Principled Data Augmentation for Learning to Solve Quadratic Programming Problems

Chendi Qian **Christopher Morris**
Department of Computer Science
RWTH Aachen University
Aachen, Germany
chendi.qian@log.rwth-aachen.de

Abstract

Linear and quadratic optimization are crucial in numerous real-world applications, ranging from training machine learning models to solving integer linear programs. Recently, learning-to-optimize methods (L2O) for linear (LPs) or quadratic programs (QPs) using message-passing graph neural networks (MPNNs) have gained traction, promising lightweight, data-driven proxies for solving such optimization problems. For example, they replace the costly computation of strong branching scores in branch-and-bound solvers, thereby reducing the need to solve many such optimization problems. However, robust L2O MPNNs remain challenging in data-scarce settings, especially when addressing complex optimization problems such as QPs. This work introduces a principled approach to data augmentation tailored for QPs via MPNNs. Our method leverages theoretically justified data augmentation techniques to generate diverse yet optimality-preserving instances. Furthermore, we integrate these augmentations into a self-supervised contrastive learning framework, thereby pretraining MPNNs for improved performance on L2O tasks. Extensive experiments demonstrate that our approach improves generalization in supervised scenarios and facilitates effective transfer learning to related optimization problems.

1 Introduction

Linear and quadratic optimization problems are fundamental problems in machine learning, operations research, and scientific computing [Boyd and Vandenberghe, 2004, Nocedal and Wright, 1999]. Many real-world applications, such as resource allocation, logistics, and training machine learning models, rely on efficiently solving large-scale *linear programming* (LPs) and *quadratic programming* (QPs). In addition, they play a key role in state-of-the-art integer-linear optimization solvers, allowing the computation of lower bounds, and are the basis for crucial heuristics such as *strong branching* for variable selection [Achterberg et al., 2005].

Recently, machine learning techniques, particularly *message-passing graph neural networks* (MPNNs) [Gilmer et al., 2017, Scarselli et al., 2008] have been explored for *learning-to-optimize* (L2O) approaches, aiming to learn to solve LPs and QPs in a data-driven fashion [Bengio et al., 2021, Cappart et al., 2023], enhancing solver efficiency and improving generalization across different problem instances. For example, Gasse et al. [2019] used MPNNs to imitate the costly strong branching score for variable selection in integer-linear optimization solvers, which requires solving many LPs during the solving process. However, training robust models for L2O remains challenging due to the scarcity of labeled data, especially for complex optimization formulations such as QPs.

In response to this challenge, *self-supervised learning* (SSL) has emerged as a powerful paradigm for pretraining models on large, unlabeled datasets [Liu et al., 2021]. *Contrastive learning*, a key approach in SSL, has demonstrated significant success in graph-based tasks by leveraging data augmentation

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

techniques to create diverse training examples You et al. [2020]. Despite advances in contrastive learning and L2O with MPNNs, a gap exists in integrating these two paradigms for LPs and QPs. Additionally, designing effective and principled data augmentation strategies for LPs and QPs remains non-trivial due to the structural constraints and optimality conditions inherent in these problems.

Present work In this work, we propose a principled approach to data augmentation for LPs and QPs, specifically designed to enhance the performance of MPNN-based L2O methods for such problems; see Fig. 1 for a high-level overview. Concretely, our contributions are as follows:

1. We introduce a set of novel and theoretically grounded augmentation techniques for LPs and QPs that preserve optimality while generating diverse training instances;
2. We apply these augmentations in both supervised and self-supervised settings, including contrastive pretraining of MPNNs to enhance downstream performance;
3. We empirically evaluate our approach on synthetic and benchmark datasets, showing that pretraining with our augmentations improves generalization and transferability across problem classes.

By bridging the gap between data augmentation and L2O for LPs and QPs, our work offers a new perspective on enhancing neural solvers through supervised learning and self-supervised pretraining. The proposed augmentations improve generalization in data-scarce settings, enabling more efficient and robust learning-based optimization methods.

1.1 Related work

This section reviews relevant literature on L2O, graph data augmentation, graph contrastive learning, and synthetic instance generation for LP and MILP problems.

MPNN and L2O Message Passing Neural Networks (MPNNs) [Gilmer et al., 2017, Scarselli et al., 2008] have been extensively studied and are broadly categorized into spatial and spectral variants. Spatial MPNNs [Bresson and Laurent, 2017, Duvenaud et al., 2015, Hamilton et al., 2017, Veličković et al., 2017, Xu et al., 2018] follow the message-passing paradigm introduced by Gilmer et al. [2017]. MPNNs have shown strong potential in learning to optimize (L2O). A widely adopted approach represents MILPs using constraint-variable bipartite graphs [Chen et al., 2022, Ding et al., 2020, Gasse et al., 2019, Khalil et al., 2022, Qian et al., 2024a]. Recent work has also aligned MPNNs with various optimization algorithms, including interior-point methods (IPMs) [Qian et al., 2024a, Qian and Morris, 2025], primal-dual hybrid gradient (PDHG) [Li et al., 2024a], and distributed algorithms [Li et al., 2024b, 2025]. From a theoretical perspective, several studies have analyzed the expressiveness of MPNNs in approximating solutions to linear and quadratic programming [Chen et al., 2022, 2023, 2024b,a, Wu et al., 2024].

Graph data augmentation Graph data augmentation is central to learning-based optimization, especially under data scarcity. Common strategies include feature-wise perturbations [You et al., 2020, Zhu et al., 2020, Suresh et al., 2021, Zhu et al., 2021, Thakoor et al., 2021, Bielak et al., 2022, Hu et al., 2023], structure-wise modifications such as node dropping and edge perturbation [Rong et al., 2019, Hassani and Khasahmadi, 2020, Zhu et al., 2020, Qiu et al., 2020, Papp et al., 2021, Zhu et al., 2021, Bielak et al., 2022, Hu et al., 2023], and spectral-domain augmentations [Yang et al., 2024a, Liu et al., 2022a, Lin et al., 2022, Wan et al., 2024], the latter aligning with recent works like S3GCL. Learning-based approaches [Yang et al., 2020, Zhao et al., 2021, Liu et al., 2022b, Yin et al., 2022, You et al., 2021, Wu et al., 2023] further enable adaptive augmentation via trainable modules. Complementary perspectives include graph rewiring [Topping et al., 2021, Karhadkar et al., 2022, Arnaiz-Rodríguez et al., 2022, Qian et al., 2023, Gutteridge et al., 2023, Barbero et al., 2023, Qian et al., 2024b] and structure learning [Jin et al., 2020, Liu et al., 2022c, Zou et al., 2023, Zhou et al., 2023, Fatemi et al., 2023], both of which can be viewed as task-specific augmentation techniques. For a broader overview, including robustness and self-supervised settings, see [Zhao et al., 2022, Ding et al., 2022].

Graph contrastive learning Graph augmentations also play a central role in graph self-supervised learning (SSL), which aims to learn transferable representations without labeled supervision. Graph SSL methods are categorized into contrastive, generative, and predictive approaches [Wu et al., 2021].

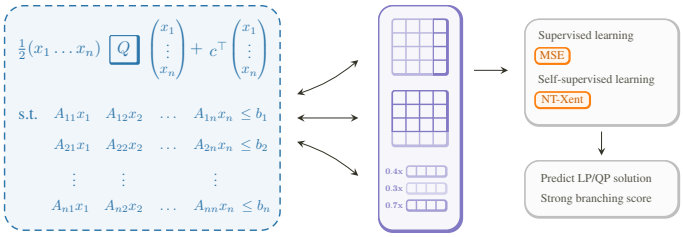

Figure 1: Overview of our framework for principled data augmentation for quadratic optimization problems. Given a QP instance, we apply transformations (e.g., adding/removing/scaling variables or constraints; see Section 2.2) to generate new instances, thereby augmenting the training dataset. We then use standard supervised learning or contrastive learning to train an MPNN.

We focus on contrastive ones, which operate at three levels: *global-to-global* (G2G), *local-to-local* (L2L), and *local-to-global* (L2G). G2G methods generate contrasting views of entire graphs, as in GraphCL [You et al., 2020] with structural perturbations, extended by CSSL [Zeng and Xie, 2021], GCC [Qiu et al., 2020] using random walks, AutoGCL [Yin et al., 2022] with learned augmentations, and AD-GCL [Suresh et al., 2021] with an adversarial objective. The only contrastive method targeting LPs is Li et al. [2024c], which adopts a CLIP-style [Radford et al., 2021] formulation. L2L methods contrast node-level views, e.g., GRACE [Zhu et al., 2020], GCA [Zhu et al., 2021], Graph Barlow Twins [Bielak et al., 2022], BGRL [Thakoor et al., 2021], and REGCL [Ji et al., 2024], with S3GCL [Wan et al., 2024] introducing spectral views. L2G methods such as DGI [Veličković et al., 2018], MVGRL [Hassani and Khasahmadi, 2020], and InfoGraph [Sun et al., 2019] contrast local and global representations to maximize mutual information.

While general-purpose augmentations are well-studied, few works target L2O problems. Duan et al. [2022] designs satisfiability-preserving augmentations for SAT problems, and Huang et al. [2023] samples neighborhoods around expert solutions. A concurrent work [Zeng et al., 2025] explores constraint permutations in MILPs. Compared with these works, we propose efficient, principled transformations, tailored for supervised and contrastive learning in linear and quadratic programming.

Instance generation While random instance generators exist for LPs and MILPs [Gasse et al., 2019, 2022], data scarcity has driven model-based approaches, including stress testing [Bowly, 2019], VAEs [Geng et al., 2023], and diffusion models [Zhang et al., 2024]. Other works reconstruct or adapt instances [Wang et al., 2023, Guo et al., 2024, Yang et al., 2024c], or generate them from code or substructures [Li et al., 2024c, Liu et al., 2024]. In contrast, our approach is model-free, mathematically principled, and generates new instances through transformations with analytically tractable solutions.

1.2 Background

We introduce notations and define MPNNs, LPs, and QPs in the following.

Notations Let $\mathbb{N} \coloneqq \{0, 1, 2, \dots\}$. For $n \geq 1$, let $[n] \coloneqq \{1, \dots, n\} \subset \mathbb{N}$. We use $\{\!\{\dots\}\!\}$ to denote multisets, i.e., the generalization of sets allowing for multiple instances of each of its elements. A *graph* G is a pair $(V(G), E(G))$ with *finite* sets of *vertices* or *nodes* $V(G)$ and *edges* $E(G) \subseteq \{\{u, v\} \subseteq V(G) \times V(G) \mid u \neq v\}$. For ease of notation, we denote the edge $\{u, v\}$ in $E(G)$ by (u, v) or (v, u). The *neighborhood* of a node v is denoted by $N(v) = \{u \in V \mid (v, u) \in E\}$, and its *degree* is $|N(v)|$. An *attributed graph* augments each node $v \in V$ with a feature vector $\sigma(v) \in \mathbb{R}^d$, yielding a node feature matrix $\boldsymbol{H} \in \mathbb{R}^{n \times d}$ where $\boldsymbol{H}_v = \sigma(v)$. The adjacency matrix of G is denoted $\boldsymbol{A} \in \{0, 1\}^{n \times n}$, with $A_{ij} = 1$ if and only if $(i, j) \in E$. Vectors $\boldsymbol{x} \in \mathbb{R}^d$ are column vectors by default.

LCQP In this work, we focus on special QPs, namely *linearly-constrained quadratic programming* (LCQP), of the following form,

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \tfrac{1}{2} \boldsymbol{x}^\intercal \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{c}^\intercal \boldsymbol{x} \text{ s.t. } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}. \tag{1}$$

Here, an LCQP instance I is a tuple $(\boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$, where $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{c} \in \mathbb{R}^n$ are the quadratic and linear coefficients of the objective, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, and $\boldsymbol{b} \in \mathbb{R}^m$ form the inequality constraints.

Note that the bounds of variables, e.g., $\boldsymbol{x} \geq \mathbf{0}$, can also be merged into the constraints. We assume that the smmetric and quadratic matrix $\boldsymbol{Q}$ is *positive definite* (PD), denoted as $\boldsymbol{Q} \succ \mathbf{0}$, so that the problem is convex and has a unique solution. The optimal solution $\boldsymbol{x}^*$ is a feasible solution such that $\frac{1}{2}\boldsymbol{x}^\intercal \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\intercal \boldsymbol{x} \geq \frac{1}{2}\boldsymbol{x}^{*\intercal} \boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{c}^\intercal \boldsymbol{x}^*$, for any feasible $\boldsymbol{x}$. The corresponding dual variables for the constraints are denoted as $\boldsymbol{\lambda}^*$.[1] By setting the matrix $\boldsymbol{Q}$ to a zero matrix, we arrive at LPs,

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \boldsymbol{c}^\intercal \boldsymbol{x} \text{ s.t. } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}. \tag{2}$$

Considering the QP of Eq. (1), the optimal primal-dual solutions must satisfy the *Karush–Kuhn–Tucker (KKT)* conditions [Nocedal and Wright, 1999, p. 321],

$$\boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{A}^\intercal \boldsymbol{\lambda}^* + \boldsymbol{c} = \mathbf{0} \tag{3a}$$
$$\boldsymbol{A}\boldsymbol{x}^* \leq \boldsymbol{b} \tag{3b}$$
$$\boldsymbol{\lambda}^* \geq \mathbf{0} \tag{3c}$$
$$\lambda_i^* \left(\boldsymbol{A}_i \boldsymbol{x}^* - b_i\right) = 0. \tag{3d}$$

Define slack variables $\boldsymbol{s}^* \coloneqq \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^*$, the inequality constraints become equalities $\boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{b} - \boldsymbol{s}^*$. The KKT conditions can then be compactly written as,

$$\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}^\intercal \\ \boldsymbol{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \begin{bmatrix} -\boldsymbol{c} \\ \boldsymbol{b} - \boldsymbol{s}^* \end{bmatrix}, \tag{4}$$

where the inequality and complementarity conditions Eqs. (3c) and (3d) are implicitly encoded via $\boldsymbol{s}^* \geq \mathbf{0}$ and $s_i^* \lambda_i^* = 0$. In practice, we can partition the inequality constraints into *active* ones $\boldsymbol{A}_a \boldsymbol{x}^* = \boldsymbol{b}_a$ and *inactive* ones $\boldsymbol{A}_{\bar{a}} \boldsymbol{x}^* < \boldsymbol{b}_{\bar{a}}$, where the slack variables satisfy $\boldsymbol{s}_a^* = \mathbf{0}$ and $\boldsymbol{s}_{\bar{a}}^* > \mathbf{0}$.

MPNNs for LCQPs Representing LPs and QPs with MPNNs has been explored in prior work. For example, Chen et al. [2022] models LPs using a bipartite constraint-variable graph, and Chen et al. [2024a] extends this to QPs by adding edges between variable nodes. We adopt the setting of Chen et al. [2024a]. Given an LCQP instance I, we construct a graph $G(I)$ with constraint nodes $C(I)$ and variable nodes $V(I)$. Edges between $C(I)$ and $V(I)$ are defined by nonzero entries of $\boldsymbol{A}$ with weights A_{cv}, for $v \in V(I), c \in C(I)$; and edges between variables are defined by nonzero $Q_{vu}, v, u \in V(I)$. Node features are $\boldsymbol{H}_{\text{c}} \coloneqq \text{reshape}(\boldsymbol{b}) \in \mathbb{R}^{m \times 1}$ for constraint nodes and $\boldsymbol{H}_{\text{v}} \coloneqq \text{reshape}(\boldsymbol{c}) \in \mathbb{R}^{n \times 1}$ for variable nodes. MPNNs learn a vectorial representation of each node in a graph by aggregating information from its neighbors, i.e.,

$$\begin{aligned} \boldsymbol{h}_c^{(t)} &\coloneqq \mathsf{UPD}_{\text{c}}^{(t)}\Big(\boldsymbol{h}_c^{(t-1)}, \sum_{v \in N(c) \cap V(I)} A_{cv} \boldsymbol{h}_v^{(t-1)}\Big) \\ \boldsymbol{h}_v^{(t)} &\coloneqq \mathsf{UPD}_{\text{v}}^{(t)}\Big(\boldsymbol{h}_v^{(t-1)}, \sum_{u \in N(v) \cap V(I)} Q_{uv} \boldsymbol{h}_u^{(t-1)}, \sum_{c \in N(v) \cap C(I)} A_{cv} \boldsymbol{h}_c^{(t)}\Big), \end{aligned} \tag{5}$$

followed by a pooling function and a readout function to predict the objective,

$$\boldsymbol{z}_I \coloneqq \mathsf{POOL}\Big(\sum_{v \in V(I)} \boldsymbol{h}_v^{(T)}, \sum_{c \in C(I)} \boldsymbol{h}_c^{(T)}\Big); \quad \text{obj}(I) \coloneqq \mathsf{READOUT}\big(\boldsymbol{z}_I\big). \tag{6}$$

2 Principled data augmentation for LCQPs

We propose a set of principled transformations for LCQPs as data augmentation. Let $\mathcal{I}$ denote a set of LCQP instances. We consider transformations $T \colon \mathcal{I} \to \mathcal{I}$ that perturb the problem structure while allowing efficient computation of optimal solutions through simple linear-algebraic operations, without requiring a solver. Our core idea is to construct affine transformations of the KKT system Eq. (4) that yield valid KKT conditions for a new problem. We notice that applying a linear mapping $\boldsymbol{M}$ on Eq. (4) does not change the equality, i.e.,

$$\boldsymbol{M} \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}^\intercal \\ \boldsymbol{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \boldsymbol{M} \begin{bmatrix} -\boldsymbol{c} \\ \boldsymbol{b} - \boldsymbol{s}^* \end{bmatrix}$$

[1] We consider only feasible and bounded problems where $\boldsymbol{x}^*$ and $\boldsymbol{\lambda}^*$ are not both zeros. If they are, then $\boldsymbol{c} = \mathbf{0}$ and $\boldsymbol{b} \geq \mathbf{0}$, reducing the problem to an unconstrained QP with $\boldsymbol{Q} \succ \mathbf{0}$ and trivial solution $\boldsymbol{x}^* = \mathbf{0}$, which we exclude.

holds for all choices of transformation matrix $\boldsymbol{M}$. To make the transformed problem valid, we need a $\boldsymbol{M}^\intercal$ on the right and further require a right (pseudo) inverse $(\boldsymbol{M}^\intercal)^\dagger$ with $\boldsymbol{M}^\intercal (\boldsymbol{M}^\intercal)^\dagger = \boldsymbol{I}$. To ensure $(\boldsymbol{M}^\intercal)^\dagger$ exists, the matrix $\boldsymbol{M}$ needs to be full column rank. Now we have

$$\boldsymbol{M} \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}^\intercal \\ \boldsymbol{A} & \boldsymbol{0} \end{bmatrix} \boldsymbol{M}^\intercal (\boldsymbol{M}^\intercal)^\dagger \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \boldsymbol{M} \begin{bmatrix} -\boldsymbol{c} \\ \boldsymbol{b} - \boldsymbol{s}^* \end{bmatrix}.$$

Finally, we could add a bias term $\boldsymbol{B}$ on both sides, and arrive at the general form,

$$\left(\boldsymbol{M} \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}^\intercal \\ \boldsymbol{A} & \boldsymbol{0} \end{bmatrix} \boldsymbol{M}^\intercal + \boldsymbol{B} \right) (\boldsymbol{M}^\intercal)^\dagger \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \boldsymbol{M} \begin{bmatrix} -\boldsymbol{c} \\ \boldsymbol{b} - \boldsymbol{s}^* \end{bmatrix} + \boldsymbol{\beta}. \tag{7}$$

Here, the matrices $\boldsymbol{M}$ and $\boldsymbol{B}$ define the transformation on the problem parameters $(\boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$, with $\boldsymbol{B}$ chosen to satisfy $\boldsymbol{B} (\boldsymbol{M}^\intercal)^\dagger \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \boldsymbol{\beta}$. The quantity $(\boldsymbol{M}^\intercal)^\dagger \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix}$, if it exists, recovers the optimal solution of the transformed problem. Thus, we can compute the new solutions using only matrix operations, without solving the transformed LCQP.

2.1 A framework for valid transformations

An ideal transformation should be expressive, valid, and computationally efficient. Here, *expressivity* refers to its ability to generate a wide range of problem instances from a given one, while *validity* ensures the transformed system remains a valid KKT system. We discuss these properties in detail below.

Expressivity Intuitively, a transformation T is more expressive than another T' if it maps an initial problem to a superset of the targets T' can reach. We observe that the transformation in Eq. (7) has maximal expressivity. For example, setting $\boldsymbol{M} = \boldsymbol{0}$, with proper $\boldsymbol{B}$ and $\boldsymbol{\beta}$ it can recover any target instance $I' = (\boldsymbol{Q}', \boldsymbol{A}', \boldsymbol{b}, \boldsymbol{c}')$ from any source I, regardless of dimensions. This flexibility relies on the bias term, without which the transformations are limited. For instance, a low-rank $\boldsymbol{A}$ cannot map to a higher-rank $\boldsymbol{A}'$. In practice, full expressivity is unnecessary, as our practical objective is not to span the entire space of QP instances, but to generate meaningful, diverse variants for data augmentation.

Validity Transformation following Eq. (7), under some conditions, can ensure that the transformed system remains a valid KKT system, which we explore below. We can write the matrices in block matrix form $\boldsymbol{M} \coloneqq \begin{bmatrix} \boldsymbol{M}_{11} & \boldsymbol{M}_{12} \\ \boldsymbol{M}_{21} & \boldsymbol{M}_{22} \end{bmatrix}$ and $\boldsymbol{B} \coloneqq \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix}$, such that dimensions for $\boldsymbol{A}$ and $\boldsymbol{Q}$ are matched. Using straightforward calculations, we have

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{T}_{11} & \boldsymbol{T}_{12} \\ \boldsymbol{T}_{21} & \boldsymbol{T}_{22} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{M}_{11} & \boldsymbol{M}_{12} \\ \boldsymbol{M}_{21} & \boldsymbol{M}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}^\intercal \\ \boldsymbol{A} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{M}_{11}^\intercal & \boldsymbol{M}_{21}^\intercal \\ \boldsymbol{M}_{12}^\intercal & \boldsymbol{M}_{22}^\intercal \end{bmatrix} + \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22}, \end{bmatrix} \\
\boldsymbol{T}_{11} &= \boldsymbol{M}_{11}\boldsymbol{Q}\boldsymbol{M}_{11}^\intercal + \boldsymbol{M}_{12}\boldsymbol{A}\boldsymbol{M}_{11}^\intercal + \boldsymbol{M}_{11}\boldsymbol{A}^\intercal\boldsymbol{M}_{12}^\intercal + \boldsymbol{B}_{11} \\
\boldsymbol{T}_{12} &= \boldsymbol{M}_{11}\boldsymbol{Q}\boldsymbol{M}_{21}^\intercal + \boldsymbol{M}_{12}\boldsymbol{A}\boldsymbol{M}_{21}^\intercal + \boldsymbol{M}_{11}\boldsymbol{A}^\intercal\boldsymbol{M}_{22}^\intercal + \boldsymbol{B}_{12} \\
\boldsymbol{T}_{21} &= \boldsymbol{M}_{21}\boldsymbol{Q}\boldsymbol{M}_{11}^\intercal + \boldsymbol{M}_{22}\boldsymbol{A}\boldsymbol{M}_{11}^\intercal + \boldsymbol{M}_{21}\boldsymbol{A}^\intercal\boldsymbol{M}_{12}^\intercal + \boldsymbol{B}_{21} \\
\boldsymbol{T}_{22} &= \boldsymbol{M}_{21}\boldsymbol{Q}\boldsymbol{M}_{21}^\intercal + \boldsymbol{M}_{22}\boldsymbol{A}\boldsymbol{M}_{21}^\intercal + \boldsymbol{M}_{21}\boldsymbol{A}^\intercal\boldsymbol{M}_{22} + \boldsymbol{B}_{22}.
\end{aligned}$$

So that the transformed equation Eq. (7) is a valid KKT form $\begin{bmatrix} \boldsymbol{Q}' & \boldsymbol{A}'^\intercal \\ \boldsymbol{A}' & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}'^* \\ \boldsymbol{\lambda}'^* \end{bmatrix} = \begin{bmatrix} -\boldsymbol{c}' \\ \boldsymbol{b}' - \boldsymbol{s}'^* \end{bmatrix}$, it requires $\boldsymbol{T}_{12} = \boldsymbol{T}_{21}^\intercal$, and we further require $\boldsymbol{T}_{11}$ be another positive definite matrix, and $\boldsymbol{T}_{22} = \boldsymbol{0}$ must hold.

For inequality constraints, the transformation matrix $\boldsymbol{M}_{22}$ must satisfy certain conditions to enable efficient computation of the new solution. These conditions are not required for feasibility or optimality but reflect our goal of avoiding QP solving. To isolate $\boldsymbol{M}_{22}$, we simplify the system by discarding the bias terms $\boldsymbol{B}, \boldsymbol{\beta}$, setting $\boldsymbol{M}_{12}, \boldsymbol{M}_{21}$ to zero, and fixing $\boldsymbol{M}_{11} = \boldsymbol{I}$. The resulting transformed KKT system is,

$$\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}^\intercal \boldsymbol{M}_{22}^\intercal \\ \boldsymbol{M}_{22}\boldsymbol{A} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & (\boldsymbol{M}_{22}^\intercal)^\dagger \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M}_{22} \end{bmatrix} \begin{bmatrix} -\boldsymbol{c} \\ \boldsymbol{b} - \boldsymbol{s}^* \end{bmatrix}. \tag{8}$$

This system indicates that the primal solution $\boldsymbol{x}^*$ remains unchanged, while the dual solution is mapped to $(\boldsymbol{M}_{22}^\intercal)^\dagger \boldsymbol{\lambda}^*$. The following proposition establishes the conditions under which this transformation preserves the solution and avoids re-solving the LCQPs; see Appendix D.1 for a proof.

Proposition 2.1. *Let $I \coloneqq (\boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ be a LCQP instance with optimal primal-dual solution $\boldsymbol{x}^*, \boldsymbol{\lambda}^*$. Consider a transformation defined by $T(I) \coloneqq (\boldsymbol{Q}, \boldsymbol{M}_{22}\boldsymbol{A}, \boldsymbol{M}_{22}\boldsymbol{b}, \boldsymbol{c})$. Then the transformed problem preserves the primal solution $\boldsymbol{x}^*$ if and only if $\boldsymbol{M}_{22}$ takes the block form $\begin{bmatrix} \boldsymbol{N}_{11} & \boldsymbol{N}_{12} \\ \boldsymbol{N}_{21} & \boldsymbol{N}_{22} \end{bmatrix}$ to match the dimensions of active and inactive constraints, where $\boldsymbol{N}_{11}$ has full row rank equal to the number of active constraints, $\boldsymbol{N}_{12}\boldsymbol{s}_a^* = \boldsymbol{0}$, $\boldsymbol{N}_{22}\boldsymbol{s}_a^* \geq \boldsymbol{0}$.*

In summary, the conditions on $\boldsymbol{M}, \boldsymbol{B}, \boldsymbol{\beta}$ for the validity of the new problem is as follows,

$$\begin{aligned} \boldsymbol{M}_{11}\boldsymbol{Q}\boldsymbol{M}_{11}^\intercal + \boldsymbol{M}_{12}\boldsymbol{A}\boldsymbol{M}_{11}^\intercal + \boldsymbol{M}_{11}\boldsymbol{A}^\intercal\boldsymbol{M}_{12}^\intercal + \boldsymbol{B}_{11} &\succ \boldsymbol{0} \\ \boldsymbol{M}_{21}\boldsymbol{Q}\boldsymbol{M}_{21}^\intercal + \boldsymbol{M}_{22}\boldsymbol{A}\boldsymbol{M}_{21}^\intercal + \boldsymbol{M}_{21}\boldsymbol{A}^\intercal\boldsymbol{M}_{22} + \boldsymbol{B}_{22} &= \boldsymbol{0} \\ \boldsymbol{B}_{12} &= \boldsymbol{B}_{21}^\intercal \\ \boldsymbol{B}\left(\boldsymbol{M}^\intercal\right)^\dagger \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} &= \boldsymbol{\beta}, \text{ and} \\ \boldsymbol{M}_{22} &\text{ satisfies Proposition 2.1.} \end{aligned} \tag{9}$$

Computational efficiency In general, satisfying the full transformation structure in Eq. (9) is challenging, particularly when arbitrary $\boldsymbol{M}, \boldsymbol{B}$ and $\boldsymbol{\beta}$ are involved. However, we notice the multiplicative term $\boldsymbol{M}$ and bias terms $\boldsymbol{B}, \boldsymbol{\beta}$ can be decoupled. That is, we can design transformations with the bias terms $\boldsymbol{B}, \boldsymbol{\beta}$, e.g. Appendix C.2, but we mainly focus on dropping them, setting $\boldsymbol{M}_{12}$ and $\boldsymbol{M}_{21}$ to zero matrices, and investigate the design space of $\boldsymbol{M}_{11}$ and $\boldsymbol{M}_{22}$. We will abbreviate the subscripts of $\boldsymbol{M}_{ii}$, and we target at designing transformations of the form

$$\begin{bmatrix} \boldsymbol{M}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}^\intercal \\ \boldsymbol{A} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{M}_1^\intercal & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M}_2^\intercal \end{bmatrix} \begin{bmatrix} (\boldsymbol{M}_1^\intercal)^\dagger & \boldsymbol{0} \\ \boldsymbol{0} & (\boldsymbol{M}_2^\intercal)^\dagger \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \begin{bmatrix} \boldsymbol{M}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M}_2 \end{bmatrix} \begin{bmatrix} -\boldsymbol{c} \\ \boldsymbol{b} - \boldsymbol{s}^* \end{bmatrix}. \tag{10}$$

We notice the commutativity and the decoupled nature,

$$\begin{bmatrix} \boldsymbol{M}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{M}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M}_2 \end{bmatrix},$$

which enables us to design $\boldsymbol{M}_1, \boldsymbol{M}_2$ separately, and merge them later.

The design space in Eq. (10) is flexible but constrained by the need to compute pseudo-inverses. For $\boldsymbol{M}_1 \in \mathbb{R}^{n' \times n}$, there are three cases: (i) $n' = n$ linear reparameterization, (ii) $n' < n$, dropping variables, and (iii) $n' > n$ adding variables. While (ii) is often ill-posed since $\boldsymbol{M}_1^\intercal$ does not have full column rank and lacks a right inverse, some special cases are still feasible, as we will show. More broadly, computing $(\boldsymbol{M}_1^\intercal)^\dagger = \boldsymbol{M}_1 \left(\boldsymbol{M}_1^\intercal \boldsymbol{M}_1\right)^{-1}$ requires full column rank and typically costs $\mathcal{O}(n^3)$. To improve efficiency, we focus on structured matrices, for example, diagonal matrices, as a particular case, enable both efficient inverse calculation and generation in $\mathcal{O}(n)$ time. These considerations motivate the use of diagonal or structured $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ for scalable data augmentation. To formalize efficiency, we introduce two notions of efficiently computable transformations. The first covers transformations whose solutions can be computed in linear time.

Definition 2.2 (Efficiently recoverable transformation)**.** *For an LCQP instance $I \coloneqq (\boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}) \in \mathcal{I}$ and its primal-dual solutions $\boldsymbol{x}^*, \boldsymbol{\lambda}^*$, a transform $T \in \mathcal{T}$ is* efficiently recoverable*, if the new solutions of the transformed instance $\mathcal{I}' = T(\mathcal{I})$ can be obtained within linear time $\mathcal{O}(n)$.*

The second one, motivated by unsupervised settings, tightens this by requiring that the transformation be independent of the original solution and focuses on generating structurally consistent instances rather than solving them.

Definition 2.3 (Solution-independent transformation)**.** *Given a LCQP instance $I \coloneqq (\boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}) \in \mathcal{I}$, a transformation $T \in \mathcal{T}$ is* solution-independent *if the solution of the transformed problem can be obtained without a solver, and the transformation parameters do not depend on the original optimal solutions $\boldsymbol{x}^*, \boldsymbol{\lambda}^*$.*

2.2 Example transformations

In the following, we discuss various transformations that fit into the above framework.

Removing idle variables As discussed above, when $\boldsymbol{M}_1$ does not have full column rank, the pseudo inverse $(\boldsymbol{M}_1^\intercal)^\dagger$ typically does not exist, making such transformations ill-posed. However, there is a special case in which we can safely remove a variable, specifically, when it is idle because its optimal value is zero, resulting in the following proposition.

Proposition 2.4. *Let $I \coloneqq (\boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ be an LCQP instance with primal-dual solution $\boldsymbol{x}^*, \boldsymbol{\lambda}^*$. Then a variable x' can be removed from the problem without affecting the optimal values of the remaining variables if and only if $x'^* = 0$.*

Removing inactive constraints Like the variable removal case above, a constraint can be removed under certain conditions by introducing a wide identity matrix $\boldsymbol{M}_2$ that selectively excludes the corresponding row.

Proposition 2.5. *Let $I \coloneqq (\boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ be an LCQP instance with optimal primal-dual solution $\boldsymbol{x}^*, \boldsymbol{\lambda}^*$. Then a constraint of the form $\boldsymbol{a}'^\intercal \boldsymbol{x}^* \leq b'$ can be removed from the problem without affecting the optimal solution if and only if it is strictly inactive, i.e., $\boldsymbol{a}'^\intercal \boldsymbol{x}^* < b'$.*

The variable and constraint removal transformations are *efficiently recoverable*, as the remaining solutions are unchanged and need no recomputation. However, they are not *solution-independent*, since identifying removable components requires access to the primal solution. Moreover, we apply a heuristic on problem instances to select inactive constraints, as described in Appendix C.3.

Scaling variable coefficients A natural class of transformations involves scaling the coefficients associated with individual variables. Specifically, scaling the j-th column of $\boldsymbol{A}$ and the j-th entry of $\boldsymbol{c}$ by a nonzero scalar α_j, while updating $\boldsymbol{Q}_{ij}$ by $\alpha_i \alpha_j$, i.e., $T(I) \coloneqq (\boldsymbol{M}_1 \boldsymbol{Q} \boldsymbol{M}_1, \boldsymbol{A}\boldsymbol{M}_1^\intercal, \boldsymbol{b}, \boldsymbol{M}_1 \boldsymbol{c})$ with a diagonal $\boldsymbol{M}_1$ preserves the structure of the QP. Under this transformation, the optimal value remains unchanged, and the solution x_j^* is rescaled by $1/\alpha_j$. This transformation is *efficiently recoverable*, as the new solution can be obtained directly from the original in $\mathcal{O}(n)$ time. Moreover, it is also *solution-independent*, as it does not require access to the original solution, but only knowledge of how to compute the new one.

Adding variables When $n' > n$, the transformation effectively adds new variables to the problem and linearly combines existing ones. Without loss of generality, we consider adding a single new variable by choosing a transformation matrix of the form $\boldsymbol{M}_1 \coloneqq \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{q}^\intercal \end{bmatrix}$, with $\boldsymbol{q} \in \mathbb{R}^n$ being an arbitrary vector. This yields a new positive definite quadratic matrix $\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{Q}\boldsymbol{q} \\ \boldsymbol{q}^\intercal \boldsymbol{Q} & \boldsymbol{q}^\intercal \boldsymbol{Q} \boldsymbol{q} \end{bmatrix}$. We can find the pseudo inverse $(\boldsymbol{M}_1^\intercal)^\dagger \coloneqq \boldsymbol{M}_1 \left(\boldsymbol{M}_1^\intercal \boldsymbol{M}_1\right)^{-1} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{q}^\intercal \end{bmatrix} \left(\boldsymbol{I} + \boldsymbol{q}\boldsymbol{q}^\intercal\right)^{-1}$, which can be computed with Sherman-Morrison formula [Shermen and Morrison, 1949]. In this case, the primal solution of the original variables does not remain the same. Interestingly, due to the structure of $\boldsymbol{M}_1$, another valid pseudo-inverse is $(\boldsymbol{M}_1^\intercal)^\dagger = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{0}^\intercal \end{bmatrix}$, which is a special case indicating that the added variable has zero contribution to the solution. This corresponds to the reverse of the variable removal transformation discussed above.

Proposition 2.6. *Let $I = (\boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ be an LCQP instance with optimal solution $\boldsymbol{x}^*, \boldsymbol{\lambda}^*$. Define the transformation $T(I) \coloneqq (\boldsymbol{M}_1 \boldsymbol{Q} \boldsymbol{M}_1, \boldsymbol{A}\boldsymbol{M}_1^\intercal, \boldsymbol{b}, \boldsymbol{M}_1 \boldsymbol{c})$, where $\boldsymbol{M}_1 \coloneqq \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{q}^\intercal \end{bmatrix}$. Then the transformed problem has optimal primal solution $(\boldsymbol{x}^*, x'^*)$ if and only if the new variable $x'^* = 0$.*

This transformation is both *efficiently recoverable* and *solution-independent*. Moreover, there also exist other implementations of variable addition in the form of Eq. (7), with a bias term. Please refer to Appendix C.4.

Scaling constraints Similar to the variable scaling transformation above, we can scale the constraint coefficients by fixing $\boldsymbol{M}_1 = \boldsymbol{I}$ and letting $\boldsymbol{M}_2$ be a square diagonal matrix. We assume all diagonal

entries of $\boldsymbol{M}_2$ are positive. If any entry is zero, the transformation reduces to inactive constraint removal above, or an active constraint is removed, and the problem will be relaxed. If any entry is negative, the corresponding inequality direction will be flipped, and the problem's solution may change. Specifically, the transformation would be $T(I) \coloneqq (\boldsymbol{Q}, \boldsymbol{M}_2\boldsymbol{A}, \boldsymbol{M}_2\boldsymbol{b}, \boldsymbol{c})$.

Under this transformation, the new dual variables are given by $\boldsymbol{\lambda}'^{*} \coloneqq \boldsymbol{M}_2^{-1}\boldsymbol{\lambda}^{*}$, which can be computed in linear time since $\boldsymbol{M}_2$ is a diagonal matrix. The primal solution remains unchanged. This transformation is both *efficiently recoverable* and *solution-independent*.

In Appendix C, we outline additional transformations, including constraint addition Appendix C.1.

2.3 Using data augmentations

Having introduced efficient data augmentation methods for LCQPs and LPs, we now describe how they can be integrated into different training pipelines. Our target task is graph regression to predict the objective value from a graph representation. Depending on the setting, we can either (1) use solution information to generate supervised labels for related problems or (2) apply solution-independent augmentations for contrastive pretraining without solutions.

Supervised learning Given a training set and a set of augmentation methods, we dynamically generate additional training instances during optimization. We randomly apply a selected augmentation or a combination of augmentations at each iteration, and train the MPNN using the supervised loss on the predicted objective value.

Contrastive pretraining We perform self-supervised contrastive learning on the entire dataset without touching the solutions of the problems, using the NT-Xent loss [Chen et al., 2020]. Specifically, for a mini-batch of N data instances, we generate two augmented views of each instance using solution-independent transformations, resulting in $2N$ data instances. We consider the two views of the same instance (i, j) as a positive pair, and the other $2(N-1)$ samples as negative. The similarity between embeddings is measured by 2-norm-normalized cosine similarity $\mathsf{sim}(\boldsymbol{u}, \boldsymbol{v}) \coloneqq \frac{\boldsymbol{u}^\intercal \boldsymbol{v}}{\|\boldsymbol{u}\|\|\boldsymbol{v}\|}$. For each instance i and its positive sample j, we have the NT-Xent loss on the pooled representations $\boldsymbol{z}_i, \boldsymbol{z}_j$ from Eq. (6) as

$$-\frac{1}{N}\sum_{i=1}^{N}\log\frac{\exp\left(\mathsf{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau\right)}{\sum_{k=1}^{2N}\mathbb{I}(k \neq i)\exp\left(\mathsf{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau\right)}, \tag{11}$$

where $\tau > 0$ is a temperature hyperparameter.

3 Experimental setup and results

To empirically validate the effectiveness of our data augmentations, we conduct a series of experiments, answering the following research questions.[2]

Q1 Do our augmentations improve supervised learning, especially under data scarcity?
Q2 Are the augmentations effective in contrastive pretraining followed by supervised finetuning?
Q3 Does the pretrained model enhance generalization on out-of-distribution (OOD) or larger datasets?
Q4 What is the practical computational overhead of the augmentations?

As LPs are a special case of QPs, we evaluate them separately. We generate $10\,000$ instances for each dataset, each with 100 variables and 100 inequality constraints, and split the data into training, validation, and test sets with $8:1:1$ ratio. To study performance under data scarcity, we partition the training set into multiple disjoint subsets in each run, each containing either 10% or 20% of the full training data. Models are trained independently on each subset, and results are aggregated across all partitions. Hyperparameters are tuned using supervised training on the full training set and fixed across all methods. Specifically, we use a 6-layer MPNN followed by a 3-layer MLP with 192 hidden dimensions and GraphNorm [Cai et al., 2021]. All experiments are conducted on a single NVIDIA L40S GPU. We evaluate performance using the mean relative objective error (in percentage) over the test set, defined as $1/|\mathcal{D}| \sum_{I\in\mathcal{D}} |(\text{obj}(I) - \text{obj}^*(I))/\text{obj}^*(I)| \cdot 100\%$, where $\mathcal{D}$ denotes the set of

[2]The repository of our source code can be accessed at `https://github.com/chendiqian/Data-Augmentation-for-Learning-to-Optimize`.

instances, $\text{obj}^*(I)$ is the optimal objective value, and $\text{obj}(I)$ is the predicted objective value. We repeat the experiments five times with different seeds and report the mean and standard deviation.

Supervised learning To address **Q1**, we investigate the impact of data augmentation on LPs and QPs in a supervised learning setting. We evaluate performance on synthetically generated datasets following the procedure described in Appendix G. Models are trained with a batch size of 32 for up to 2000 epochs, with early stopping after 200 epochs of patience. We evaluate performance under data scarcity by training models on subsets containing 10%, 20%, and 100% of the training data. As baselines, we apply the augmentations proposed by You et al. [2020]: node dropping, edge perturbation, and feature masking. In addition, we evaluate each of our proposed augmentations separately and in combination. Hyperparameter details for all augmentations are provided in Appendix F. Results, shown in Table 1, reveal that the augmentations from You et al. [2020] fail to improve performance consistently and can even be detrimental compared to training without augmentation. In contrast, all of our proposed methods consistently yield better performance, and combining augmentations further amplifies the improvement up to 62.6% on LP with 20% of training data, aligning with empirical evidence observed in You et al. [2020].

Table 1: Supervised learning performance on LP/QP datasets with and without data augmentation under different levels of data scarcity. The best-performing method is colored in green, the second-best in blue, and third in orange. Our proposed augmentations consistently improve performance.

	LP			QP		
Augmentation	10%	20%	100%	10%	20%	100%
None	8.539±0.206	6.149±0.153	2.784±0.081	5.304±0.229	3.567±0.034	1.240±0.088
Drop node	8.618±0.152	7.131±0.072	4.654±0.047	6.355±0.254	4.867±0.126	2.594±0.087
Mask node	8.979±0.153	7.591±0.117	3.216±0.149	5.865±0.414	4.005±0.161	1.347±0.067
Flip edge	7.794±0.115	6.503±0.125	4.358±0.081	5.485±0.133	4.361±0.138	2.561±0.091
Drop vars.	4.996±0.094	3.484±0.014	1.484±0.070	4.232±0.195	2.374±0.085	0.835±0.042
Drop cons.	5.563±0.097	3.691±0.087	1.656±0.057	3.407±0.099	2.104±0.073	0.821±0.063
Scale cons.	5.491±0.093	3.872±0.136	1.612±0.121	3.909±0.142	2.448±0.033	0.722±0.058
Scale vars.	4.682±0.162	3.208±0.116	1.241±0.052	3.790±0.013	2.468±0.124	0.649±0.040
Add cons.	6.245±0.118	4.299±0.048	1.878±0.059	3.731±0.065	2.304±0.042	0.814±0.052
Add vars.	6.565±0.118	4.696±0.101	2.188±0.102	4.475±0.276	2.885±0.045	1.021±0.055
Combo	3.465±0.045	2.300±0.073	1.051±0.021	2.434±0.061	1.532±0.033	0.542±0.013

Contrastive pretraining To address **Q2**, we evaluate whether contrastive pretraining can improve supervised fine-tuning, potentially under data scarcity. We adopt a semi-supervised setting: a small subset ($10\%, 20\%, 100\%$) of the training data is labeled, while the whole training set is available as unlabeled data. During pretraining, we use the complete unlabeled training set and train only an MPNN backbone without a prediction head, following the deployment described in Section 2.3. We pretrain for 800 epochs with a batch size of 128, and set $\tau = 0.1$. To assess pretraining quality and pick the best set of hyperparameters, we use linear probing [Veličković et al., 2018], training only a linear regression layer on top of a frozen MPNN to efficiently evaluate feature quality. For finetuning, we follow Zeng and Xie [2021], attaching an MLP head and jointly training it with the MPNN using supervised regression loss, essentially the same setup as supervised learning, but initialized from a pretrained model.

As baselines, we consider several graph contrastive learning methods. GraphCL [You et al., 2020] generates views via node dropping, edge perturbation, and feature masking. GCC [Qiu et al., 2020] samples random walk subgraphs. IGSD [Zhang et al., 2023] uses graph diffusion [Gasteiger et al., 2019] combined with model distillation. MVGRL [Hassani and Khasahmadi, 2020] also employs diffusion-based views but performs contrastive learning at the graph-node level. Additionally, we include mutual information maximization methods such as InfoGraph [Sun et al., 2019] and DGI [Veličković et al., 2018]. Beyond contrastive methods, we evaluate the generative SSL method GAE [Kipf and Welling, 2016], which reconstructs graph edge weights.

As shown in Table 2, GraphCL and GCC pretraining can improve performance in some cases but do not consistently yield better results. Other baselines even degrade performance. In contrast, our pretraining methods substantially improve, reducing the objective gap by 59.4% on LP and 54.1% on QP with only 10% of the training data. This highlights the effectiveness of our data augmentations, which are specifically tailored for optimization problem instances and significantly enhance fine-tuning.

Table 2: Pretrained-finetuned model performance on LP/QP datasets under different levels of data scarcity. Our pretraining consistently improves performance and outperforms the baselines.

	LP			QP		
Pretraining	10%	20%	100%	10%	20%	100%
None	8.539±0.206	6.149±0.153	2.784±0.081	5.304±0.229	3.567±0.034	1.240±0.088
GraphCL	9.694±1.656	6.033±0.653	2.613±0.266	6.024±0.859	3.865±0.296	1.253±0.114
GCC	8.248±0.916	5.761±0.186	2.686±0.201	11.344±0.198	6.356±0.964	1.472±0.076
IGSD	18.097±2.276	7.631±0.794	3.082±0.269	12.799±1.485	6.306±0.628	1.302±0.094
MVGRL	20.392±2.476	9.072±1.854	2.852±0.112	8.440±0.981	4.932±1.083	1.343±0.063
InfoGraph	18.338±3.285	7.464±0.818	2.956±0.179	10.306±0.189	6.246±0.180	1.409±0.132
DGI	19.661±4.468	9.671±2.764	3.156±0.188	10.014±0.528	7.223±0.142	1.425±0.109
GAE	9.082±0.901	6.032±0.241	3.434±0.255	5.848±0.181	3.759±0.158	1.381±0.027
Ours	3.472±0.086	2.794±0.049	1.588±0.056	3.791±0.097	2.427±0.083	0.926±0.031

Generalization To address **Q3**, we compare the performance of models trained from scratch versus models initialized with contrastive pretraining and then finetuned. For LPs, we generate four types of relaxed LP instances derived from MILPs: Set Cover (SC), Maximum Independent Set (MIS), Combinatorial Auction (CA), and Capacitated Facility Location (CFL), following Gasse et al. [2019]. For QPs, we generate instances of soft-margin SVM, Markowitz portfolio optimization, and LASSO regression following Jung et al. [2022]. If possible, problem sizes and densities are kept similar to the pretraining datasets; see more details in Appendix G. As shown in Tables 3 and 4, pretrained models outperform models trained from scratch in almost all cases, demonstrating strong transferability to OOD tasks. The evaluation on larger datasets can be found in Appendix E.2.

Table 3: Generalization performance of contrastive pretrained MPNNs on OOD LP instances.

		Ratio		
Family	**Pretrained**	10%	20%	100%
SC	No	5.297±0.787	2.531±0.861	0.752±0.042
	CL	1.965±0.206	1.349±0.161	0.662±0.055
MIS	No	0.674±0.016	0.465±0.022	0.203±0.006
	Yes	0.722±0.056	0.421±0.047	0.177±0.020
CA	No	2.381±0.048	1.803±0.027	0.906±0.025
	Yes	2.303±0.184	1.713±0.138	0.753±0.055
CFL	No	0.401±0.011	0.255±0.024	0.059±0.006
	CL	**0.367±0.047**	0.217±0.018	0.064±0.008

Table 4: Generalization performance of contrastive pretrained MPNNs on OOD QP instances.

		Ratio		
Family	**Pretrained**	10%	20%	100%
SVM	No	0.191±0.006	0.109±0.007	0.027±0.004
	Yes	0.141±0.017	0.077±0.009	0.024±0.004
Portfolio	No	3.766±0.198	1.841±0.162	0.402±0.005
	Yes	3.331±0.376	1.637±0.013	0.353±0.014
LASSO	No	5.405±0.018	4.083±0.025	2.159±0.493
	Yes	5.169±0.171	3.726±0.267	1.178±0.097

Regarding **Q4**, see Appendix E.1 for results and discussion.

4 Conclusion

We introduced a principled framework for data augmentation in learning to optimize over linear and quadratic programming. By leveraging affine transformations of the KKT system, we designed a family of expressive, solution-preserving, and computationally efficient transformations. Our method allows for augmentations that either admit exact solution recovery or preserve key structural properties without requiring access to the original solutions, making them suitable for supervised and contrastive learning. Extensive experiments show that these augmentations consistently improve performance under data scarcity, generalize to larger and out-of-distribution problems, and outperform existing graph augmentation baselines. This work highlights the benefits of optimization-aware augmentation strategies and opens new directions for robust, scalable L2O under limited supervision.

Acknowledgements

Christopher Morris and Chendi Qian are partially funded by a DFG Emmy Noether grant (468502433) and RWTH Junior Principal Investigator Fellowship under Germany's Excellence Strategy. We thank Erik Müller for crafting the figures.

References

T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1): 42–54, 2005. 1

A. Arnaiz-Rodríguez, A. Begga, F. Escolano, and N. Oliver. Diffwire: Inductive graph rewiring via the lovasz bound. *arXiv preprint*, 2022. 2

F. Barbero, A. Velingker, A. Saberi, M. Bronstein, and F. Di Giovanni. Locality-aware graph-rewiring in gnns. *arXiv preprint*, 2023. 2

Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. 1

P. Bielak, T. Kajdanowicz, and N. V. Chawla. Graph barlow twins: A self-supervised representation learning framework for graphs. *Knowledge-Based Systems*, 256:109631, 2022. 2, 3

S. Bowly. *Stress testing mixed integer programming solvers through new test instance generation methods.* PhD thesis, University of Melbourne, Parkville, Victoria, Australia, 2019. 3

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. 1, 26

X. Bresson and T. Laurent. Residual gated graph convnets. *arXiv preprint*, 2017. 2

T. Cai, S. Luo, K. Xu, D. He, T.-y. Liu, and L. Wang. Graphnorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning*, pages 1204–1215. PMLR, 2021. 8

Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023. 1

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. 8

Z. Chen, J. Liu, X. Wang, J. Lu, and W. Yin. On representing linear programs by graph neural networks. *arXiv preprint*, 2022. 2, 4

Z. Chen, J. Liu, X. Wang, J. Lu, and W. Yin. On representing mixed-integer linear programs by graph neural networks. *arXiv preprint*, 2023. 2

Z. Chen, X. Chen, J. Liu, X. Wang, and W. Yin. Expressive power of graph neural networks for (mixed-integer) quadratic programs. *arXiv preprint*, 2024a. 2, 4

Z. Chen, J. Liu, X. Chen, W. Wang, and W. Yin. Rethinking the capacity of graph neural networks for branching strategy. *Advances in Neural Information Processing Systems*, 37:123991–124024, 2024b. 2

J.-Y. Ding, C. Zhang, L. Shen, S. Li, B. Wang, Y. Xu, and L. Song. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1452–1459, 2020. 2

K. Ding, Z. Xu, H. Tong, and H. Liu. Data augmentation for deep graph learning: A survey. *ACM SIGKDD Explorations Newsletter*, 24(2):61–77, 2022. 2

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. 3

D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *arXiv preprint*, 2015. 2

B. Fatemi, S. Abu-El-Haija, A. Tsitsulin, M. Kazemi, D. Zelle, N. Bulut, J. Halcrow, and B. Perozzi. Ugsl: A unified framework for benchmarking graph structure learning. *arXiv preprint*, 2023. 2

F. Furini, E. Traversi, P. Belotti, A. Frangioni, A. Gleixner, N. Gould, L. Liberti, A. Lodi, R. Misener, H. Mittelmann, et al. Qplib: a library of quadratic programming instances. *Mathematical Programming Computation*, 11:237–265, 2019. 23, 29

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: 15580–15592, 2019. 1, 2, 3, 10, 31

M. Gasse, S. Bowly, Q. Cappart, J. Charfreitag, L. Charlin, D. Chételat, A. Chmiela, J. Dumouchelle, A. Gleixner, A. 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. 3

J. Gasteiger, S. Weißenberger, and S. Günnemann. Diffusion improves graph learning. *Advances in Neural Information Processing Systems*, 32:13366–13378, 2019. 9

Z. Geng, X. Li, J. Wang, X. Li, Y. Zhang, and F. Wu. A deep instance generative framework for milp solvers under limited data availability. *Advances in Neural Information Processing Systems*, 36: 26025–26047, 2023. 3

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017. 1, 2

A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. Christophel, K. Jarck, T. Koch, J. Linderoth, et al. Miplib 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021. 23

Z. Guo, Y. Li, C. Liu, W. Ouyang, and J. Yan. Acm-milp: Adaptive constraint modification via grouping and selection for hardness-preserving milp instance generation. In *International Conference on Machine Learning*, pages 16869–16890. PMLR, 2024. 3

B. Gutteridge, X. Dong, M. M. Bronstein, and F. Di Giovanni. Drew: Dynamically rewired message passing with delay. In *International Conference on Machine Learning*, pages 12252–12267. PMLR, 2023. 2

W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30:1025–1035, 2017. 2

K. Hassani and A. H. Khasahmadi. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*, pages 4116–4126. PMLR, 2020. 2, 3, 9, 31

Y. Hu, S. Ouyang, J. Liu, G. Chen, Z. Yang, J. Wan, F. Zhang, Z. Wang, and Y. Liu. Graph ranking contrastive learning: A extremely simple yet efficient method. *arXiv preprint*, 2023. 2

T. Huang, A. M. Ferber, Y. Tian, B. Dilkina, and B. Steiner. Searching large neighborhoods for integer linear programs with contrastive learning. In *International Conference on Machine Learning*, pages 13869–13890. PMLR, 2023. 3

C. Ji, Z. Huang, Q. Sun, H. Peng, X. Fu, Q. Li, and J. Li. Regcl: rethinking message passing in graph contrastive learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 8544–8552, 2024. 3

W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang. Graph structure learning for robust graph neural networks. *arXiv preprint*, 2020. 2

H. Jung, J. Park, and J. Park. Learning context-aware adaptive solvers to accelerate quadratic programming. *arXiv preprint*, 2022. 10

K. Karhadkar, P. K. Banerjee, and G. Montúfar. FoSR: First-order spectral rewiring for addressing oversquashing in gnns. *arXiv preprint*, 2022. 2

E. B. Khalil, C. Morris, and A. Lodi. Mip-gnn: A data-driven framework for guiding combinatorial solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10219–10227, 2022. 2

T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint*, 2016. 9, 31

B. Li, L. Yang, Y. Chen, S. Wang, Q. Chen, H. Mao, Y. Ma, A. Wang, T. Ding, J. Tang, et al. PDHG-unrolled learning-to-optimize method for large-scale linear programming. *arXiv preprint*, 2024a. 2

Q. Li, T. Ding, L. Yang, M. Ouyang, Q. Shi, and R. Sun. On the power of small-size graph neural networks for linear programming. In *Advances in Neural Information Processing Systems*, volume 37, pages 38695–38719, 2024b. 2

Q. Li, M. Ouyang, T. Ding, Y. Wang, Q. Shi, and R. Sun. Towards explaining the power of constant-depth graph neural networks for structured linear programming. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=INow59Vurm`. 2

S. Li, J. Kulkarni, I. Menache, C. Wu, and B. Li. Towards foundation models for mixed integer linear programming. *arXiv preprint*, 2024c. 3

L. Lin, J. Chen, and H. Wang. Spectral augmentation for self-supervised learning on graphs. *arXiv preprint*, 2022. 2

H. Liu, J. Wang, W. Zhang, Z. Geng, Y. Kuang, X. Li, B. Li, Y. Zhang, and F. Wu. Milp-studio: Milp instance generation via block structure decomposition. *arXiv preprint*, 2024. 3

N. Liu, X. Wang, D. Bo, C. Shi, and J. Pei. Revisiting graph contrastive learning from the perspective of graph spectrum. *Advances in Neural Information Processing Systems*, 35:2972–2983, 2022a. 2

Q. Liu, L. Chen, F. Fang, and X. Qian. Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering*, 33(8):3105–3118, 2021. 1

S. Liu, R. Ying, H. Dong, L. Li, T. Xu, Y. Rong, P. Zhao, J. Huang, and D. Wu. Local augmentation for graph neural networks. In *International Conference on Machine Learning*, pages 14054–14072. PMLR, 2022b. 2

Y. Liu, Y. Zheng, D. Zhang, H. Chen, H. Peng, and S. Pan. Towards unsupervised deep graph structure learning. In *ACM Web Conference 2022*, 2022c. 2

J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 1999. 1, 4

P. A. Papp, K. Martinkus, L. Faber, and R. Wattenhofer. Dropgnn: Random dropouts increase the expressiveness of graph neural networks. *Advances in Neural Information Processing Systems*, 34: 21997–22009, 2021. 2

C. Qian and C. Morris. Towards graph neural networks for provably solving convex optimization problems. *arXiv preprint*, 2025. 2

C. Qian, A. Manolache, K. Ahmed, Z. Zeng, G. V. d. Broeck, M. Niepert, and C. Morris. Probabilistically rewired message-passing neural networks. *arXiv preprint*, 2023. 2

C. Qian, D. Chételat, and C. Morris. Exploring the power of graph neural networks in solving linear optimization problems. In *International Conference on Artificial Intelligence and Statistics*, pages 1432–1440. PMLR, 2024a. 2

C. Qian, A. Manolache, C. Morris, and M. Niepert. Probabilistic graph rewiring via virtual nodes. *arXiv preprint*, 2024b. 2

J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1150–1160, 2020. 2, 3, 9, 31

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021. 3

Y. Rong, W. Huang, T. Xu, and J. Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint*, 2019. 2

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008. 1, 2

J. Shermen and W. Morrison. Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix. *Annual Mathmatical Statistics*, 20:621–625, 1949. 7

F.-Y. Sun, J. Hoffmann, V. Verma, and J. Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint*, 2019. 3, 9, 31

S. Suresh, P. Li, C. Hao, and J. Neville. Adversarial graph augmentation to improve graph contrastive learning. *Advances in Neural Information Processing Systems*, 34:15920–15933, 2021. 2, 3

S. Thakoor, C. Tallec, M. G. Azar, M. Azabou, E. L. Dyer, R. Munos, P. Veličković, and M. Valko. Large-scale representation learning on graphs via bootstrapping. *arXiv preprint*, 2021. 2, 3

J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint*, 2021. 2

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint*, 2017. 2

P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm. Deep graph infomax. *arXiv preprint*, 2018. 3, 9, 31

G. Wan, Y. Tian, W. Huang, N. V. Chawla, and M. Ye. S3gcl: Spectral, swift, spatial graph contrastive learning. In *International Conference on Machine Learning*, pages 49973–49990. PMLR, 2024. 2, 3

H. Wang, J. Liu, X. Chen, X. Wang, P. Li, and W. Yin. Dig-milp: a deep instance generator for mixed-integer linear programming with feasibility guarantee. *arXiv preprint*, 2023. 3

C. Wu, C. Wang, J. Xu, Z. Liu, K. Zheng, X. Wang, Y. Song, and K. Gai. Graph contrastive learning with generative adversarial network. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2721–2730, 2023. 2

C. Wu, Q. Chen, A. Wang, T. Ding, R. Sun, W. Yang, and Q. Shi. On representing convex quadratically constrained quadratic programs via graph neural networks. *arXiv preprint*, 2024. 2, 29

L. Wu, H. Lin, C. Tan, Z. Gao, and S. Z. Li. Self-supervised learning on graphs: Contrastive, generative, or predictive. *IEEE Transactions on Knowledge and Data Engineering*, 2021. 2

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *arXiv preprint*, 2018. 2

K. Yang, H. Han, W. Jin, and H. Liu. Spectral-aware augmentation for enhanced graph representation learning. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 2837–2847, 2024a. 2

L. Yang, B. Li, T. Ding, J. Wu, A. Wang, Y. Wang, J. Tang, R. Sun, and X. Luo. An efficient unsupervised framework for convex quadratic programs via deep unrolling. *arXiv preprint arXiv:2412.01051*, 2024b. 29

T. Yang, H. Ye, and H. Xu. Learning to generate scalable milp instances. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 159–162, 2024c. 3

Z. Yang, M. Ding, C. Zhou, H. Yang, J. Zhou, and J. Tang. Understanding negative sampling in graph representation learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1666–1676, 2020. 2

Y. Yin, Q. Wang, S. Huang, H. Xiong, and X. Zhang. Autogcl: Automated graph contrastive learning via learnable view generators. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8892–8900, 2022. 2, 3

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. 2, 3, 9, 30, 31

Y. You, T. Chen, Y. Shen, and Z. Wang. Graph contrastive learning automated. In *International Conference on Machine Learning*, pages 12121–12132. PMLR, 2021. 2

J. Zeng and P. Xie. Contrastive self-supervised learning for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10824–10832, 2021. 3, 9

S. Zeng, M. Zhou, S. Zhang, Y. Hu, F. Wu, and X.-Y. Li. Clcr: Contrastive learning-based constraint reordering for efficient milp solving. *arXiv preprint*, 2025. 3

H. Zhang, S. Lin, W. Liu, P. Zhou, J. Tang, X. Liang, and E. P. Xing. Iterative graph self-distillation. *IEEE Transactions on Knowledge and Data Engineering*, 36(3):1161–1169, 2023. 9, 31

Y. Zhang, C. Fan, D. Chen, C. Li, W. Ouyang, M. Zhu, and J. Yan. MILP-FBGen: LP/MILP instance generation with Feasibility/Boundedness. In *International Conference on Machine Learning*, volume 235, pages 58881–58896. PMLR, 2024. 3

T. Zhao, Y. Liu, L. Neves, O. Woodford, M. Jiang, and N. Shah. Data augmentation for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11015–11023, 2021. 2

T. Zhao, W. Jin, Y. Liu, Y. Wang, G. Liu, S. Günnemann, N. Shah, and M. Jiang. Graph data augmentation for graph machine learning: A survey. *arXiv preprint*, 2022. 2

Z. Zhou, S. Zhou, B. Mao, X. Zhou, J. Chen, Q. Tan, D. Zha, Y. Feng, C. Chen, and C. Wang. Opengsl: A comprehensive benchmark for graph structure learning. *Advances in Neural Information Processing Systems*, 36:17904–17928, 2023. 2

Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang. Deep graph contrastive representation learning. *arXiv preprint*, 2020. 2, 3

Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, pages 2069–2080, 2021. 2, 3

D. Zou, H. Peng, X. Huang, R. Yang, J. Li, J. Wu, C. Liu, and P. S. Yu. SE-GSL: A general and effective graph structure learning framework through structural entropy optimization. *arXiv preprint*, 2023. 2

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state our main contributions, which are supported by both theoretical analysis and empirical results, and are appropriately scoped to reflect the demonstrated capabilities of our method.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We explicitly discuss the limitations of our approach in Appendix A, including dataset scale, generalizability to real-world benchmarks, and modeling assumptions.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: All theoretical results are accompanied by clearly stated assumptions and complete proofs, which are provided in Appendix D.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide all necessary details to reproduce our main results in Section 3, Appendix F, and Appendix G, including data generation, model setup, training procedures, and evaluation metrics.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide an anonymized code repository with clear instructions to reproduce all main experimental results.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We provide all relevant experimental details, including data splits, training configurations, and hyperparameter settings in Section 3, Appendix F, and Appendix G, ensuring clarity and reproducibility.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We report the numbers in mean and standard deviation of the objective error over five random seeds. Even considering the error bar, our empirical results are still significant.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We specify the compute setup, including device type and per-epoch computation time, in Section 3. This includes the use of an NVIDIA L40S GPU and detailed timing in Table 6, sufficient for reproduction.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have thoroughly reviewed the NeurIPS Code of Ethics and confirm that our research complies with its principles in all respects.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss broader impacts in Appendix B, including potential benefits such as improving solver efficiency and optimization-based decision-making, as well as potential risks related to misuse in applications.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper does not release any pretrained models or real-world datasets with risk of misuse.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All existing assets used in the paper, such as graph contrastive learning baselines and synthetic datasets generation, are open-source and properly cited. No scraped data is used, all datasets are synthetically generated.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets`

has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We introduce a new suite of synthetic LP and QP datasets along with data augmentation tools for optimization learning tasks. These assets are well-documented and will be released under an open-source license.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This work does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This work does not involve research with human subjects or crowdsourcing.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.

- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: This work does not use LLMs as part of the core methods or contributions. Any LLM usage, if any, was limited to minor writing assistance and does not affect the scientific rigor or originality of the research.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Limitations

While our proposed data augmentation framework for QP and LP instances is efficient and principled, several limitations remain. First, some transformations, e.g., Appendix C.2 and Section 2.2, require access to the optimal primal-dual solution of the original problem, which may not always be available in practice. Second, our framework assumes convexity (i.e., $\boldsymbol{Q} \succ 0$) and problem feasibility, and does not directly extend to non-convex or infeasible cases. Additionally, non-linear quadratic programming is beyond our scope. Also, we train and evaluate solely on synthetic QP and LP instances because real-world datasets are scarce and heterogeneous. Benchmarks such as MIPLIB [Gleixner et al., 2021] and QPLIB [Furini et al., 2019] contain too few samples and exhibit significant variability in problem size and structure, making them unsuitable for training deep models. Finally, while our transformation framework is mathematically expressive, we restrict it to a small subset of efficient, feasible augmentations in practice, leaving its full expressive power unexplored.

B Broader impact

This work introduces principled data augmentation methods for learning-based solvers in convex optimization, focusing on predicting objective values and solutions for LP/QP problems. These methods can also replace heuristics, such as strong branching in mixed-integer optimization. Our approach may benefit logistics, finance, and scientific computing applications by enabling more robust and data-efficient learning. As a foundational contribution, this work does not raise concerns about privacy, security, or fairness. It does not involve sensitive data or end-user interaction, and the proposed techniques are general-purpose and not tied to specific domains. Nevertheless, when applied to real-world optimization systems, care should be taken to ensure reliability and fairness.

C Additional transformations

Here, we outline additional transformations, omitted from the main paper for space reasons.

C.1 Adding constraints

Analogous to adding variables (see Section 2.2), we can augment a problem instance by introducing additional constraints. This is done by extending the constraint matrix using a transformation of the form $\boldsymbol{M}_2 \coloneqq \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{m}^\intercal \end{bmatrix}$, where $\boldsymbol{m} \in \mathbb{R}^m_{\geq 0}$ is a non-negative vector. This effectively appends a new constraint that is a convex combination of the existing ones. Notably, this transformation, along with the scaling constraints in Section 2.2, satisfies the condition derived in Eq. (13). In unsupervised settings where neither the primal-dual solutions nor the active constraints are known, we store all constraints and add a new one that is linearly combined with them and does not affect the solution. The added constraint remains inactive, corresponding to a zero dual variable, transforming both *efficiently recoverable* and *solution-independent*.

Proposition C.1. *Let $I = (\boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ be an LCQP instance with optimal solution $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$. We define our transformation as $T(I) \coloneqq (\boldsymbol{Q}, \boldsymbol{M}_2\boldsymbol{A}, \boldsymbol{M}_2\boldsymbol{b}, \boldsymbol{c})$, where $\boldsymbol{M}_2 \coloneqq \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{m}^\intercal \end{bmatrix}$. Then the transformed instance has an optimal solution $\boldsymbol{x}^*, [\boldsymbol{\lambda}^*, \lambda'^*]$ if and only if the new dual variable $\lambda'^* = 0$.*

C.2 Biasing the problem

We introduce a data augmentation strategy that leverages nontrivial bias terms while fixing $\boldsymbol{M}$ to the identity. Specifically,

$$\left(\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}^\intercal \\ \boldsymbol{A} & \boldsymbol{0} \end{bmatrix} + \boldsymbol{B} \right) \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \begin{bmatrix} -\boldsymbol{c} \\ \boldsymbol{b} - \boldsymbol{s}^* \end{bmatrix} + \boldsymbol{\beta}. \tag{12}$$

Consider the transformation in Eq. (12), where we fix $\boldsymbol{M} \coloneqq \boldsymbol{I}$ and design a suitable bias matrix $\boldsymbol{B}$. To satisfy the conditions in Eq. (9), we construct $\boldsymbol{B}_{11} \coloneqq \boldsymbol{R}\boldsymbol{R}^\intercal$, where $\boldsymbol{R} \in \mathbb{R}^{n \times k}$ is a random matrix, ensuring that the resulting $\boldsymbol{Q}$ remains positive definite. We then set $\boldsymbol{B}_{21} = \boldsymbol{B}_{12}^\intercal \in$

$\mathbb{R}^{m\times n}$ as a random matrix, and $\boldsymbol{B}_{22} \coloneqq \mathbf{0}$. Accordingly, we must have $\boldsymbol{\beta} \coloneqq \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix}$, ensuring the transformed KKT system is satisfied. This yields the transformed instance $T(I) \coloneqq (\boldsymbol{Q} + \boldsymbol{B}_{11}, \boldsymbol{A} + \boldsymbol{B}_{21}, \boldsymbol{b} + \boldsymbol{B}_{21}\boldsymbol{x}^*, \boldsymbol{c} - \boldsymbol{B}_{11}\boldsymbol{x}^* + \boldsymbol{A}^\intercal\boldsymbol{\lambda}^*)$ under which the original primal and dual solutions remain valid; therefore, it is an efficiently recoverable transformation. However, this transformation is not solution-independent, since it requires access to $\boldsymbol{x}^*, \boldsymbol{\lambda}^*$, which is an intrinsic drawback of such biasing transformations.

C.3 Finding inactive constraints

We introduce the following heuristics. For all inequality constraints in a given problem, including the variable bounds, e.g., $x_i \geq 0$, we calculate a score $h_i \coloneqq \boldsymbol{a}_i^\intercal \boldsymbol{x} + b_i$. The constraints with lower scores are more likely to be active. See Appendix C.3 for an illustration. Given the number of constraints m

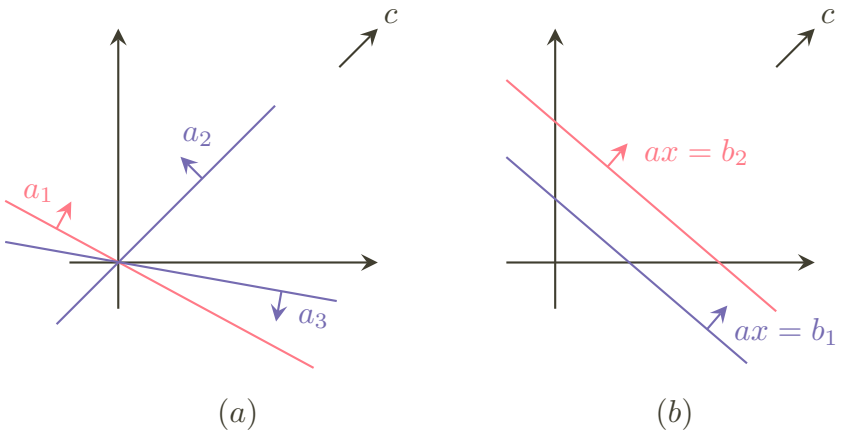

Figure 2: Illustration of our heuristics. (a). Smaller $\boldsymbol{a}^\intercal\boldsymbol{c}$ is more likely to be inactive. (b). Lower b is more likely to be inactive.

and the number of variables n, if $m > n$ (due to variable bounds constraints) and the constraints are not degenerate, n constraints will be active. Therefore, we pick the $m - n$ constraints with the largest h_i as heuristic inactive constraints. Then this could be plugged into the removing constraint transformation.

We evaluate the effectiveness on our datasets, by calculating the accuracy given ground truth inactive constraint set $\bar{\mathcal{A}}_{\text{gt}}$ and heuristic one $\bar{\mathcal{A}}_{\text{heu}}$,

$$\text{acc} \coloneqq \frac{|\bar{\mathcal{A}}_{\text{gt}} \cap \bar{\mathcal{A}}_{\text{heu}}|}{\bar{\mathcal{A}}_{\text{heu}}}$$

over all the instances; see Table 5 for a summary of our results.

Table 5: Accuracy summary.

	LP				QP			
Size	100	150	200	250	100	150	200	250
Acc. (%)	88.5±2.9	90.1±3.2	89.1±2.8	88.3±3.6	91.8±3.4	92.9±2.7	91.6±2.1	91.4±2.4

The heuristic's performance is satisfactory, with a low false-positive rate. Although it is generally designed for LPs, it is also effective for QPs.

C.4 Adding variable

Besides Proposition 2.6, we derive another efficient method to add a new variable.

Proposition C.2. *Given an LCQP instance $I \coloneqq (\boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ and its optimal primal and dual solution $\boldsymbol{x}^*, \boldsymbol{\lambda}^*$, the data augmentation $T(I) \coloneqq \left(\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{0}^\intercal & q \end{bmatrix}, [\boldsymbol{A} \quad \boldsymbol{a}], \boldsymbol{b}, \begin{bmatrix} \boldsymbol{c} \\ c' \end{bmatrix} \right)$, where $q > 0$, $\boldsymbol{a} \in \mathbb{R}^m$ is arbitrary, $c' \coloneqq -\boldsymbol{a}^\intercal \boldsymbol{\lambda}^*$, allows for an extra variable x' without affecting the original optimal solution, if and only if the new variable $x'^* = 0$.*

Such data transformation construction is efficient and valid. The only drawback is that it is not solution-independent. Therefore, we provide an improved augmentation that introduces a new constraint.

Corollary C.2.1. *Given a QP instance $I \coloneqq (\boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ and its optimal primal and dual solution $\boldsymbol{x}^*, \boldsymbol{\lambda}^*$, the data augmentation $T(I) \coloneqq \left(\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{0}^\intercal & q \end{bmatrix}, \begin{bmatrix} \boldsymbol{A} & \boldsymbol{a} \\ \boldsymbol{0}^\intercal & 1 \end{bmatrix}, \begin{bmatrix} \boldsymbol{b} \\ 0 \end{bmatrix}, \begin{bmatrix} \boldsymbol{c} \\ c' \end{bmatrix} \right)$, where $q > 0$, $\boldsymbol{a} \in \mathbb{R}^m$, $c' \in \mathbb{R}$ is arbitrary, allows for an extra variable x' an extra dual variable λ' without affecting the original optimal solution if and only if the new primal variable $x'^* = 0$.*

Proposition C.3. *There exists a set of matrices $\{\boldsymbol{M}, \boldsymbol{B}, \boldsymbol{\beta}\}$, such that the design in Corollary C.2.1 can be realized in the form of Eq.* (7)*.*

D Omitted proofs

Here, we outline missing proofs from the main paper.

D.1 Proof for Proposition 2.1

Proof. The rank condition of $\boldsymbol{N}_{11}$ is straightforward. Calculating the (pseudo) inverse $\begin{bmatrix} \boldsymbol{N}_{11}^\intercal & \boldsymbol{N}_{21}^\intercal \\ \boldsymbol{N}_{12}^\intercal & \boldsymbol{N}_{22}^\intercal \end{bmatrix}^\dagger$ requires $\boldsymbol{N}_{11}$ or $\boldsymbol{N}_{22}$ being invertible. From the KKT condition Eq. (3d), we know $\boldsymbol{\lambda}_{\bar{a}}^* = \boldsymbol{0}$. So we don't have to consider ${\boldsymbol{N}_{22}^\intercal}^{-1}$ or its existence; however, $\boldsymbol{N}_{11}$ must be invertible, otherwise we cannot solve the optimal solution of the transformed problem without a QP solver. Intuitively speaking, if $\boldsymbol{N}_{11}$ is not of full rank, we will drop some equality constraints, which might cause the solution to be relaxed, thus requiring us to solve the transformed QP problem with a QP solver.

Let us consider $\boldsymbol{N}_{12}$. We rewrite the second equation of Eq. (8) as

$$\begin{bmatrix} \boldsymbol{N}_{11} & \boldsymbol{N}_{12} \\ \boldsymbol{N}_{21} & \boldsymbol{N}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{A}_a \\ \boldsymbol{A}_{\bar{a}} \end{bmatrix} \boldsymbol{x}^* = \begin{bmatrix} \boldsymbol{N}_{11} & \boldsymbol{N}_{12} \\ \boldsymbol{N}_{21} & \boldsymbol{N}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{b}_a \\ \boldsymbol{b}_{\bar{a}} - \boldsymbol{s}_{\bar{a}}^* \end{bmatrix}. \tag{13}$$

The first equation leads to

$$\boldsymbol{N}_{11} \boldsymbol{A}_a \boldsymbol{x}^* + \boldsymbol{N}_{12} \boldsymbol{A}_{\bar{a}} \boldsymbol{x}^* = \boldsymbol{N}_{11} \boldsymbol{b}_a + \boldsymbol{N}_{12} \boldsymbol{b}_{\bar{a}} - \boldsymbol{N}_{12} \boldsymbol{s}_{\bar{a}}^*, \tag{14}$$

which is equivalent to constructing new equality constraints with a linear combination of current constraints. We observe that $\boldsymbol{x}^*$ remains for the transformed problem if and only if $\boldsymbol{N}_{12} \boldsymbol{s}_{\bar{a}}^* = \boldsymbol{0}$. A special case is $\boldsymbol{N}_{12} = \boldsymbol{0}$, which removes the effect of inactive constraints on active ones. Otherwise, we must resolve this equation for $\boldsymbol{x}^*$. Intuitively speaking, the margins of inactive constraints $\boldsymbol{s}^*$ might relax or tighten an existing equation constraint, and we will have to solve the linear equation Eq. (13) rather than lazily copying the $\boldsymbol{x}^*$ for the new problem.

The condition of $\boldsymbol{N}_{22}$ is from the fact that if we multiply a negative constant on an inequality $a \leq b$, the inequality direction will flip. Let us look at the second equation of Eq. (13). We shall have

$$\boldsymbol{N}_{21} \boldsymbol{A}_a \boldsymbol{x}^* + \boldsymbol{N}_{22} \boldsymbol{A}_{\bar{a}} \boldsymbol{x}^* = \boldsymbol{N}_{21} \boldsymbol{b}_a + \boldsymbol{N}_{22} \left(\boldsymbol{b}_{\bar{a}} - \boldsymbol{s}_{\bar{a}}^* \right), \tag{15}$$

given $\boldsymbol{A}_a \boldsymbol{x}^* = \boldsymbol{b}_a$, we have

$$\boldsymbol{N}_{22} \boldsymbol{A}_{\bar{a}} \boldsymbol{x}^* = \boldsymbol{N}_{22} \left(\boldsymbol{b}_{\bar{a}} - \boldsymbol{s}_{\bar{a}}^* \right) \tag{16}$$

must hold. The matrix $\boldsymbol{N}_{22}$ gives us combinations for a set of new inequality constraints given the existing constraints. Let us take out a row $\boldsymbol{v}^\intercal$ from the $\boldsymbol{N}_{22}$,

$$\boldsymbol{v}^\intercal \boldsymbol{A}_{\bar{a}} \boldsymbol{x}^* = \boldsymbol{v}^\intercal \boldsymbol{b}_{\bar{a}} - \boldsymbol{v}^\intercal \boldsymbol{s}_{\bar{a}}^*, \tag{17}$$

yields a vector $\boldsymbol{a}^\intercal = \boldsymbol{v}^\intercal \boldsymbol{A}_{\bar{a}}$ on the LHS and a scalar $b = \boldsymbol{v}^\intercal \boldsymbol{b}_{\bar{a}}$ on the RHS. We know that for a new inequality constraint, we should guarantee $\boldsymbol{a}^\intercal \boldsymbol{x}^* \leq b$, so $\boldsymbol{v}^\intercal \boldsymbol{s}_{\bar{a}}^* \geq 0$ must be satisfied. Generalizing this to all rows of $\boldsymbol{N}_{22}$, we should have $\boldsymbol{N}_{22} \boldsymbol{s}_{\bar{a}}^* \geq \boldsymbol{0}$. □

D.2 Proof for Proposition 2.4

Lemma D.1. *KKT conditions are sufficient and necessary for LCQPs.*

See Boyd and Vandenberghe [2004, p. 244]. LCQPs satisfy the Slater conditions.

Proof. We extend the variable set by isolating the target variable x' and rewriting the problem with variables $\begin{bmatrix}\boldsymbol{x}\\ x'\end{bmatrix} \in \mathbb{R}^{n+1}$, and the corresponding coefficients as $\boldsymbol{Q}' = \begin{bmatrix}\boldsymbol{Q} & \boldsymbol{q}\\ \boldsymbol{q}^\intercal & q\end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}$, $\boldsymbol{A}' = [\boldsymbol{A} \quad \boldsymbol{a}] \in \mathbb{R}^{m\times(n+1)}$, and $\boldsymbol{c}' = \begin{bmatrix}\boldsymbol{c}\\ c\end{bmatrix} \in \mathbb{R}^{n+1}$, $\boldsymbol{b} \in \mathbb{R}^m$.

(If direction) Suppose $x'^* = 0$, we can drop it as well as its corresponding coefficients $\boldsymbol{q}, q, \boldsymbol{a}, c$. We shall have the KKT conditions Eqs. (3a) and (3b) satisfied for the optimal solutions $\begin{bmatrix}\boldsymbol{x}^*\\ x'^*\end{bmatrix}$ and $\boldsymbol{\lambda}^*$,

$$\boldsymbol{Q}'\begin{bmatrix}\boldsymbol{x}^*\\ 0\end{bmatrix} + \boldsymbol{A}'^\intercal\boldsymbol{\lambda}^* + \boldsymbol{c}' = \boldsymbol{0}$$
$$\boldsymbol{A}'\begin{bmatrix}\boldsymbol{x}^*\\ 0\end{bmatrix} \le \boldsymbol{b}.$$

We can introduce a row selection matrix $\boldsymbol{M}_1 \coloneqq [\boldsymbol{I} \quad \boldsymbol{0}] \in \{0,1\}^{n\times(n+1)}$, which effectively drops the last row, and projects the KKT conditions as

$$\boldsymbol{M}_1\boldsymbol{Q}'\boldsymbol{M}_1^\intercal\boldsymbol{M}_1\begin{bmatrix}\boldsymbol{x}^*\\ 0\end{bmatrix} + \boldsymbol{M}_1\boldsymbol{A}'^\intercal\boldsymbol{\lambda}^* + \boldsymbol{M}_1\boldsymbol{c}' = \boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{A}^\intercal\boldsymbol{\lambda}^* + \boldsymbol{c} = \boldsymbol{0}$$
$$\boldsymbol{A}'\boldsymbol{M}_1^\intercal\boldsymbol{M}_1\begin{bmatrix}\boldsymbol{x}^*\\ 0\end{bmatrix} = \boldsymbol{A}\boldsymbol{x}^* \le \boldsymbol{b}.$$

Thus, removing x' along with its associated coefficients preserves primal and dual feasibility, and $\boldsymbol{x}^*$ remains optimal for the reduced problem.

(Only if direction) Suppose we can remove x' without affecting the optimality of $\boldsymbol{x}^*$. Then the reduced and full KKT conditions must be consistent for all possible coefficients associated with x'. That is,

$$\begin{aligned}\boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{A}^\intercal\boldsymbol{\lambda}^* + \boldsymbol{c} &= \boldsymbol{0}\\ \boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{q}x'^* + \boldsymbol{A}^\intercal\boldsymbol{\lambda}^* + \boldsymbol{c} &= \boldsymbol{0}\\ \boldsymbol{q}^\intercal\boldsymbol{x}^* + qx'^* + \boldsymbol{a}^\intercal\boldsymbol{\lambda}^* + c' &= 0\\ \boldsymbol{A}\boldsymbol{x}^* &\le \boldsymbol{b}\\ \boldsymbol{A}\boldsymbol{x}^* + \boldsymbol{a}x'^* &\le \boldsymbol{b}.\end{aligned}$$

For these conditions to hold for *all* choices of $\boldsymbol{q}, q, \boldsymbol{a}$, it must be that $x'^* = 0$; otherwise, the residual terms would depend on those coefficients and the solution would vary. Hence, $x'^* = 0$ is necessary for the removal to be valid without affecting the remaining solution. □

D.3 Proof for Proposition 2.5

Proof. We rewrite the problem by appending the target constraint as an additional row. The constraint matrix becomes $\boldsymbol{A}' = \begin{bmatrix}\boldsymbol{A}\\ \boldsymbol{a}'^\intercal\end{bmatrix} \in \mathbb{R}^{(m+1)\times n}$ and $\boldsymbol{b}' = \begin{bmatrix}\boldsymbol{b}\\ b'\end{bmatrix} \in \mathbb{R}^{(m+1)}$, and the corresponding dual variables are $\boldsymbol{\lambda}' = \begin{bmatrix}\boldsymbol{\lambda}\\ \lambda'\end{bmatrix} \in \mathbb{R}^{(m+1)}$.

(If direction) Suppose the additional constraint is strictly inactive, i.e., $\lambda'^* = 0$. We can introduce a row selection matrix $\boldsymbol{M}_2 \coloneqq [\boldsymbol{I} \quad \boldsymbol{0}] \in \{0,1\}^{m\times(m+1)}$. The KKT conditions for the full system are,

$$\boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{A}'^\intercal\boldsymbol{\lambda}'^* + \boldsymbol{c} = \boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{A}^\intercal\boldsymbol{\lambda}^* + \boldsymbol{a}'\lambda'^* + \boldsymbol{c} = \boldsymbol{0}$$
$$\begin{bmatrix}\boldsymbol{A}\\ \boldsymbol{a}'^\intercal\end{bmatrix}\boldsymbol{x}^* \le \begin{bmatrix}\boldsymbol{b}\\ b'\end{bmatrix}.$$

Since $\lambda'^* = 0$, we can write.

$$\boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{A}'^\intercal \boldsymbol{M}_2^\intercal \boldsymbol{M}_2 \boldsymbol{\lambda}'^* + \boldsymbol{c} = \boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{A}^\intercal \boldsymbol{\lambda}^* + \boldsymbol{c} = \boldsymbol{0}$$
$$\boldsymbol{M}_2 \boldsymbol{A}' \boldsymbol{x}^* = \boldsymbol{A}\boldsymbol{x}^* \leq \boldsymbol{b} = \boldsymbol{M}_2 \boldsymbol{b}'.$$

This corresponds to the KKT conditions of the original problem with the constraint removed. Thus, removing the inactive constraint does not affect optimality.

(Only if direction) Suppose the constraint is removed and the optimal solution $\boldsymbol{x}^*, \boldsymbol{\lambda}^*$ remains valid for all possible coefficients $\boldsymbol{a}', b'$. Then both of the following must hold,

$$\boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{A}'^\intercal \boldsymbol{\lambda}'^* + \boldsymbol{c} = \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{A}^\intercal \boldsymbol{\lambda}^* + \boldsymbol{a}' \lambda'^* + \boldsymbol{c} = \boldsymbol{0}$$
$$\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{A}^\intercal \boldsymbol{\lambda}^* + \boldsymbol{c} = \boldsymbol{0}.$$

Subtracting the two equations yields $\boldsymbol{a}' \lambda'^* = \boldsymbol{0}$. For this to hold for arbitrary $\boldsymbol{a}'$, it must be that $\lambda'^* = 0$. Hence, the constraint must have been inactive. □

D.4 Proof for Proposition 2.6

Proof. **(If direction)** Let $\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{x} \\ x' \end{bmatrix}$ with $x'^* = 0$, we can verify that

$$\boldsymbol{M}_1 \boldsymbol{Q} \boldsymbol{M}_1^\intercal \boldsymbol{x}'^* + \boldsymbol{M}_1 \boldsymbol{A}^\intercal \boldsymbol{\lambda}^* + \boldsymbol{M}_1 \boldsymbol{c} = \boldsymbol{M}_1 \left(\boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{A}^\intercal \boldsymbol{\lambda}^* + \boldsymbol{c} \right) = \boldsymbol{0}$$
$$\boldsymbol{A}\boldsymbol{M}_1^\intercal \boldsymbol{x}'^* = \boldsymbol{A}\boldsymbol{x}^* \leq \boldsymbol{b}.$$

Thus, $(\boldsymbol{x}'^*, \boldsymbol{\lambda}^*)$ satisfies the KKT conditions for the transformed problem.

(Only if direction) The equation

$$\boldsymbol{A}\boldsymbol{M}_1^\intercal \boldsymbol{x}'^* = \boldsymbol{A}\boldsymbol{x}^* + \boldsymbol{A}\boldsymbol{q}x'^* \leq \boldsymbol{b}$$

must hold for all choices of $\boldsymbol{q}$, therefore $x'^* = 0$. □

D.5 Proof for Proposition C.1

Proof. **(If direction)** Suppose $\lambda'^* = 0$, and the transformed KKT conditions

$$\boldsymbol{Q}\boldsymbol{x}^* + \begin{bmatrix} \boldsymbol{A}^\intercal & \boldsymbol{m} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda}^* \\ \lambda'^* \end{bmatrix} + \boldsymbol{c} = \boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{A}^\intercal \boldsymbol{\lambda}^* + \boldsymbol{c} = \boldsymbol{0}$$
$$\boldsymbol{M}_2 \boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{M}_2 \boldsymbol{b}$$

hold.

(Only if direction) The equations of pre- and post-transformation KKT must hold

$$\begin{aligned} \boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{A}^\intercal \boldsymbol{\lambda}^* + \boldsymbol{c} &= \boldsymbol{0} \\ \boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{A}^\intercal \boldsymbol{M}_2^\intercal \begin{bmatrix} \boldsymbol{\lambda}^* \\ \lambda'^* \end{bmatrix} + \boldsymbol{c} &= \boldsymbol{0} \end{aligned} \tag{18}$$

for all choices of $\boldsymbol{m}$, therefore, $\lambda' = 0$. □

D.6 Proof for Section C

Proof for Appendix C.4:

Proof. **(If direction)** Given an LCQP instance $I \coloneqq (\boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ and its optimal primal and dual solution $\boldsymbol{x}^*, \boldsymbol{\lambda}^*$, new variable $x'^* = 0$, the following holds,

$$\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{0}^\intercal & q \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^* \\ x'^* \end{bmatrix} + \begin{bmatrix} \boldsymbol{A}^\intercal \\ \boldsymbol{a}^\intercal \end{bmatrix} \boldsymbol{\lambda}^* = - \begin{bmatrix} \boldsymbol{c} \\ c' \end{bmatrix}$$
$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{a} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^* \\ x'^* \end{bmatrix} = \boldsymbol{b} - \boldsymbol{s}^*.$$

(Only if direction) Similar to the proof of Proposition 2.6. □

Proof for Corollary C.2.1:

Proof. **(If direction)** Let $\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{x} \\ x' \end{bmatrix}$ with $x'^* = 0$, we have

$$\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{0}^\intercal & q \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{x}'^* \end{bmatrix} + \begin{bmatrix} \boldsymbol{A}^\intercal & \boldsymbol{0} \\ \boldsymbol{a}^\intercal & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda}^* \\ \lambda'^* \end{bmatrix} + \begin{bmatrix} \boldsymbol{c} \\ c' \end{bmatrix} = \boldsymbol{0}$$
$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{a} \\ \boldsymbol{0}^\intercal & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{x}'^* \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{b} \\ 0 \end{bmatrix},$$

which is satisfiable for some λ'^*.

(Only if direction) The inequality,

$$[\boldsymbol{A} \quad \boldsymbol{a}] \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{x}'^* \end{bmatrix} \leq \boldsymbol{b}$$

must hold for any choices of $\boldsymbol{a}$, therefore $x'^* = 0$. □

Proof for Proposition C.3:

Proof. Prove by construction. We can have $\boldsymbol{M} \coloneqq \boldsymbol{I}$, and $\boldsymbol{B}_{11}, \boldsymbol{B}_2 \coloneqq \boldsymbol{0}$, $\boldsymbol{B}_{21} = \boldsymbol{B}_{12}^\intercal \coloneqq \begin{bmatrix} \boldsymbol{0} & \boldsymbol{a} \\ \boldsymbol{0}^\intercal & 1 \end{bmatrix}$, $\boldsymbol{\beta} \coloneqq \begin{bmatrix} \boldsymbol{0} \\ -c' \\ \boldsymbol{0} \\ 0 \end{bmatrix}$. □

E Additional experiments

Here, we provide results for additional experiments.

E.1 Computational overhead

To address **Q4**, we compare the per-epoch training time for regular training, data augmentation, and contrastive pretraining. Since data augmentation runs on CPU and model training on GPU, their absolute computation times are not directly comparable. Instead, we report the training time (excluding validation) with and without augmentation to evaluate practical overhead. Results in Table 6 show the mean and standard deviation over 100 epochs, using batch size 32 (128 for pretraining) and `num_workers` $= 4$.

Table 6: Per-epoch training time (in seconds) over 100 epochs. Data augmentation adds negligible overhead. Pretraining uses a larger batch size and remains efficient.

	LP			QP		
Aug.	10%	100%	Pretrain	10%	100%	Pretrain
No	$1.005_{\pm 0.097}$	$7.752_{\pm 0.143}$	N/A	$1.244_{\pm 0.076}$	$10.374_{\pm 0.158}$	N/A
Yes	$0.975_{\pm 0.061}$	$7.822_{\pm 0.197}$	$7.699_{\pm 0.151}$	$1.235_{\pm 0.069}$	$10.247_{\pm 0.343}$	$9.648_{\pm 0.139}$

E.2 Size generalization

For both LPs and QPs, we generate larger instances with increased numbers of variables and constraints (proportional to the original sizes) while maintaining a fixed average graph degree. The exact parameters are detailed in Appendix F. As shown in Table 7, increasing instance size improves performance across all methods, suggesting some extent of size generalization. Importantly, models pre-trained consistently outperform models trained from scratch across all problem sizes and levels of data scarcity.

Table 7: Generalization performance of contrastive pretrained MPNNs on larger LP and QP instances. Models pretrained with contrastive learning consistently outperform those trained from scratch.

		LP			QP		
Size	**Pretrained**	10%	20%	100%	10%	20%	100%
100%	No	8.539±0.206	6.149±0.153	2.784±0.081	5.304±0.229	3.567±0.034	1.240±0.088
	Yes	3.472±0.086	2.794±0.049	1.588±0.056	3.791±0.097	2.427±0.083	0.926±0.031
150%	No	6.707±0.052	4.968±0.095	2.431±0.032	4.253±0.166	2.947±0.070	0.986±0.002
	Yes	2.948±0.225	2.304±0.029	1.331±0.044	2.999±0.120	2.021±0.051	0.766±0.031
200%	No	5.820±0.069	4.173±0.085	2.052±0.087	3.971±0.028	2.474±0.049	0.847±0.044
	Yes	2.624±0.091	2.122±0.102	1.188±0.048	2.688±0.103	1.793±0.103	0.654±0.033
250%	No	5.275±0.082	3.759±0.049	1.912±0.023	3.708±0.089	2.382±0.065	0.775±0.030
	Yes	2.451±0.108	1.912±0.063	1.121±0.029	2.601±0.232	1.671±0.054	0.605±0.023

E.3 Other pretraining

Besides contrastive pretraining, we also pretrain with supervised learning with and without augmentation on the random LP dataset, and test its OOD performance on the set cover dataset. As shown in Table 8, supervised pretraining performs surprisingly well, especially with data augmentation.

Table 8: Generalization performance of supervised pretrained MPNNs on OOD set cover instances.

	Ratio		
Pretrained	10%	20%	100%
No	5.297±0.787	2.531±0.861	0.752±0.042
CL	1.965±0.206	1.349±0.161	0.662±0.055
Su.	1.994±0.115	1.521±0.085	0.778±0.006
Su.+aug.	**1.034±0.055**	**0.848±0.045**	**0.545±0.026**

E.4 Ablation on temperature parameter

We conduct experiments on the temperature parameter τ in the contrastive loss $Eq.$ (11). We use the same setting as in Table 2, pick $\tau \in \{0.01, 0.1, 1\}$, and run it with our method on LP instances. The results are summarized as in Table 9. As shown, $\tau = 0.1$ is consistently the best option.

Table 9: Ablation on the temperature parameter τ.

	Ratio		
τ	10%	20%	100%
0.01	3.483±0.074	2.903±0.078	1.671±0.053
0.1	**3.472±0.086**	**2.794±0.049**	**1.588±0.056**
1	4.269±0.197	2.802±0.032	1.628±0.046

E.5 QPLIB

We evaluate QPLIB [Furini et al., 2019]. Due to the limited size, large scale, and extreme heterogeneity of QPLIB instances, performing a standard train/validation/test split is infeasible. Existing works that use QPLIB for evaluation typically rely on perturbing problem coefficients [Wu et al., 2024, Yang et al., 2024b]. These approaches, however, have significant limitations:

1. The augmentations are not label-preserving and thus require solving each perturbed instance from scratch;
2. Perturbations may break feasibility, which limits the perturbation strength;
3. As a result, weaker perturbations are often used, but they yield training instances that are nearly identical to the test instances, making evaluation less meaningful.

Our proposed method, on the other hand, includes both structural and feature perturbation and is targeted at better generalization performance. For the current perturbation method and ours to work well, one can reduce the perturbation rate to an arbitrarily small value to fit the test data perfectly, which appears nonsensical. To study transferability and fitting efficiency on QPLIB, we conduct a pre-training experiment.

1. We generate a foundation dataset of 10000 large-scale random QP instances with sizes ranging from 1000 to 1500 variables and constraints, which are 100–200 times larger than the problems used in Table 2. Since no labels are needed for contrastive pretraining, data generation is efficient. Training takes around 30 seconds per epoch using 4 NVIDIA L40S GPUs.
2. We select feasible LCQP instances from QPLIB, relax integer constraints, and train an MPNN to fit these problems in a supervised setting. We compare models trained from scratch with models initialized from the pretrained weights.

Here are the problem statistics and results:

Table 10: Statistics of selected QPLIB instances.

Name	cons.	vars.	A density	Q density
QPLIB_3694	3280	3240	0.001208	0.000313
QPLIB_3708	12917	12930	0.000171	0.000628
QPLIB_3861	4650	4530	0.000856	0.000222
QPLIB_3871	1040	1025	0.003772	0.001000
QPLIB_8559	5000	10000	0.000500	0.000700

We train each model for 100 epochs and compare predicted objectives to the actual optimal value. Longer training will diminish the advantage of pretraining, since we are testing on the same data we used for training. We repeat the training 3 times with random seeds and report the mean.

Table 11: Training performance on selected QPLIB instances. Pretraining substantially improves convergence.

Name	Pretrain	Obj.(optimal)	Obj.(predict)
QPLIB_3694	No	0.000	10.362
	Yes	0.000	0.142
QPLIB_3708	No	-42.469	-42.469
	Yes	-42.469	-42.469
QPLIB_3861	No	0.000	10.895
	Yes	0.000	-0.193
QPLIB_3871	No	0.000	7.205
	Yes	0.000	-0.193
QPLIB_8559	No	15.793	29.208
	Yes	15.793	12.331

As shown in Table 11, pretrained models generally fit faster and more accurately within limited training epochs. QPLIB_3708 is an exception, though both models converge very close to the actual objective. These results support the scalability and transferability of our method.

F Hyperparameters

This section lists some crucial hyperparameters except those already described in Section 3.

In supervised learning, we generate augmented data on the fly and control the degree of perturbation using an *augmentation strength* parameter α. For example, a node-dropping strength of $\alpha = 0.3$ removes 30% of the graph's nodes. We treat the variable-constraint graph as homogeneous for baselines from You et al. [2020]. For scaling-based augmentations (e.g., variable or constraint coefficients), we interpret strength α as multiplication by e^{α}. For constraint addition, we add a fraction α of new

constraints, each a convex combination of three existing constraints. To balance original and augmented views, we sample a scaled strength $\alpha' := \alpha\epsilon$, where $\epsilon \sim \mathcal{U}(0, 1)$, ensuring smooth interpolation between unaltered and fully augmented data. The strengths used in our experiments are listed below.

Table 12: Augmentation strength of various augmentations in Table 1.

Augmentation	LP	QP
Drop node	0.10	0.05
Mask node	0.10	0.05
Flip edge	0.10	0.05
Drop vars.	0.99	0.99
Drop cons.	0.99	0.99
Scale cons.	1.00	1.00
Scale vars.	1.00	1.00
Add cons.	0.50	0.50
Add vars.	0.80	0.60

For the combined augmentations, we use the following combination for both LP and QP, and for each instance, we sample two augmentations from the list below.

Table 13: Augmentation strength of combined augmentations in Table 1.

Augmentation	Strength
Drop vars.	0.00
Drop cons.	0.50
Scale cons.	0.50
Scale vars.	0.50
Add cons.	0.60
Add vars.	0.00

For contrastive pretraining, we omit interpolation and apply fixed augmentation strengths. For GraphCL [You et al., 2020], we tune the strengths of its three components: for LP, we use 0.2 node dropping, 0.1 edge flipping, and 0.13 feature masking; for QP, 0.3, 0.17, and 0.15, respectively. For GCC [Qiu et al., 2020], we use a random walk length of 50 for LP and 200 for QP. IGSD [Zhang et al., 2023] and MVGRL [Hassani and Khasahmadi, 2020] apply graph diffusion with 10% edge addition. InfoGraph [Sun et al., 2019], DGI [Veličković et al., 2018], and GAE [Kipf and Welling, 2016] require no additional hyperparameters. For our method, we use the following configuration.

Table 14: Augmentation strength of combined augmentations in Table 2.

Augmentation	LP	QP
Drop cons. (heuristic)	0.05	0.07
Scale cons.	0.40	1.03
Scale vars.	1.07	0.65
Add cons.	0.36	0.33
Add vars.	0.46	0.26

G Data generation

We use the following pseudo code for random LP and QP generation; see Algorithms 1 and 2.

For the instances used in Tables 1 and 2 and Table 7, we generate the instances with sizes as described in Table 15.

For the OOD LP instances in Table 3, we follow the generation procedure in Gasse et al. [2019]. The idea is to keep the dimensions and the density of the matrix $\boldsymbol{A}$ similar to those of the problems we pretrain on. Hence, we generate with 100 rows and columns and 5% $\boldsymbol{A}$ matrix density for SC; 100

Algorithm 1 Generate a sparse feasible LP instance.

Require: Number of constraints m, number of variables n, constraint density ρ_A, random seed
Ensure: Matrices $(\boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b})$ defining an LP
1: Initialize random number generator `rng` with the given seed
2: Generate sparse random matrix $\boldsymbol{A} \sim \mathcal{N}(0,1)^{m \times n}$ with density ρ_A
3: Sample $\boldsymbol{x}_{\text{raw}} \sim \mathcal{N}(0,1)^n$
4: Set $\boldsymbol{x} \coloneqq |\boldsymbol{x}_{\text{raw}}|$ (elementwise absolute value)
5: Compute $\boldsymbol{s}_{\text{noise}} \sim \mathcal{N}(0,1)^m$ (slack noise)
6: Set $\boldsymbol{b} \coloneqq \boldsymbol{A}\boldsymbol{x} + |\boldsymbol{s}_{\text{noise}}|$
7: Sample random linear term $\boldsymbol{c} \sim \mathcal{N}(0,1)^n$
8: **return** $(\boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b})$

Algorithm 2 Generate a sparse feasible QP instance.

Require: Number of constraints m, number of variables n, constraint density ρ_A, matrix density ρ_P, random seed
Ensure: Matrices $(\boldsymbol{Q}, \boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b})$ defining a QP
1: Initialize random number generator `rng` with the given seed
2: Generate sparse random matrix $\boldsymbol{A} \sim \mathcal{N}(0,1)^{m \times n}$ with density ρ_A
3: Sample $\boldsymbol{x}_{\text{raw}} \sim \mathcal{N}(0,1)^n$
4: Set $\boldsymbol{x} \coloneqq |\boldsymbol{x}_{\text{raw}}|$ (elementwise absolute value)
5: Compute $\boldsymbol{s}_{\text{noise}} \sim \mathcal{N}(0,1)^m$ (slack noise)
6: Set $\boldsymbol{b} \coloneqq \boldsymbol{A}\boldsymbol{x} + |\boldsymbol{s}_{\text{noise}}|$
7: Sample random linear term $\boldsymbol{c} \sim \mathcal{N}(0,1)^n$
8: Generate sparse positive semi-definite matrix $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$ using SciPy `make_sparse_spd_matrix` with sparsity parameter $\alpha = 1 - \rho_P$
9: **return** $(\boldsymbol{Q}, \boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b})$

nodes and 0.02 edge probability for MIS; 100 items and 100 bids for CA; and 25 customers, three facilities, 0.5 ratio for CFL problem.

For the OOD QP instances, we generate SVM problems with Algorithm 3 with 100 samples and 0.05 density.

Algorithm 3 Generate soft-margin SVM QP instance.

Require: Number of samples n, number of features d, regularization parameter λ, feature density ρ, random number generator `rng`
Ensure: Matrices $(\boldsymbol{Q}, \boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b})$ defining a QP
1: Generate positive samples matrix $\boldsymbol{A}_1 \sim \mathcal{N}\left(\frac{1}{d\rho}, \frac{1}{d\rho}\right)^{\frac{n}{2} \times d}$
2: Generate negative samples matrix $\boldsymbol{A}_2 \sim \mathcal{N}\left(-\frac{1}{d\rho}, \frac{1}{d\rho}\right)^{\frac{n}{2} \times d}$
3: Stack data: $\boldsymbol{A} \coloneqq \begin{bmatrix} \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \end{bmatrix}$
4: Sparsify $\boldsymbol{A}$ with density ρ
5: Construct label vector $\boldsymbol{y} \coloneqq [1, \ldots, 1, -1, \ldots, -1] \in \{-1, 1\}^n$
6: Apply labels to data: $\boldsymbol{A} \coloneqq \boldsymbol{A} \cdot \boldsymbol{y}^\intercal$
7: Form constraint matrix: $\boldsymbol{A} \coloneqq -[\boldsymbol{A} \;\; \boldsymbol{I}_n]$
8: Set right-hand side: $\boldsymbol{b} \coloneqq -\mathbf{1}_n$
9: Define quadratic matrix $\boldsymbol{Q} \coloneqq \text{diag}([1, \ldots, 1, 0, \ldots, 0]) \in \{0, 1\}^{(d+n) \times (d+n)}$
10: Define linear term: $\boldsymbol{c} \coloneqq [0, \ldots, 0, \lambda, \ldots, \lambda] \in \{0, \lambda\}^{d+n}$
11: **return** $(\boldsymbol{Q}, \boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b})$

We generate portfolio problems as Algorithm 4, with 100 assets and 0.05 density.

We generate LASSO problems using Algorithm 5 with 50 samples, 50 features, and a density of 0.05.

Table 15: Random instance generation hyperparameters.

	LP			QP			
Size	#Col	#Row	$\boldsymbol{A}$ density	#Col	#Row	$\boldsymbol{A}$ density	$\boldsymbol{Q}$ density
100%	100	100	0.05	100	100	0.05	0.05
150%	150	150	0.03	150	150	0.03	0.03
200%	200	200	0.025	200	200	0.025	0.025
250%	250	250	0.02	250	250	0.02	0.02

Algorithm 4 Generate mean-variance portfolio QP instance.

Require: Number of assets n, covariance matrix density ρ, random number generator `rng`
Ensure: Matrices $(\boldsymbol{Q}, \boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{A}_{\text{eq}}, \boldsymbol{b}_{\text{eq}})$ defining a QP

1: Generate sparse positive definite covariance matrix $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$ using `make_sparse_spd_matrix` with $\alpha = 1 - \rho$, and eigenvalues in $[0.1, 0.9]$
2: Set linear cost vector $\boldsymbol{c} \coloneqq \mathbf{0}_n$
3: Generate inequality constraint matrix $\boldsymbol{A} \sim \mathcal{N}(0, 0.01)^{1 \times n}$
4: Generate equality constraint matrix $\boldsymbol{A}_{\text{eq}} \coloneqq 0.01 \cdot \mathbf{1}_n^\top$
5: Set inequality right hand side: $\boldsymbol{b} \coloneqq [-1]$
6: Set equality right hand side: $\boldsymbol{b}_{\text{eq}} \coloneqq [1]$
7: **return** $(\boldsymbol{Q}, \boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{A}_{\text{eq}}, \boldsymbol{b}_{\text{eq}})$

Algorithm 5 Generate LASSO QP instance.

Require: Number of samples n, number of features d, feature density ρ, regularization parameter λ, random number generator `rng`
Ensure: Matrices $(\boldsymbol{Q}, \boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b})$ defining a QP

1: Generate sparse design matrix $\boldsymbol{X} \sim \mathcal{N}(0,1)^{n \times d}$ with density ρ
2: Sample true weight vector $\boldsymbol{w}_{\text{true}} \sim \mathcal{N}(0,1)^d$
3: Generate noise vector $\boldsymbol{\epsilon} \sim \mathcal{N}(0, 0.5)^n$
4: Compute target: $\boldsymbol{y} \coloneqq \boldsymbol{X}\boldsymbol{w}_{\text{true}} + \boldsymbol{\epsilon}$
5: Compute quadratic term: $\boldsymbol{Q}_0 \coloneqq \frac{1}{2}\boldsymbol{X}^\top \boldsymbol{X}$
6: Compute linear term: $\boldsymbol{c}_0 \coloneqq -\boldsymbol{X}^\top \boldsymbol{y}$
7: Form block matrix: $\boldsymbol{Q} \coloneqq \begin{bmatrix} \boldsymbol{Q}_0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{2d \times 2d}$
8: Form cost vector: $\boldsymbol{c} \coloneqq \begin{bmatrix} \boldsymbol{c}_0 \\ \lambda \cdot \mathbf{1}_d \end{bmatrix}$
9: Construct constraint matrix:

$$\boldsymbol{A} \coloneqq \begin{bmatrix} -\boldsymbol{I}_d & -\boldsymbol{I}_d \\ \boldsymbol{I}_d & -\boldsymbol{I}_d \end{bmatrix}$$

10: Set constraint right-hand side: $\boldsymbol{b} \coloneqq \mathbf{0}_{2d}$
11: **return** $(\boldsymbol{Q}, \boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b})$