ECAI 2025
I. Lynce et al. (Eds.)

doi:10.3233/FAIA250894

A Distance Metric for Mixed Integer Programming Instances

Gwen Maudet[a,*] **and Grégoire Danoy**[b,**]

[a]SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg
[b]FSTM/DCS, SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg
ORCID (Gwen Maudet): https://orcid.org/0000-0003-0340-2542, ORCID (Grégoire Danoy): https://orcid.org/0000-0001-9419-4210

Abstract. Mixed-integer linear programming (MILP) is a powerful tool for addressing a wide range of real-world problems, but it lacks a clear structure for comparing instances. A reliable similarity metric could establish meaningful relationships between instances, enabling more effective evaluation of instance set heterogeneity and providing better guidance to solvers, particularly when machine learning is involved. Existing similarity metrics often lack precision in identifying instance classes or rely heavily on labeled data, which limits their applicability and generalization. To bridge this gap, this paper introduces the first mathematical distance metric for MILP instances, derived directly from their mathematical formulations. By discretizing right-hand sides, weights, and variables into classes, the proposed metric draws inspiration from the Earth mover's distance to quantify mismatches in weight-variable distributions for constraint comparisons. This approach naturally extends to enable instance-level comparisons. We evaluate both an exact and a greedy variant of our metric under various parameter settings, using the StrIPLIB dataset. Results show that all components of the metric contribute to class identification, and that the greedy version achieves accuracy nearly identical to the exact formulation while being nearly 200-times faster. Compared to state-of-the-art baselines—including feature-based, image-based, and neural network models—our unsupervised method consistently outperforms all non-learned approaches and rivals the performance of a supervised classifier on class and subclass grouping tasks.

1 Introduction

Mixed-integer linear programming (MILP) is a fundamental tool for formulating and solving optimization problems. It involves optimizing an objective function subject to constraints that include both continuous and discrete decision variables. This flexibility allows MILP to address a wide variety of real-world problems across diverse fields, including transportation [1], manufacturing [23], and e-health services [18].

The MILP space remains vast and unstructured in terms of the links between instances, and the introduction of structure within it, as discussed in this paper, yields two primary benefits. First, it facilitates the characterization of sets of instances. For instance, it enables the creation of heterogeneous benchmarks, as exemplified by the MIPLIB 2017 library [7]. This also allows for the evaluation of the heterogeneity of an evaluation instance set, offering insights into the generalizability of the assessed methods. Second, incorporating structural information can significantly enhance solver performance. In particular, machine learning (ML)-based approaches integrated into MILP solvers—commonly referred to as ML-MILPs—have demonstrated efficacy only within restricted subsets of similar instances [10, 4, 5, 27, 17]. Structuring the MILP space into clusters of homogeneous instances is therefore a critical step toward enabling the broader applicability of ML-MILP techniques across diverse problem domains.

Early attempts to structure the MILP space focused on problem classes definition based on recurring real-world problem types to structure MILPs [3]. However, these classes represent only a subset of the MILP space. To capture the full range of instances, tagging systems have been introduced [7], which assign multiple hierarchical tags to instances, from the most specific to the most general. However, this system does not create disjoint classes, which are necessary for a clean and meaningful structure. To overcome these limitations, a similarity metric between instances would allow for a formal characterization of their relationships, enabling the development of a comprehensive classification of the MILP space. Current similarity methods include the Image-Based Structural Similarity (ISS) approach [19], which treats constraint matrices as grayscale images analyzed by autoencoders. However, this method is sensitive to the ordering of constraints. Graph Neural Networks (GNNs) [20] address this by using invariant graph representations, but their applicability is limited to pre-trained problem classes, restricting their generalizability. The MIPLIB 2017 framework [7] offers a different approach by using over 100 handcrafted features in a high-dimensional feature space. However, it lacks a rigorous theoretical foundation for feature selection and normalization, which diminishes its robustness. A more comprehensive discussion of these methods can be found in section 2.

This paper presents a training-free mathematical distance for comparing MILP instances, designed to identify instances belonging to the same class as similar solely from their formulation. Variables, weights, and right-hand sides are categorized into discrete classes, allowing the definition of constraint distances based on mismatches in weight-variable pairs and right-hand sides. The overall MILP distance is then computed by matching constraints in a way that minimizes their distance, combined with the distance between objective

* Corresponding Author. Email: gwen.maudet@uni.lu
** Corresponding Author. Email: gregoire.danoy@uni.lu

functions. To accommodate instances of varying sizes, the method employs a normalized representation that captures the proportions of similar weight-variable pairs and constraints. By demonstrating that this metric aligns with the Earth Mover's Distance (EMD) [16], we establish that it satisfies the properties of a mathematical distance. For practical use, we introduce a greedy heuristic for efficient metric computation, ensuring seamless integrations with reduced computational cost. This is detailed in section 3.

We conduct a series of evaluations using the StrIPLIB library [3], which hierarchically organizes MILP instances into well-defined classes and subclasses. Two experiments are performed: the first replicates the protocol of [20] to assess the ability to identify instances from the same class (19 in total), while the second focuses on three selected classes to evaluate subclass-level grouping. In both cases, we measure performance by computing the proportion of nearest neighbors that belong to the same class (or subclass) as each test instance. We evaluate several variants of our distance metric, each omitting a specific component (e.g., weights, variables, right-hand sides, or objective function) to assess its contribution. Results show that each component plays a meaningful role in improving class identification. We also compare our greedy version with the exact one, observing a performance gap under 4% for 17 of the 19 classes, while achieving a 200× speedup. Compared to state-of-the-art similarity metrics [19, 20, 7], our method outperforms both the feature-based [7] and image-based [19] approaches, and matches the performance of the GNN-based method [20], despite requiring no supervised training. Full experimental details are provided in section 4.

Our contributions can be listed as follows:

- Identification of the challenge in structuring the MILP space, particularly pertinent in the context of ML, which can be addressed through a similarity metric.
- Introduction of a normalized representation of MILP instances, leveraging feature classification to enable meaningful comparisons across instances of varying sizes.
- Definition of a mathematical distance that captures structural patterns within and across constraints.
- Comprehensive experiments with new benchmarks and additional baselines, demonstrating our approach's efficiency.

2 Related Works and Positioning

In this section, we present the latest advancements in ML applied to MILP solvers, highlighting the current limitations with regard to generalizability. Next, we discuss the existing efforts for structuring the MILP space, starting with instance categorization methods and followed by similarity approaches between instances. Finally, we explain how our approach addresses the shortcomings of existing methods. We emphasize that our method has the potential to resolve the generalizability issue faced by ML-MILPs.

2.1 ML for MILP

ML has played a growing role in advancing MILP solvers by automating key components such as branching strategies, search heuristics, and cutting plane selection, leading to substantial performance improvements [10, 4, 5, 17, 27]. State-of-the-art on ML-MILPs, including those proposed in [8, 6, 9, 21, 12, 22, 15], are typically trained on synthetic datasets or well-established benchmarks limited to specific problem classes. Their generalization is usually evaluated by scaling instance size, rather than by assessing structural diversity. As a result, these models often fail to generalize effectively to heterogeneous or real-world instances. Although heterogeneous benchmarks such as MIPLIB 2017 [7] have been employed to assess the performance of ML-enhanced techniques—such as cutting plane generation or adaptive search policies [24, 11]—results have often been mixed due to challenges related to model convergence and the selection of suitable training sets.

At present, ML-MILP models generally require structurally homogeneous instance sets—i.e., instances drawn from the same underlying problem type, albeit with varying sizes. These models continue to struggle with generalization across the entire MILP space. A promising direction to address this limitation is to partition the MILP space into groups of structurally similar instances. Such structured partitioning would enable the design of specialized ML-MILP models tailored to each group, thereby improving generalizability.

Furthermore, most existing studies define their own evaluation sets, often without a principled assessment of heterogeneity. This practice complicates the comparison of different ML-MILP approaches in terms of their generalization capacity. Establishing a rigorous characterization of heterogeneity within evaluation datasets is thus essential for meaningful benchmarking and fair performance comparisons.

On these two grounds, we review existing approaches from two complementary angles: (i) methods for partitioning the MILP space and (ii) techniques for evaluating heterogeneity between instances.

2.2 Categorization of MILP Instances

Efforts to partition the MILP instance space have a long history. Early work focused on categorizing well-known problem domains in logistics and flow management [2, 14], which facilitated the development of specialized solution techniques targeted at specific problem types. More recently, StrIPLIB [3] has extended this by proposing a dataset of over 21,000 MILP instances drawn from the literature and organizing them into 33 hierarchically structured classes and subclasses. While this represents a significant step toward formalizing the MILP instance space, the classification framework exhibits several shortcomings. Many classes are not mutually exclusive; for example, the General Assignment problem reduces to a Knapsack problem when restricted to a single agent. Furthermore, the scope and granularity of the classes vary widely. Some categories, such as Bin Packing, encompass thousands of instances (e.g., 3,640), whereas others—like Binary/Ternary Code Construction—contain only a handful (e.g., 2). As a result, certain classes disproportionately dominate the dataset, while others cover only narrow and specialized niches, raising concerns about the structural balance and homogeneity of this existing classification appraoch. Additionally, this categorization is far from exhaustive. Real-world MILP problems often fall outside the boundaries of the predefined classes, rendering the current classification schemes insufficient for capturing the full diversity of the MILP space.

In parallel, MIPLIB 2017 [7], a benchmark library of real-world instances, employs a constraint tagging system to describe instance structures. These tags are linked to manually defined constraint templates, allowing for the characterization of instances directly from their formulations. While this approach enables broad coverage—any instance can be described using a combination of tags—the system does not support partitioning. Tags are inherently hierarchical, ranging from general to highly specific, and instances often receive multiple tags, precluding a clean division of the space into disjoint, homogeneous groups.

Despite these initiatives, a robust and comprehensive segmentation of the MILP space into disjoint, structurally homogeneous classes remains an open challenge. Achieving such a segmentation would likely require the development of a principled and reliable similarity metric capable of quantifying structural relationships between instances.

2.3 Similarity Measures Between MILPs

Various methods have been proposed to assess the similarity between MILP instances, either to guide solvers in the resolution process or to evaluate the heterogeneity of instance sets. For the first objective, Steever *et al.* explored similarity methods, emphasizing that the structure of a MILP can reveal valuable insights into its solution process without more details. Their initial approach, the ISS method [19], represents instances by encoding constraint matrices as grayscale images analyzed via autoencoders, producing a lower-dimensional representation of the input instance. Subsequently, they introduced a GNN-based approach [20], where an input graph representation of an instance is processed to output the probability of the instance belonging to each of the trained classes, using training data. The second objective, evaluating heterogeneity, was pursued in the construction of the MIPLIB2017 library [7], where their similarity metric was employed to define instance libraries that maximize heterogeneity. This metric represents instances through 100 handcrafted features, including attributes such as problem size, coefficient distributions, and categorization tags. In both methods, the similarity between instances is measured using the Euclidean distance in the output space.

Despite these contributions, existing similarity measures face important limitations. Unsupervised approaches such as the feature-based method lack rigorous theoretical justification for feature selection, which limits confidence in their applicability across diverse MILP formulations. The ISS method has been shown to underperform compared to the GNN-based alternative. On the other hand, the supervised GNN model relies on predefined instance classes during training—a framework that lacks theoretical coherence for defining a global, principled structure over the MILP space. This reliance introduces significant difficulties in selecting appropriate training sets, especially when aiming to generalize to the full diversity of MILP problems.

In summary, existing similarity metrics remain constrained either by poor performance for unsupervised appraoch or by the need for supervised training on ill-defined class labels, motivating the search for alternative, mathematically grounded approaches that can support unsupervised structuring of the MILP space.

2.4 Positioning

To overcome the limitations identified in previous approaches, we propose a lightweight similarity metric for MILP instances that is directly derived from their mathematical formulations and supported by well-defined mathematical properties. Instead of relying on predefined, discrete problem classes, our approach generalizes this concept by defining similarity as a continuous measure of structural closeness. Within a given problem class—assuming a consistent representation—strong structural patterns tend to persist across instances, even as instance sizes vary. That is, larger instances may introduce additional variables or constraints, yet the fundamental structure (e.g., variable types, constraint templates, and coefficient distributions) typically remains stable. The goal is thus to identify these persistent structural similarities that are preserved as instance size increases.

The proposed metric not only facilitates the selection of evaluation sets by controlling heterogeneity—similar to the motivation in [7]—but also enables the quantitative assessment of heterogeneity in existing benchmarks. This, in turn, provides a principled means for evaluating the generalizability of optimization and learning methods. Moreover, as noted by Steever *et al.*, such similarity metrics can be leveraged to guide solver behavior, enhancing performance. From an ML perspective, a reliable similarity metric allows for the segmentation of the MILP space into structurally coherent subsets. This enables training ML-MILP models on homogeneous instance groups, which improves convergence and mitigates the difficulties associated with training set selection. In contrast to handcrafted heuristics—often tailored to specific problem families—ML methods are inherently data-driven and adapt to the structure of the training data. As such, defining similarity directly from instance formulations, rather than relying on problem-type semantics, presents a promising approach for developing ML-MILP methods that are both scalable and broadly applicable.

Nevertheless, the task of explicitly partitioning the MILP space—though a natural extension of our metric—is beyond the scope of this paper. Our primary goal here is to validate the relevance and robustness of the proposed distance measure itself.

3 Formal Definition of the Distance

In this section, we formally define a distance metric for comparing MILP instances, with the goal of identifying structurally similar instances—even when their sizes differ—by leveraging common patterns typically observed within instances of the same problem class. The metric is constructed by categorizing variables, their corresponding weights, and right-hand sides, followed by introducing a normalized instance representation aimed at eliminating dimension-dependent parameters. Using this representation, we define a distance between individual constraints and extend it to entire MILP instances by comparing proportions of similar weight-variable pairs and constraints, rather than direct pairwise comparisons. This metric satisfies the properties of a mathematical distance. having similarities with the well known EMD.

3.1 Background

A MILP instance $\mathbf{P}$ can be expressed mathematically as (1):

$$\begin{aligned} &\text{Minimize } z = \sum_{j \in \mathbb{N}_0} w_0^j v^j, \\ &\text{Subject to} \quad i \in \{1, \ldots, m\},\ c_i : \sum_{j \in \mathbb{N}_i} w_i^j v^j \leq b_i, \end{aligned} \tag{1}$$

where the key components are defined as follows:

- $(v^j)_{1 \leq j \leq n}$: the set of variables in the instance.
- $C = (c_i)_{1 \leq i \leq m}$: the set of constraints, with c_0 conventionally denoting the objective function. For each constraint c_i, $\mathbb{N}_i \subseteq \{1, \ldots, n\}$ represents the indices of variables with non-zero weights, with $n_i = |\mathbb{N}_i|$ the number of variables in the constraint.
- $b_i \in \mathbb{R}$: the right-hand side of constraint c_i.
- $(v^j)_{j \in \mathbb{N}_i}$: the variables associated with constraint c_i.
- $(w_i^j)_{j \in \mathbb{N}_i}$: the weights associated to the variables in c_i.

For clarity, when necessary, we denote $c_i(\mathbf{P})$ to explicitly refer to constraint (or objective function) c_i associated with instance $\mathbf{P}$.

3.2 Classification of MILP Features

To define the distance between constraints—and by extension, entire MILP instances—we classify the three essential features of MILP formulations: right-hand sides, variables, and weights. This classification discretizes these features into distinct classes, enabling simplified comparisons through binary similarity measures, while also accounting for redundancies both within and across constraints.

Formalism of MILP Feature Classes

The classification process assigns classes to elements as follows:

- Each right-hand side b_i is assigned a class $t(b_i) \in \mathcal{C}(b)$.
- Each variable v^j is assigned a class $t(v^j) \in \mathcal{C}(v)$.
- Each weight w_i^j is assigned a class $t(w_i^j) \in \mathcal{C}(w)$.

To define the distance, we use the indicator function $\mathbb{1}$, which compares two elements x_1 and x_2 of type x (right-hand sides, variables, or weights). If x_1 and x_2 are in the same class, $\mathbb{1}(t(x_1) \neq t(x_2)) = 0$; otherwise, it equals 1.

Choices Made for the Classification

For variables, we use the standard classification into three categories: binary (B): $v^j \in \{0, 1\}$, integer (I): $v^j \in \mathbb{Z}$, and continuous (C): : $v^j \in \mathbb{R}$. Constraints on lower and upper bounds are excluded to maintain a reduced number of classes. For weights and right-hand sides, the domains are $\mathbb{R}$ and $\mathbb{R}^*$, respectively. To define classes, instead of defining intervals, which may introduce boundary challenges, we classify by isolating the most frequently occurring singletons as distinct classes. The remaining values form a complementary class encompassing all other elements. The classification is derived from the MIPLIB 2017 collection, a comprehensive dataset of 1065 MILP instances [7] representing diverse MILP problems. Analysis of this dataset identifies -1 (40%) and 1 (33%) as the most frequent singletons for weights, and 0 (57%) and 1 (21%) for right-hand sides, with high occurrence rates validating the singleton-based approach. Similar patterns are observed in the strIPLIB instances used for evaluation. In summary, the classification scheme is as follows:

- $\mathcal{C}(v) = \{B, I, C\}$ for variables,
- $\mathcal{C}(w) = \{-1, 1, \mathbb{R} \setminus \{-1, 1\}\}$ for weights,
- $\mathcal{C}(b) = \{0, 1, \mathbb{R} \setminus \{0, 1\}\}$ for right-hand sides.

This discretization scheme is motivated by principles from entropy-based discretization [26], aiming to preserve the most informative distinctions in the data. Additionally, by using the same number of classes across different features and ensuring balanced class proportions, we improve comparability of feature importances and prevent any single feature type from dominating the similarity computation.

3.3 Normalized Representation of an Instance

From this classification, redundancies are identified both within constraints, in terms of repeated weight-variable pairs, and across constraints with identical constraint representations. To capture repetitions of weight-variable pairs, let $n(\hat{w}_j, \hat{v}_k, i)$ represent the number of occurrences of the pair $\hat{w}_j, \hat{v}_k$ in constraint c_i, and define their proportion as $p_c(\hat{w}_j, \hat{v}_k, i) = \frac{n(\hat{w}_j, \hat{v}_k, i)}{n_i}$. Similarly, redundancies between structurally identical constraints are addressed by identifying the set of structurally unique constraints $(c_i)_{i \in \mathbb{M}(\mathbf{P})}$, where $\mathbb{M}(\mathbf{P}) \subseteq \{1, \ldots, m\}$, and for $i \in \mathbb{M}(\mathbf{P})$, m_i denotes the number of repetitions of constraint c_i. The proportion of a constraint c_i within the instance $\mathbf{P}$ is given by $p_{\mathbf{P}}(\mathbf{P}, c_i) = \frac{m_i}{m}$. Thus, we can define a normalized representation of an instance by removing the dimensions induced by the instance itself, representing the proportions of occurrence of weight-variable pairs within a constraint and the proportions of occurrence of constraints within the instance. Using the notation $\sum_{\hat{w}_j, \hat{v}_k} = \sum_{\hat{w}_j \in \mathcal{C}(w)} \sum_{\hat{v}_k \in \mathcal{C}(v)}$, we define the normalized version of an instance $\mathbf{P}$ as in (2):

$$\begin{aligned} &\text{Minimize } z = \sum_{\hat{w}_j, \hat{v}_k} p_c(\hat{w}_j, \hat{v}_k, 0) \hat{w}_j \hat{v}_k, \\ &\text{Subject to} \quad i \in \mathbb{M}(\mathbf{P}), \\ &p_{\mathbf{P}}(\mathbf{P}, i) \times c_i : \sum_{\hat{w}_j, \hat{v}_k} p_c(\hat{w}_j, \hat{v}_k, i) \hat{w}_j \hat{v}_k \leq t(b_i). \end{aligned} \tag{2}$$

Table 1. Normalized representation of the *app1-2* instance. The first row displays the objective function to minimize, followed by the list of constraints. The first column represents the proportional occurrence (denoted as e$x = 10^x$) of each constraint type, and the second column shows the constraint representation. In each constraint, a weight-variable pair is denoted as *the proportion of that pair* × *the weight class* · *the variable class*. The class $\mathbb{R}$ represents the class excluding singletons for both the weight and the right-hand side.

Minimize $1.0 \times -1 \cdot \mathrm{B}$

Prop	Constraint Representation
2.0e-1	$0.33 \times \mathbb{R} \cdot \mathrm{B} + 0.33 \times 1 \cdot \mathrm{C} + 0.33 \times -1 \cdot \mathrm{C} \leq \mathbb{R}$
2.0e-1	$0.33 \times 1 \cdot \mathrm{B} + 0.33 \times -1 \cdot \mathrm{C} + 0.33 \times 1 \cdot \mathrm{C} \leq 1$
2.0e-1	$0.5 \times 1 \cdot \mathrm{C} + 0.5 \times -1 \cdot \mathrm{C} \leq 0$
1.9e-1	$0.83 \times \mathbb{R} \cdot \mathrm{C} + 0.17 \times -1 \cdot \mathrm{C} \leq 0$
1.9e-1	$0.83 \times \mathbb{R} \cdot \mathrm{C} + 0.17 \times 1 \cdot \mathrm{C} \leq 0$
4.5e-3	$0.8 \times \mathbb{R} \cdot \mathrm{C} + 0.2 \times -1 \cdot \mathrm{C} \leq 0$
4.5e-3	$0.8 \times \mathbb{R} \cdot \mathrm{C} + 0.2 \times 1 \cdot \mathrm{C} \leq 0$
4.0e-3	$1.0 \times -1 \cdot \mathrm{B} \leq \mathbb{R}$
2.1e-3	$0.33 \times \mathbb{R} \cdot \mathrm{B} + 0.33 \times -1 \cdot \mathrm{C} + 0.33 \times 1 \cdot \mathrm{C} \leq \mathbb{R}$
2.1e-3	$0.33 \times 1 \cdot \mathrm{B} + 0.33 \times 1 \cdot \mathrm{C} + 0.33 \times -1 \cdot \mathrm{C} \leq 1$
2.1e-3	$0.5 \times -1 \cdot \mathrm{C} + 0.5 \times 1 \cdot \mathrm{C} \leq 0$
1.5e-5	$1.0 \times -1 \cdot \mathrm{C} \leq \mathbb{R}$
1.5e-5	$1.0 \times 1 \cdot \mathrm{C} \leq 1$

An example is provided in Table 1, where the normalized version of the instance *app1-2* from MIPLIB 2017 is illustrated (https://MIPlib.zib.de/instance_details_app1-2.html). This instance consists of 53,467 constraints[1] and 26,871 variables, yet only 13 unique constraint representations, each containing at most three distinct types of weight-variable pairs.

It is worth noting that a strong similarity can be drawn with the MIPLIB 2017 tagging process, which constructs manually defined constraint templates [2]. Our standardization approach enables the definition of these templates while also generalizing the process through the automatic definition of templates that compactly represent the instance.

[1] Additional constraints were introduced in our model to convert equality constraints into two inequality constraints.

[2] See https://MIPlib.zib.de/statistics.html

3.4 Distance Definitions

Our objective is to define a distance measure between MILP instances that effectively groups instances from the same problem class, even when their sizes differ. Specifically, by analyzing modifications of similar problem types of varying sizes, we aim to assign zero distance to: (i) constraints that vary in the number of variables but maintain proportionality in the distribution of weight-variable pairs; (ii) instances with different numbers of constraints but the same proportion of similar constraints. Our distance measure is based on the normalized representation in (2), where we aim to minimize mismatches in the weight-variable pairs for constraint comparison, and minimize the distances between constraints to determine the overall distance between instances.

Distance between Weight-Variable Pairs

First, to compare a weight-variable pair, we establish the following rules: (i) if the weights belong to different classes, we add $\alpha > 0$; (ii) if the variables belong to different classes, we add $\beta > 0$. Therefore, for a pair of weight and variable $\hat{w}_j, \hat{v}_k$ and another pair $\hat{w}_l, \hat{v}_o$, their similarity is expressed as (3):

$$d_{w,v}(\hat{w}_j, \hat{v}_k, \hat{w}_l, \hat{v}_o) = \alpha \mathbb{1}(\hat{w}_j \neq \hat{w}_l) + \beta \mathbb{1}(\hat{v}_k \neq \hat{v}_o) \quad (3)$$

$d_{w,v}$ represents the discrete distance metric, it satisfies the properties of a mathematical distance.

Theorem 1. *The function $d_{w,v}$ defined in* (3) *is a mathematical distance between elements of $\mathcal{C}(w) \times \mathcal{C}(v)$.*

Distance Between Constraints

The distance between two constraints, c_q and c_r, is defined as the minimal transfer of proportions of weight-variable pairs from c_q to c_r. This transfer is weighted by $d_{w,v}$, and it includes an additional term accounting for potential differences in their right-hand sides, weighted by γ. Considering the proportions of weight-variable pair occurrences $(p(\hat{w}_j, \hat{v}_k, q))_{\hat{w}_j, \hat{v}_k}$ for c_q and $(p(\hat{w}_l, \hat{v}_o, r))_{\hat{w}_l, \hat{v}_o}$ for c_r, the simiarity is expressed mathematically as (4):

$$\begin{aligned} &d_c(c_q, c_r) = \\ &\min \sum_{\hat{w}_j, \hat{v}_k} \sum_{\hat{w}_l, \hat{v}_o} d_{w,v}(\hat{w}_j, \hat{v}_k, \hat{w}_l, \hat{v}_o) \cdot f(\hat{w}_j, \hat{v}_k, \hat{w}_l, \hat{v}_o) \\ &+ \gamma \mathbb{1}(t(b_q) \neq t(b_r)), \end{aligned} \quad (4)$$

subject to the following constraints given in (4.1):

$$\begin{aligned} &\forall \hat{w}_j, \hat{v}_k, \hat{w}_l, \hat{v}_o, f(\hat{w}_j, \hat{v}_k, \hat{w}_l, \hat{v}_o) \geq 0, \\ &\forall \hat{w}_j, \hat{v}_k, \sum_{\hat{w}_l, \hat{v}_o} f(\hat{w}_j, \hat{v}_k, \hat{w}_l, \hat{v}_o) = p_c(\hat{w}_j, \hat{v}_k, q), \\ &\forall \hat{w}_l, \hat{v}_o, \sum_{\hat{w}_j, \hat{v}_k} f(\hat{w}_j, \hat{v}_k, \hat{w}_l, \hat{v}_o) = p_c(\hat{w}_l, \hat{v}_o, r). \end{aligned} \quad (4.1)$$

Here, $f(\hat{w}_j, \hat{v}_k, \hat{w}_l, \hat{v}_o)$ represents the transfer of proportions from the weight-variable pair $\hat{w}_j, \hat{v}_k$ in c_q to the pair $\hat{w}_l, \hat{v}_o$ in c_r. The constraints ensure that the total transfer from all pairs $\hat{w}_j, \hat{v}_k$ equals the proportion $p_c(\hat{w}_j, \hat{v}_k, q)$, and similarly, the total transfer to $\hat{w}_l, \hat{v}_o$ equals $p_c(\hat{w}_l, \hat{v}_o, r)$. Notably, this corresponds to the EMD [16], also commonly referred to as the Wasserstein distance [25]. This distance quantifies the minimum amount of change required to transition from one distribution to another. Combined with an additional distance component, this metric satisfies the properties of a mathematical distance.

Theorem 2. *The function d_c defined in* (4) *is a mathematical distance between constraints represented as* (2)*.*

Distance Between Instances

The distance between two instances is defined using the previously established constraint distance. For instances $\mathbf{P_s}$ and $\mathbf{P_t}$, the set $(c_{i_s}(\mathbf{P_s}), p_{\mathbf{P}}(\mathbf{P_s}, c_{i_s}(\mathbf{P_s}))_{i_s \in \mathbb{M}(\mathbf{P_s})}$ represents distinct constraints and their proportions for $\mathbf{P_s}$, with a similar representation for $\mathbf{P_t}$. The distance is calculated by transferring the proportions of constraints from $\mathbf{P_s}$ to $\mathbf{P_t}$, weighted by d_c, and including an additional term weighted by $\zeta > 0$ to account for the objective function distance, as shown in (5):

$$\begin{aligned} &d_{\mathbf{P}}(\mathbf{P_s}, \mathbf{P_t}) = \\ &\min \sum_{i_s \in \mathbb{M}(\mathbf{P_s})} \sum_{i_t \in \mathbb{M}(\mathbf{P_t})} d_c(c_{i_s}(\mathbf{P_s}), c_{i_t}(\mathbf{P_t})) \cdot f(i_s, i_t) \\ &+ \zeta d_c(c_0(\mathbf{P_s}), c_0(\mathbf{P_t})), \end{aligned} \quad (5)$$

subject to the following constraints given in (5.1):

$$\begin{aligned} &\forall i_s \in \mathbb{M}(\mathbf{P_s}), \forall i_t \in \mathbb{M}(\mathbf{P_t}), f(i_s, i_t) \geq 0, \\ &\forall i_s \in \mathbb{M}(\mathbf{P_s}), \sum_{i_t \in \mathbb{M}(\mathbf{P_t})} f(i_s, i_t) = p_{\mathbf{P}}(\mathbf{P_s}, c_{i_s}(\mathbf{P_s})), \\ &\forall i_t \in \mathbb{M}(\mathbf{P_t}), \sum_{i_s \in \mathbb{M}(\mathbf{P_s})} f(i_s, i_t) = p_{\mathbf{P}}(P_t, c_{i_t}(\mathbf{P_t})). \end{aligned} \quad (5.1)$$

Similarly to the distance between constraints, $f(i_s, i_t)$ represents the transfer of proportions from the constraint $c_{i_s}(\mathbf{P_s})$ to $c_{i_t}(\mathbf{P_t})$, with constraints ensuring that the transfer respects the proportions $p_{\mathbf{P}}(\mathbf{P_s}, i_s)$ and $p_{\mathbf{P}}(\mathbf{P_t}, i_t)$. This first term corresponds to the EMD, ensuring that, when combined with another distance, $d_{\mathbf{P}}$ adheres to the properties of a mathematical distance.

Theorem 3. *The function $d_{\mathbf{P}}$ defined in* (5) *is a mathematical distance between instances represented as* (2)*.*

This metric enables the formal comparison of MILP instances, supports defining neighborhoods of instances, and facilitates classification methods that rely on a valid distance measure. Our goal was to design a distance metric capable of identifying as similar the instances belonging to the same type, even when their sizes differ. Specifically, when identical constraints (i.e., constraints with similar proportions of weight-variable pairs) appear in the same proportions across the compared instances, the distance is $d_{\mathbf{P}} = 0$. In more general cases, the proposed distance quantifies the minimum proportion of changes necessary to transform one instance into another, based on the normalized representation of (2). In particular, a change in weight incurs a penalty of α, a change in variables incurs a penalty of β, a modification to the right-hand side incurs a penalty of γ, and an additional weighting factor ζ is applied to account for differences in the objective functions. Still, a known limitation of this formulation is its flexibility with respect to dimensionality: for example, a constraint involving a single variable can have zero distance from one in which that variable appears multiple times, even though their semantics may differ significantly.

3.5 Greedy Heuristic for Distance Computation

The exact computation of the EMD has a complexity of $O(n^3 \log(n))$ [13], which presents a significant computational challenge due to its application in both constraint-level and instance-level distance calculations. To address this, we propose an alternative greedy heuristic that iteratively matches the closest pairs, first at the constraint level and subsequently at the instance level, offering a computationally efficient and straightforward approach. At the constraint level, weight-variable pairs from one constraint are iteratively matched with their closest counterparts (based on $d_{w,v}$), with matching proportions determined by the smaller of the two proportions. A constraint contains at most $|\mathcal{C}(w)| \times |\mathcal{C}(v)|$ unique pairs, where $|\mathcal{C}(\cdot)|$ denotes the number of elements in the set. Thus, comparing two constraints has a complexity of $O(|\mathcal{C}(w)|^2 \times |\mathcal{C}(v)|^2)$. Extending this to compare all constraints of instances $\mathbf{P_s}$ and $\mathbf{P_t}$, the total complexity for constraint-level comparisons becomes $O(|\mathcal{C}(w)|^2 \times |\mathcal{C}(v)|^2 \times |\mathbb{M}(\mathbf{P_s})| \times |\mathbb{M}(\mathbf{P_t})|)$, where $|\mathbb{M}(\cdot)|$ represents the number of distinct constraints in an instance. At the instance level, constraints from $\mathbf{P_s}$ are iteratively matched with their closest counterparts in $\mathbf{P_t}$ (based on d_c), again matching proportions according to the smaller value. This step adds a complexity of $O(|\mathbb{M}(\mathbf{P_s})| \times |\mathbb{M}(\mathbf{P_t})|)$, which remains negligible when compared to the constraint distance computation.

4 Evaluation

The main objective of our experimental study is to evaluate the ability of similarity measures to group structurally related MILP instances. While problem classification is not without limitations, instances within the same class typically share key structural traits—which is precisely what we aim to capture. We leverage this principle by using the strIPLIB dataset [3], which contains over 21,000 MILP instances organized into hierarchically structured classes and subclasses. We deliberately choose not to use MIPLIB 2017 [7], as its real-world instances often span multiple problem types, making class-based evaluation less suitable. We recall that some modeling choices in our distance formulation—such as the categorization of weights and right-hand sides—are derived from empirical observations on MIPLIB instances.

Two experiments are conducted to evaluate the performance of our similarity metric. The first replicates the methodology proposed in [20], which assesses the capability to identify instances belonging to the same class. The second extends this analysis by examining the ability to group instances within the same subclass for classes already studied. All source code and resources are publicly available on the Git repository at https://gitlab.com/uniluxembourg/snt/pcog/ultrabo/clustering-for-search-strategy.

For each experiment, we sample 50 instances per class (or subclass), denoted by $c \in \mathcal{C}$, where $\mathcal{C}$ is the set of all considered classes. Each class provides 10 test instances, written as $\mathbf{P}_c^i(t)$ for $i \in [1, 10]$, and 40 reference instances, denoted $\mathbf{P}_c^j(r)$ for $j \in [1, 40]$. Each test instance $\mathbf{P}_c^i(t)$ is compared against all reference instances $\mathbf{P}_{\acute{c}}^j(r)$ from every class $\acute{c} \in \mathcal{C}$. The 40 reference instances with the smallest distances $\text{TOP40}\big(d(\mathbf{P}_c^i(t), \mathbf{P}_{\acute{c}}^j(r))\big)$ are selected. The *top-40 accuracy* is defined as the proportion of these 40 nearest neighbors that belong to the same class c. The final score for each class is the average top-40 accuracy over its 10 test instances. It is important to note that each added class contributes 40 additional comparison points per test instance. As such, performance is influenced by the total number of (sub)classes considered, and results are only comparable within the same experiment, not across experiments. To enhance clarity, the highest score for a class (or subclass) among baselines is highlighted in green in the result tables. Conversely, if the performance is at least 50% worse than the best score, the corresponding cell is marked in red, with intermediate values represented by a gradient transitioning between these colors.

Our method, denoted *Formal*, depends on parameters α, β, γ and ζ, which respectively control the influence of weights, variables, right-hand sides and the objective function in the distance computation. In the first experiment, we conduct a sensitivity analysis: we test configurations where one parameter is deactivated (denoted with a $\not$ symbol), and compare these to the baseline setting where all parameters are set to 1 (denoted $\emptyset$). All sensitivity variants use the greedy version of the distance computation for efficiency. We also report results from the exact version of our metric (denoted $\emptyset E$), and include computation times for both greedy and exact variants. For the second experiment, we restrict our comparison to the greedy version with $\alpha = \beta = \gamma = \zeta = 1$.

We compare our proposed metric against the three main similarity measures available in the literature—these are, to the best of our knowledge, the only established methods for assessing similarity between MILP instances:

- **Features-Based Method** [7]: This method represents an instance in a 100-dimensional space by extracting features. The similarity between two instances is calculated as the Euclidean distance within this feature space. We denote this method as *Feat*. The feature extraction code is publicly available [3], and we utilized this implementation for all experiments. Notably, a direct comparison of this method to other baselines has not been conducted in previous works.
- **ISS** [19]: In this approach, instances are converted into grayscale images, which are subsequently fed into an autoencoder. The similarity between instances is computed as the Euclidean distance between the autoencoder's output vectors. We denote this method as *ISS*. Since the authors did not provide an available implementation, we relied on the reported results from the original paper for the first part of our experiments. However, this method is excluded from the second part of our analysis due to the unavailability of an open-source implementation.
- **GNN Similarity** [20]: This method leverages a GNN trained to classify instances into 19 predefined categories, using the reference set of 40 instances per class from the first experiment. The similarity between instances is defined as the Euclidean distance between the output embeddings of the GNN. We denote this method as *GNN*. Although the official repository provides a pre-trained model, it does not include the necessary code for constructing or training the GNN. Consequently, we rely on the reported results from the corresponding paper for experiments under the same framework and utilize the pre-trained GNN model for additional experiments.

4.1 Identifying Membership to Similar Classes

The primary objective of this evaluation is to assess the ability of our method to effectively group instances belonging to the same class. To ensure a fair comparison, we adopt the same experimental framework described in [20], leveraging their specified test and reference sets. This approach allows for direct performance comparisons with

[3] https://MIPlib.zib.de/downloads/feature_extractor.zip

the *ISS* and *GNN* methods by using the data provided in [20]. Additionally, we evaluate the performance of the *Features* method. The experiment considers 19 problem classes from the strIPLIB dataset. The abbreviations used for these classes are detailed in [20, 19]. These classes were justified in [19] as follows: *"19 problem types which contained at least 50 total instances across all sources for that problem type"*. However, the specific selection process for these 50 instances per class is not detailed. It is also noteworthy that, since the release of the library, 6 additional problem classes now contain more than 50 instances. Furthermore, some previously listed classes, such as *bpp*, *bp2*, and *bif*, have since been reclassified as subclasses.

	Formal						Feat	ISS	GNN
	$\not\alpha$	$\not\beta$	$\not\gamma$	$\not\zeta$	$\emptyset E$	$\emptyset$			
bpp	42	47	49	26	47	47	27	32	81
bp2	62	71	100	71	95	95	24	52	100
bif	50	44	50	50	50	50	21	32	43
clp	54	59	37	64	57	56	28	12	37
col	40	40	40	40	40	40	27	36	40
cpm	49	54	51	52	65	54	32	47	100
cut	26	48	57	34	49	48	20	16	99
cvr	82	62	87	92	91	91	15	40	93
cwl	100	97	95	97	100	97	100	23	100
gap	100	100	98	98	100	100	22	30	100
inr	56	61	55	84	63	63	88	19	84
kps	100	100	100	100	100	100	35	82	100
lot	48	57	72	49	68	56	36	28	87
map	55	65	67	42	62	65	33	27	97
pcp	59	87	91	87	83	87	33	57	100
rel	28	38	31	27	38	38	18	22	25
sch	100	100	100	99	100	100	45	24	87
tup	97	100	100	93	100	100	64	75	100
vrp	100	100	100	100	100	100	56	100	100
mean	66	70	73	69	74	73	38	40	83

Table 2. Evaluation of similarity measures to identify whether instances belong to the same problem class.

The experimental results are summarized in Table 2. Focusing first on the different variants of our method *Formal*, we observe that removing any component of the distance (by setting its weight to zero) generally leads to a drop in performance. The full version ($\emptyset$) consistently outperforms both $\not\alpha$ and $\not\zeta$—the latter in all but one class—and is marginally superior to $\not\beta$ in all but three classes. The variant without right-hand side comparison ($\not\gamma$) yields results that are, on average, similar to the full version, but shows a significantly higher worst-case drop (up to 19%), indicating greater variability and instability. When comparing the greedy version ($\emptyset$) to the exact formulation ($\emptyset E$), results are identical in 17 out of 19 classes. The average runtime to compute the top-40 neighbors per test instance is 2.8×10^{-3} seconds (std 4.4×10^{-3}) for the greedy version, compared to 0.37 seconds (std 0.65) for the exact method. This highlights not only a 200× speedup but also a much more stable runtime, making the greedy variant clearly preferable in practice. In comparison to the baselines, both our method (*Formal*) and the supervised *GNN* approach consistently outperform *Features* and *ISS*. The latter two perform at least 50% worse than the best score in 12 out of the 19 evaluated classes. While *GNN* benefits from supervised training on nearly 800 labeled instances from similar classes, the performance gap between the two remains small. *Formal* achieves results within 5% of the best score in 12 out of 19 classes, compared to 14 out of 19 for *GNN*, with similarly close average accuracy.

Regarding class-wise performance, more than ten classes exhibit near-perfect top-40 accuracy above 95%, while others perform significantly worse. Further analysis (not shown) highlights notable structural overlaps between certain classes. For instance, at least 10% of the top 40 most similar instances retrieved by *Formal* in some cases belong to a different class, and vice versa. Notably, *bpp* and *cut* are explicitly marked as structurally similar in the StrIPLIB documentation. Likewise, *cpm* and *cwl* both fall under capacity-constrained formulations, and *map* and *col* share similar structures despite being distinct classes. The *rel* class also performs poorly across all metrics, likely due to its high internal heterogeneity and overlap with the *bif* class. These overlaps appear to result from the inherently ambiguous nature of instance class definitions, which often blur structural boundaries between problems.

4.2 *Identifying Belongings to Similar Subclasses*

		Formal	Feat	GNN
bpp	conflicts	0.51	0.32	0.5
	item-fragmentation	0.46	0.37	0.89
	plain	0.99	0.28	1.0
	two-dimensional	0.84	0.28	1.0
lot	linked-lot-sizes	1.0	0.31	0.48
	sizing multi-level	0.52	0.47	0.6
	sizing multi-level-with-setup-time	0.42	0.33	0.45
	sizing single-level	1.0	0.47	0.46
vrp	capacitated	0.96	0.21	1.0
	concrete-delivery	0.73	0.5	1.0
	time-windows	1.0	0.38	1.0
mean		77	36	76

Table 3. Evaluation of similarity measures to identify whether instances belong to the same subclass.

The objective of this second part of the simulation is analogous to the first; however, it focuses on problem subclasses, making it intrinsically more complex as it requires distinguishing between sets of more closely related instances. To maintain consistency within the training framework of the *GNN* method and to ensure a fair comparison, the subclasses studied belong exclusively to the classes trained by *GNN*. We select the set of subclasses based on the following criteria: (i) the class must be part of the *GNN* reference set; (ii) the subclass must contain at least 50 instances; and (iii) there must be more than one eligible subclass per class. This selection yields three classes, each with three to four subclasses. Notably, three of the four subclasses within the *bpp* class were explicitly trained as separate classes by the *GNN*, potentially giving it a specific advantage. The instances are selected using the seed $\{1\}$ to ensure reproducibility.

The results of this experiment are presented in Table 3. Once again, the *Features* method performs significantly worse compared to both *Formal* and *GNN*. However, it remains challenging to definitively determine superiority between *Formal* and *GNN*. For the *bpp* class, *GNN* outperforms *Formal* by 48% and 16% on two of the subclasses, respectively, with both methods achieving similar results on the remaining two subclasses, although *GNN* was specifically trained to recognize these subclasses. For the *lot* class, each method excels in two of the four subclasses. However, while *GNN* underperforms by at least 52%, *Formal*'s performance is never more than 6% below the best, demonstrating greater reliability across these subclasses. For the *vrp* class, *GNN* achieves perfect classification, while *Formal* closely follows, with an average gap of approximately 10%. Overall, both methods reach comparable mean performance, with *Formal* slightly ahead by 1%.

To summarize these experiments, *Formal* significantly outperforms all unsupervised baselines (*ISS* and *Features*). When compared to *GNN*, neither method clearly surpasses the other. However, *GNN* benefits from supervised training on this specific set of problem classes using a labeled dataset of 760 instances. These problem classes have inherent limitations, and *GNN* is confined to its training framework, making the generalizability of its similarity metric across the entire MILP space impractical.

5 Conclusion and Future Works

In this paper, we introduced a novel similarity metric for MILP that exhibits the mathematical properties of a distance function. This metric is derived solely from the intrinsic data of the problems being compared, without requiring any training process. We demonstrated that our proposed metric outperforms the considered baselines, particularly challenging a classification method that relies on a labeled training dataset.

Our proposed similarity metric offers several advantages. First, it can effectively characterize the heterogeneity within a set of instances, making it applicable to various use cases, such as selecting heterogeneous benchmark instances (e.g., MIPLIB) or quantifying the diversity of any evaluation set. Second, it defines instance classes based solely on the structural representation of constraints, without needing to consider the specific objective of the instances. This approach not only bridges existing instance classes but also enables the association of unlabeled or sparsely labeled instances with known groups.

Future work will focus on extending this methodology to develop a classification of the entire MILP space, with the aim of generalizing ML-MILP methods across all MILP instances. As highlighted in previous studies, ML methods for MILP solvers (e.g., branching, searching, cutting) typically perform well only on structurally similar instances. A comprehensive classification of the MILP space will facilitate the development of specialized ML-MILP models targeting specific instance classes, improving their overall effectiveness.

Acknowledgements

This research was supported by the Agence Nationale de la Recherche (grant ANR-22-CE46-0011) and the Luxembourg National Research Fund (grant INTER/ANR/22/17133848) through the UltraBO Project. Computational experiments were carried out using the High-Performance Computing (HPC) facilities of the University of Luxembourg.

We also thank Mark Turner for insightful discussions, Pierre Talbot for his unwavering support, and Louis Gass as well as Simon Lemal for their assistance with the mathematical formulation of the problem.

References

[1] C. Archetti, L. Peirano, and M. G. Speranza. Optimization in multimodal freight transportation problems: A Survey. *European Journal of Operational Research*, 299(1):1–20, May 2022. ISSN 0377-2217. doi: 10.1016/j.ejor.2021.07.031. URL https://www.sciencedirect.com/science/article/pii/S0377221721006263.

[2] G. Ausiello, A. D'Atri, and M. Protasi. On the structure of combinatorial problems and structure preserving reductions. In A. Salomaa and M. Steinby, editors, *Automata, Languages and Programming*, pages 45–60, Berlin, Heidelberg, 1977. Springer. ISBN 978-3-540-37305-6. doi: 10.1007/3-540-08342-1_4.

[3] M. Bastubbe, A. Helber, L. Kirchhart, M. L{\"u}bbecke, N. Rieken, and J. Witt. strIPlib: Structured Integer Programming Library, 2025. URL https://striplib.or.rwth-aachen.de.

[4] Y. Bengio, A. Lodi, and A. Prouvost. Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon. *European Journal of Operational Research*, 2021. URL http://arxiv.org/abs/1811.06128. arXiv:1811.06128 [cs, stat].

[5] Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris, and P. Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of machine learning research*, 2023. URL http://arxiv.org/abs/2102.09544. arXiv:2102.09544 [cs, math, stat].

[6] M. Gasse, D. Chetelat, N. Ferroni, L. Charlin, and A. Lodi. Exact Combinatorial Optimization with Graph Convolutional Neural Networks. In *NeurIPS*, 2019.

[7] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. Christophel, K. Jarck, T. Koch, J. Linderoth, M. Lübbecke, H. D. Mittelmann, D. Ozyurt, T. K. Ralphs, D. Salvagnin, and Y. Shinano. MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, Sept. 2021. ISSN 1867-2957. doi: 10.1007/s12532-020-00194-3. URL https://doi.org/10.1007/s12532-020-00194-3.

[8] H. He, H. Daume III, and J. M. Eisner. Learning to Search in Branch and Bound Algorithms. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/hash/757f843a169cc678064d9530d12a1881-Abstract.html.

[9] A. G. Labassi, D. Chételat, and A. Lodi. Learning to Compare Nodes in Branch and Bound with Graph Neural Networks. *NeurIPS*, 2022.

[10] A. Lodi and G. Zarpellon. On learning and branching: a survey. *TOP*, 25(2):207–236, July 2017. ISSN 1863-8279. doi: 10.1007/s11750-017-0451-6. URL https://doi.org/10.1007/s11750-017-0451-6.

[11] G. Maudet and G. Danoy. Search Strategy Generation for Branch and Bound Using Genetic Programming, Dec. 2024. URL http://arxiv.org/abs/2412.09444. arXiv:2412.09444 [cs].

[12] V. Nair, K. Dvijotham, I. Dunning, and O. Vinyals. Learning Fast Optimizers for Contextual Stochastic Integer Programs. In *AAAI*, 2018.

[13] J. Orlin. A faster strongly polynomial minimum cost flow algorithm. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 377–387, New York, NY, USA, Jan. 1988. Association for Computing Machinery. ISBN 978-0-89791-264-8. doi: 10.1145/62212.62249. URL https://dl.acm.org/doi/10.1145/62212.62249.

[14] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 229–234, New York, NY, USA, Jan. 1988. Association for Computing Machinery. ISBN 978-0-89791-264-8. doi: 10.1145/62212.62233. URL https://dl.acm.org/doi/10.1145/62212.62233.

[15] M. Paulus and A. Krause. Learning To Dive In Branch And Bound. *Advances in Neural Information Processing Systems*, 36:34260–34277, Dec. 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6bbda0824bcc20749f21510fd8b28de5-Abstract-Conference.html.

[16] Y. Rubner, C. Tomasi, and L. J. Guibas. The Earth Mover's Distance as a Metric for Image Retrieval. *International Journal of Computer Vision*, 40(2):99–121, Nov. 2000. ISSN 1573-1405. doi: 10.1023/A:1026543900054. URL https://doi.org/10.1023/A:1026543900054.

[17] L. Scavuzzo, K. Aardal, A. Lodi, and N. Yorke-Smith. Machine learning augmented branch and bound for mixed integer linear programming. *Mathematical Programming*, Aug. 2024. ISSN 0025-5610, 1436-4646. doi: 10.1007/s10107-024-02130-y. URL https://link.springer.com/10.1007/s10107-024-02130-y.

[18] O. Shafaghsorkh and A. Ayough. Application of soft operations research methods in healthcare: A systematic review. *Journal of Industrial Engineering and Management Studies*, 9(1):136–147, July 2022. ISSN 2476-308X. doi: 10.22116/jiems.2022.335541.1483. URL https://jiems.icms.ac.ir/article_152906.html. Publisher: Iran Center for Management Studies.

[19] Z. Steever, C. Murray, J. Yuan, M. Karwan, and M. Lübbecke. An Image-Based Approach to Detecting Structural Similarity Among Mixed Integer Programs. *INFORMS Journal on Computing*, Mar. 2022. doi: 10.1287/ijoc.2021.1117. URL https://pubsonline.informs.org/doi/abs/10.1287/ijoc.2021.1117. Publisher: INFORMS.

[20] Z. Steever, K. Hunt, M. Karwan, J. Yuan, and C. C. Murray. A Graph-Based Approach for Relating Integer Programs. *INFORMS Journal on Computing*, page ijoc.2023.0255, Mar. 2024. ISSN 1091-9856, 1526-5528. doi: 10.1287/ijoc.2023.0255. URL https://pubsonline.informs.

org/doi/10.1287/ijoc.2023.0255.

[21] Y. Tang, S. Agrawal, and Y. Faenza. Reinforcement Learning for Integer Programming: Learning to Cut. In *nternational Conference on Machine Learning*, 2020.

[22] D. Thuerck, B. Sofranac, M. E. Pfetsch, and S. Pokutta. Learning Cuts via Enumeration Oracles. *Advances in Neural Information Processing Systems*, 36:79108–79123, Dec. 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/fa0126bb7ebad258bf4ffdbbac2dd787-Abstract-Conference.html.

[23] A. Tiwari, P. N. Hoyos, W. Hutabarat, C. Turner, N. Ince, X.-P. Gan, and N. Prajapat. Survey on the use of computational optimisation in UK engineering companies. *CIRP Journal of Manufacturing Science and Technology*, 9:57–68, May 2015. ISSN 1755-5817. doi: 10.1016/j.cirpj.2015.01.003. URL https://www.sciencedirect.com/science/article/pii/S175558171500005X.

[24] M. Turner, T. Koch, F. Serrano, and M. Winkler. Adaptive Cut Selection in Mixed-Integer Linear Programming. *Open Journal of Mathematical Optimization*, 4:1–28, 2023. ISSN 2777-5860. doi: 10.5802/ojmo.25. URL https://ojmo.centre-mersenne.org/articles/10.5802/ojmo.25/.

[25] C. Villani. The Wasserstein distances. In M. Berger, B. Eckmann, P. De La Harpe, F. Hirzebruch, N. Hitchin, L. Hörmander, A. Kupiainen, G. Lebeau, M. Ratner, D. Serre, Y. G. Sinai, N. J. A. Sloane, A. M. Vershik, and M. Waldschmidt, editors, *Optimal Transport*, volume 338, pages 93–111. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-540-71049-3 978-3-540-71050-9. doi: 10.1007/978-3-540-71050-9_6. URL http://link.springer.com/10.1007/978-3-540-71050-9_6. Series Title: Grundlehren der mathematischen Wissenschaften.

[26] I. H. Witten, E. Frank, and M. A. Hall. *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann series in data management systems. Morgan Kaufmann, Burlington, MA, 3rd ed edition, 2011. ISBN 978-0-12-374856-0.

[27] J. Zhang, C. Liu, X. Li, H.-L. Zhen, M. Yuan, Y. Li, and J. Yan. A survey for solving mixed integer programming via machine learning. *Neurocomputing*, 519:205–217, Jan. 2023. ISSN 0925-2312. doi: 10.1016/j.neucom.2022.11.024. URL https://www.sciencedirect.com/science/article/pii/S0925231222014035.