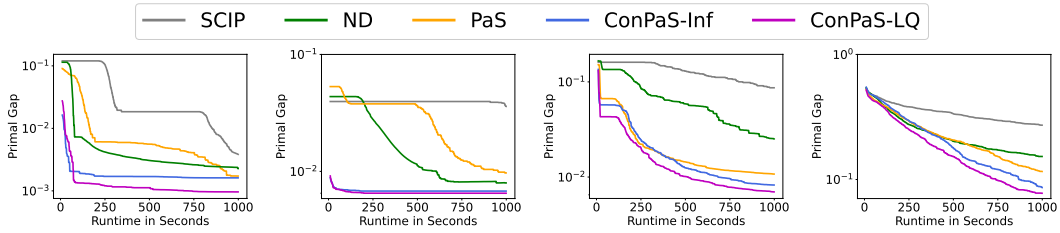
4. Empirical Evaluation

Benchmark Problems We evaluate on four NP-hard benchmark problems: (1) the minimum vertex cover (MVC) and (2) maximum independent set (MIS) instances are generated according to the Barabasi-Albert random graph model [1], with 6,000 nodes and an average degree of 5; (3) CA instances are generated with 2,000 items and 4,000 bids according to the arbitrary relations [18]; and (4) IP instances are taken from the NeurIPS 2021 ML4CO competition [8]. For each benchmark problem, we have 400, 100 and 100 instances in the training, validation and test sets, respectively.

Baselines We compare ConPaS with three baselines: (1) SCIP (v8.0.1) [19], the state-of-the-art open-source ILP solver, with the aggressive mode fine-tuned to focus on improving the objective value; (2) Neural Diving (ND) [22]; and (3) Predict-and-Search (PaS) [11].

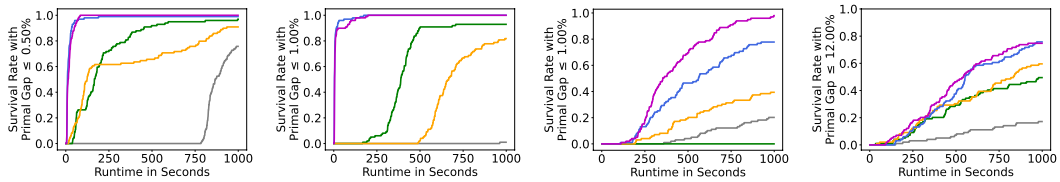
Metrics We use the following metrics to evaluate all approaches: (1) The *primal gap* [3] is the normalized difference between the primal bound v and a precomputed best known objective value v^*; and (2) The *survival rate* to meet a certain primal gap threshold is the fraction of instances with primal gaps below the threshold [26].

Hyperparameters We conduct experiments on 2.4 GHz Intel Core i7 CPU with 16 GB memory. Training is done on a NVIDIA P100 GPU with 32 GB memory. For data collection, we collect 50 best found solutions for each training instance for an hour using Gurobi [10]. For testing, we set the time limit to 1,000 seconds to solve the reduced MILP of each test instance with SCIP [19].



(a) MVC. (b) MIS. (c) CA. (d) IP.

Figure 2: The primal gap as a function of runtime.



(a) MVC. (b) MIS. (c) CA. (d) IP.

Figure 3: The survival rate as a function of runtime to meet a certain primal gap threshold.

Results We test two variants of ConPaS, denoted by ConPaS-Inf and ConPaS-LQ, that use infeasible solutions and low-quality solutions as negative samples, respectively. Figure 2 shows the primal gap as a function of runtime. Overall, SCIP performs the worst. PaS achieves lower average primal gaps than ND on three of the problems at 1,000 seconds runtime cutoff. Both ConPaS-Inf and ConPaS-LQ show significantly better anytime performance than all baselines on all benchmark problems. ConPaS-LQ performances slightly better than ConPaS-Inf. Figure 3 shows the survival rate to meet a certain primal gap threshold. The primal gap threshold is chosen as the medium of the average primal gap at 1,000 seconds runtime cutoff among all approaches rounded to the nearest 0.50%. ND surprisingly has the lowest survival rate (even lower than SCIP) on the CA instances, indicating high variance in performance of both SCIP and ND,but ND is better than both SCIP and PaS on both the two graph optimization problems. PaS has higher survival rates on the CA and IP instances. ConPaS-Inf and ConPaS-LQ have the best survival rate at 1,000 minutes runtime cutoff on all instances. Specifically, on the MVC, MIS and CA instance, at the runtime cutoffs when they both first reach 100% survival rates, the best baseline only achieves about 10%-80% survival rates.

5. Conclusion

We proposed ConPaS, a contrastive predict-and-search framework for MILPs. We learned to predict high-quality solutions by contrasting optimal and near-optimal solutions with infeasible or low-quality solutions. In testing, we solved a reduced MILP by restricting the search space to proximity to the predicted solutions. In experiments, we showed that ConPaS found solutions not only better but also faster than two state-of-the-art ML-based approaches. ConPaS also demonstrated generalizability to larger instances that were unseen during training. Solving MILPs based on solution predictions, such as ConPaS, ND and PaS, does not guarantee completeness or optimality. Therefore, it is important and interesting future work to integrate them in the Branch-and-Bound search.

References

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

[2] Yasmine Beck and Martin Schmidt. A gentle and incomplete introduction to bilevel optimization. 2021. URL https://optimization-online.org/?p=17182.

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

[4] Antonia Chmiela, Elias Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. Learning to schedule heuristics in branch and bound. *Advances in Neural Information Processing Systems*, 34:24235–24246, 2021.

[5] 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. PMLR, 2022.

[6] Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical programming*, 98(1):23–47, 2003.

[7] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[8] Maxime Gasse, Simon Bowly, Quentin Cappart, Jonas Charfreitag, Laurent Charlin, Didier Chételat, Antonia Chmiela, Justin Dumouchelle, Ambros Gleixner, Aleksandr M Kazachkov, et al. The machine learning for combinatorial optimization competition (ml4co): Results and insights. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 220–231. PMLR, 2022.

[9] Yonatan Geifman and Ran El-Yaniv. Selectivenet: A deep neural network with an integrated reject option. In *International conference on machine learning*, pages 2151–2159. PMLR, 2019.

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

[11] 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 *The Eleventh International Conference on Learning Representations*, 2022.

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

[13] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross 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.

[14] Taoan Huang and Bistra 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.

[15] 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*, pages 13869–13890. PMLR, 2023.

[16] Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

[17] Elias B Khalil, Bistra Dilkina, George L Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *Ijcai*, pages 659–666, 2017.

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

[19] Stephen J Maher, Tobias Fischer, Tristan Gally, Gerald Gamrath, Ambros Gleixner, Robert Lion Gottwald, Gregor Hendel, Thorsten Koch, Marco Lübbecke, Matthias Miltenberger, et al. The scip optimization suite 4.0. 2017.

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

[21] Maxime Mulamba, Jayanta Mandi, Michelangelo Diligenti, Michele Lombardi, Victor Bucarey Lopez, and Tias 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.

[22] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid Von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

[23] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

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

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

[26] Nicolas Sonnerat, Pengming Wang, Ira Ktena, Sergey Bartunov, and Vinod Nair. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201*, 2021.

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