An Image-based Approach to Detecting Structural Similarity Among Mixed Integer Programs

Zachary Steever (corresponding author)
Ph.D. Candidate
Department of Industrial and Systems Engineering
University at Buffalo, The State University of New York
Phone: +1-716-698-4410
zjsteeve@buffalo.edu

Dr. Chase Murray
Assistant Professor
Department of Industrial and Systems Engineering
University at Buffalo, The State University of New York
Phone: +1-716-645-4716
cmurray3@buffalo.edu

Dr. Junsong Yuan
Associate Professor
Department of Computer Science and Engineering
University at Buffalo, The State University of New York
Phone: +1-716-645-0562
jsyuan@buffalo.edu

Dr. Mark Karwan
Praxair Professor in Operations Research
Department of Industrial and Systems Engineering
University at Buffalo, The State University of New York
Phone: +1-716-645-2422
mkarwan@buffalo.edu

Dr. Marco Lübbecke
Professor and Head of Group
RWTH Aachen University
Phone: +49-241-80-93362
marco.luebbecke@rwth-aachen.de

Steever, Zachary and Murray, Chase and Yuan, Junsong and Karwan, Mark and Lübbecke, Marco. An Image-based Approach to Detecting Structural Similarity Among Mixed Integer Programs (April 20, 2020). Available at SSRN: https://ssrn.com/abstract=3437981

An Image-based Approach to Detecting Structural Similarity Among Mixed Integer Programs

Abstract: Operations researchers have long drawn insight from the structure of constraint coefficient matrices (CCMs) for mixed integer programs (MIPs). We propose a new question: Can pictorial representations of CCM structure be used to identify similar MIP models and instances? In this paper, CCM structure is visualized using digital images, and computer vision techniques are employed to detect latent structural features therein. The resulting feature vectors are used to measure similarity between images and, consequently, MIPs. An introductory analysis examines a subset of the instances from strIPlib and MIPLIB 2017, two online repositories for MIP instances. Results indicate that structure-based comparisons may allow for relationships to be identified between MIPs from disparate application areas. Additionally, image-based comparisons reveal that ostensibly similar variations of an MIP model may yield instances with markedly different mathematical structures.

Keywords: matrix structure, instance comparison, model comparison, computer vision, feature engineering

1 Introduction

The field of operations research is rooted in the process of defining, analyzing, and comparing alternatives. In this paper, we consider comparisons among mixed-integer programming (MIP) models and instances. Traditionally, similarity between MIPs has been defined based on specific model characteristics. Kendall's queueing notation (Kendall 1951) is a well known example of a model ontology. Similar "M/M/1"-style descriptors exist for models from machine and project scheduling (Brucker et al. 1999), warehouse management (van den Berg and Zijm 1999), multi-objective optimization (Marler and Arora 2004), pickup and delivery (Berbeglia et al. 2007), facility location (Hamacher and Nickel 1998, Farahani and Hekmatfar 2009), vehicle routing (Eksioglu et al. 2009), and game theory (Liang and Xiao 2013). Unfortunately, these comparative methods are siloed based on specific problem types

and application areas. As a result, they do not allow for connections to be drawn between models from different fields of research.

More recently, a new approach has emerged, whereby MIP similarity is measured according to the properties of individual instances. This approach was introduced by Gleixner et al. (2019a), who define an MIP similarity metric based on instance statistics (e.g., the proportion of decision variables which are binary). In line with the second approach, this paper presents a methodology for comparing MIPs based on the structure of the constraint coefficient matrices (CCMs) for one or more instances of a model. The CCM is the matrix $\mathbf{A}$ in an integer program of the form

$$\max\{\mathbf{cx} : \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}_+^n\}. \tag{1}$$

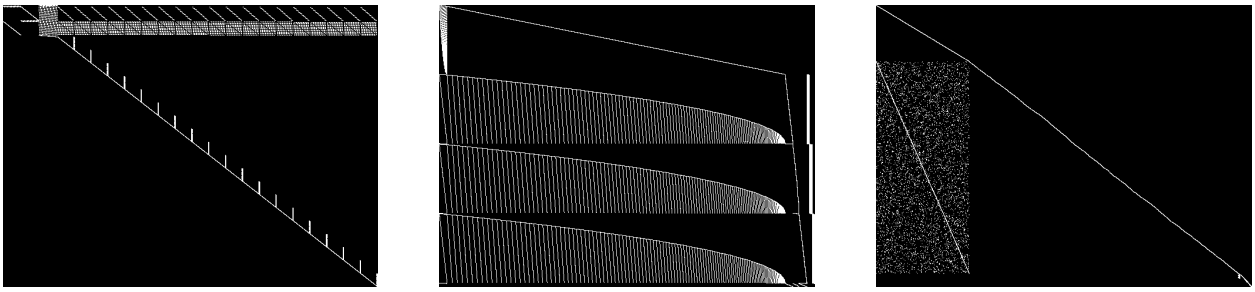
Operations researchers have long understood the importance of CCM structure in the context of solution techniques. Dantzig-Wolfe decomposition is widely used when a CCM contains both block-diagonals and coupling constraints (Dantzig and Wolfe 1960). Similarly, a CCM may present blocks of constraints which are connected by a subset of complicating variables (variables which, when assigned a constant value, leave behind a much simpler decision problem). Benders' decomposition exploits this CCM structure by generating two sub-problems – one with the complicating variables and another without – each of which is more tractable than the original problem (Benders 1962). Network flow problems, in particular, benefit from the decomposition of such block structures (Glockner and Nemhauser 2000, Jones et al. 1993). CCM structure has also been exploited to derive cutting planes (Marchand et al. 2002), and to identify total unimodularity - a property which guarantees that the linear programming relaxation of an MIP instance will yield integer solutions (Commoner 1973).

Although CCM structure has been leveraged extensively for solving individual MIP instances, it has not been well exploited as a basis for MIP comparison. The ability to compare and relate MIPs in this way bears a number of useful applications. Problems with similar CCM structures may benefit from similar solution approaches. Modelers may also wish

to modify constraints in their MIP based on a modeling technique which made a related problem easier to solve. This type of comparison may even help in identifying the types of structures that make certain problems "hard" to solve. Moreover, a comparison based on CCM structure is agnostic to instance application, allowing for connections to be drawn between problems which may otherwise be viewed as unrelated. As a result, an approach for comparing MIPs based on CCM structure addresses the shortcomings of comparison based on either problem difficulty or model characteristics.

The proposed image-based approach for identifying structural similarity among MIPs is comprised of three key steps.

1. **Generate images:** Computers represent black and white images as $m \times n$ matrices, where each matrix entry (or pixel) contains a value of either 0 (black) or 255 (white). By setting all nonzero coefficient values equal to 255, a CCM can be treated as a digital image. Figure 1 contains several examples of images rendered from CCMs in this manner. The rows and columns of each pixelated image represent the constraints and decision variables in an MIP instance, respectively, and each white pixel (i, j) implies a nonzero coefficient for variable j in constraint i.



(a) `P-n19-k2_cvrp`: Capacitated vehicle routing problem.

(b) `BR6_19-hifi`: Container loading problem.

(c) `d1000_09_alternative`: Map labeling problem.

Figure 1: Image representations of the CCM structure for three MIP instances contained in strIPlib (Bastubbe et al. 2020).

2. **Encode:** The second step is to encode CCM images into feature vectors. A feature vector contains measurements of the quantifiable attributes, or features, of an object.

Encoding an image into a feature vector serves two purposes. First, comparing images according to feature-vector encodings (rather than directly comparing pixel values) replaces element-wise errors with feature-wise errors. These feature-wise errors are less variant with respect to image transformations such as translation, rotation, and reflection (Larsen et al. 2015). Additionally, a feature vector provides a common-sized representation for images of varying shapes and sizes. In this paper, feature vectors are extracted from an autoencoder (a form of neural network) which has been trained on a set of CCM images.

3. **Compare:** Finally, an image-based structural similarity (ISS) metric is used to quantify the resemblance between pairs of images. This metric (defined formally by Equation (2) in Section 3.3) returns the Euclidean distance between a pair of feature vectors. A pair of images which are structurally identical will have an ISS value of zero.

Collectively, we refer to this process as *image-based comparison of integer programs* (ICIP). The purpose of this paper is to formally define ICIP, and to provide examples of the types of relationships it is capable of identifying. In the future, these relationships may be leveraged when studying and solving new problems. To demonstrate the utility of the proposed approach, we analyzed a subset of instances from two online MIP instance repositories. The *library of structured integer programs* (strIPlib) is a collection of more than 10,000 MIP instances (Bastubbe et al. 2020). These instances come from "pure" problems (e.g., the bin packing problem, the cutting stock problem), which are known to have clear and exploitable structures (namely for the application of column generation and other decomposition techniques). The relationships between some of these pure problems are known a priori. These relationships provide a "ground truth," and allow for validation of the proposed comparison approach. New relationships between problems are also revealed through these comparisons.

The second repository considered in our analysis is the *mixed integer programming library* (MIPLIB 2017), a result of collaborative efforts by more than 20 researchers (Gleixner et al. 2019b). Now in its sixth edition, MIPLIB 2017 offers a collection of 1,065 MIP instances. A

subset of these instances contain information about the submitting author and descriptions of the instance application are provided. Many of these instances come from complex, real-world settings, as opposed to clearly defined theoretical problem definitions. These instances, which we refer to as "blended," are compared to the strIPlib problems.

The remainder of this paper is organized as follows. Section 2 continues the discussion on the only other quantifiable comparison of MIP instances (from Gleixner et al. (2019a) and MIPLIB 2017). Details of ICIP are described in Section 3, while data collection and model training procedures are discussed in Section 4. The effectiveness of this approach is assessed in Section 5 through an analysis of strIPlib and MIPLIB 2017. Although paper length restrictions limit the scope of these analyses, additional results are available on a companion website (`https://icip.optimatorlab.org`). Finally, Section 6 provides a summary discussion and outlines opportunities for future research.

2 Instance comparison in MIPLIB 2017

Details of the instance comparison method used in MIPLIB 2017 (referred to simply as MIPLIB for the remainder of this paper) are presented in Gleixner et al. (2019a). To the best of our knowledge, this is the only other documented method for quantifying the similarity between MIP instances. Note that this is unique from published works which provide descriptive or performance-based comparisons for a small number of related MIPs (e.g., Croxton et al. 2003, Ostrowski et al. 2011, Ku and Beck 2016). This section is, therefore, devoted to a summary of the instance comparison scheme employed by MIPLIB. Following this discussion, the novel features of ICIP are highlighted.

2.1 Comparison using instance statistics as features

In MIPLIB, instances are compared according to feature vectors containing 110 scaled instance statistics. For a given instance, these statistics describe properties of the CCM,

decision variables, objective function coefficients, and right-hand-side vector ($\mathbf{A}$, $\mathbf{x}$, $\mathbf{c}$, and $\mathbf{b}$ in Definition (1), respectively). The similarity between two MIP instances is measured as the Euclidean distance between their feature vectors. Each instance in MIPLIB is accompanied by a list of the five most similar remaining instances according to this metric.

In Gleixner et al. (2019a), the collection of 110 instance statistics is divided into 11 feature groups. Some of the CCM-related feature groups include `size` (describing the number of rows, columns, and nonzero entries in the CCM), `matrix coefficients` (containing the mean, minimum, and other measurements of the coefficient values), and `row dynamism` (which describes the variance of coefficient values within an average constraint). The `constraint classification` and `decomposition` feature groups, however, serve the express purpose of identifying structures in the model. They are, therefore, of particular interest in this paper.

2.1.1 Constraint classification features

The `constraint classification` feature group contains the percentages of constraints in an MIP instance which belong to each of 17 uniquely defined constraint classes. Definitions for these constraint classes are provided in Table 1. The notion of constraint classification was originally introduced by Nemhauser et al. (1992), although this early constraint classification scheme was not employed to compare MIP instances. Rather, this approach was used to inform preprocessing, constraint generation, branching, and other components of the solution process.

Both constraint classification schemes share an important property: The set of constraint classes is hierarchical, such that a constraint is assigned only to the lowest class in the hierarchy that contains it. This presents two particular challenges with respect to instance comparison. First, assigning a constraint to a single class does not account for hierarchical relationships between constraint classes. For example, all `invariant knapsack` constraints are also `knapsack` constraints, as defined in Table 1. The MIPLIB's comparison scheme, how-

ever, will identify no constraint-based similarities between an instance with only `invariant knapsack` constraints and another with only `knapsack` constraints. Second, the upper-most constraint class acts as a catchall. This class, called `general linear` in Gleixner et al. (2019a), provides no information regarding the structure of a constraint, but is used as a point of comparison between MIP instances. Over 9% of the constraints across all instances in MIPLIB are classified as `general linear` (i.e., none of the other defined constraint forms apply).

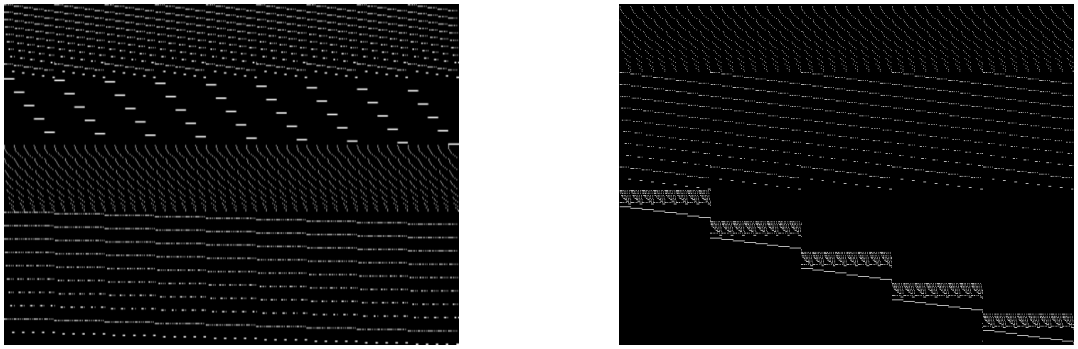
Table 1: Constraint classes defined by Gleixner et al. (2019a).

Abbr.	**Type**	**Linear constraints ...**
GEN	general linear	with no special structure
MIX	mixed binary	of the form $\sum a_k x_k + \sum p_j s_j \{\leq, =\}\ b, \quad x_i \in \{0,1\}\ \forall\ i,\ s_j$ cont. $\forall\ j$
INT	integer knapsack	of the form $\sum a_k x_k \leq b, \quad x_i \in \mathbb{Z}\ \forall\ i, b \in \mathbb{N}$
KPS	knapsack	of the form $\sum a_k x_k \leq b, \quad x_i \in \{0,1\}\ \forall\ i, b \in \mathbb{N} \geq 2$
BIN	binpacking	of the form $\sum a_i x_i + ax \leq a, \quad x, x_i \in \{0,1\}\ \forall\ i, a \in \mathbb{N} \geq 2$
EQK	equation knapsack	of the form $\sum a_i x_i = b, \quad x_i \in \{0,1\}\ \forall\ i, b \in \mathbb{N} \geq 2$
INV	invariant knapsack	of the form $\sum x_i \leq b, \quad x_i \in \{0,1\}\ \forall\ i, b \in \mathbb{N} \geq 2$
CAR	cardinality	of the form $\sum x_i = k, \quad x_i \in \{0,1\}\ \forall\ i, k \geq 2$
COV	set covering	of the form $\sum x_i \geq 1, \quad x_i \in \{0,1\}\ \forall\ i$
PAC	set packing	of the form $\sum x_i \leq 1, \quad x_i \in \{0,1\}\ \forall\ i$
PAR	set partitioning	of the form $\sum x_i = 1, \quad x_i \in \{0,1\}\ \forall\ i$
VAR	variable bound	of the form $ax + by \leq c, \quad x \in \{0,1\}$
PRE	precedence	of the type $ax - ay \leq b$ where x and y must have the same type
AGG	aggregation	of the type $ax + by = c$
SIN	singleton	with a single variable
FRE	free	with no finite side
EMP	empty	with no variables

2.1.2 Decomposition features

The `decomposition` feature group includes 10 statistics related to CCM structure. These statistics are extracted using Generic Column Generation, or GCG (Gamrath and Lübbecke

2010). GCG is a decomposition-focused solver based on optimization software, SCIP (Gleixner et al. 2018). Specifically, (Gleixner et al. 2018) use GCG in version 3.0.0, based on SCIP version 6.0.0. For a system of linear equations or inequalities, any permutation of the rows and columns will yield a mathematically identical representation of the underlying instance. GCG employs a number of "structure detectors" to identify row and column permutations that may be suitable for the application of Dantzig-Wolfe or Benders' decomposition (Gleixner et al. 2018). Figure 2 contains images of an example CCM before and after the application of GCG's structure detection algorithm.



(a) Raw CCM image for `10teams` (b) Decomposed CCM image for `10teams`

Figure 2: Visualizing the CCM structure for `10teams`, one of the MIP instances in MIPLIB. Figure 2a represents this CCM in its raw form; the row and column sequencing corresponds to the order in which constraints and variables, respectively, were defined by the modeler. In Figure 2b, the rows and columns of this CCM have been permuted by GCG such that nonzero coefficients (in the context of an image, white pixels) are clustered. The manner in which these specific images were generated is discussed in Section 3.1.

GCG returns a number of candidate decompositions for each instance. For each decomposition, GCG also provides a number of related statistics which constitute the `decomposition` feature group. As an example, the maximum white area (maxwhite) score describes the fraction of contiguous zero-valued entries within a CCM.

2.2 Gaps addressed by image-based comparison

The instance comparison approach employed by MIPLIB is rigorous in its description of MIP instances, with over 100 uniquely defined features. The resulting feature vector encompasses

each term in Definition (1) of an MIP instance ($\mathbf{A}$, $\mathbf{x}$, $\mathbf{c}$, and $\mathbf{b}$). This is an advantage over the approach presented in this paper, which considers only the matrix $\mathbf{A}$ in Definition (1). Additionally, the Gleixner et al. (2019a) approach is less reliant than our image-based approach on GCG's structure detection procedure. Specifically, decomposition is used only to generate a select few features in the Gleixner et al. (2019a) method. As outlined in Section 3, our image-based approach relies on the assumption that the decomposition with the greatest maxwhite score best reflects the underlying structure of the corresponding instance. More broadly, the heuristic nature of GCG's structure detection algorithm implies that ICIP will not be invariant to row and column permutations. However, matrix seriation (re-ordering) algorithms comprise an entire field of study on their own, and are outside the scope of this paper.

Despite its strengths, there are several challenges faced by MIPLIB's instance comparison approach which are mitigated by ICIP. First, the proposed approach avoids the difficulties associated with making comparisons based on constraint classes (e.g., representing hierarchical relationships, or including a catchall class which provides no descriptive information). In fact, the proposed approach does not rely on manually-defined features of any kind. Instead, computer vision and deep learning techniques are employed to extract feature vectors automatically. This has two effects, the first of which is a reduction in the amount of human bias involved in the comparison procedure. Additionally, computers are capable of identifying image characteristics which are highly discriminative, and which a human may not think to use (or even perceive) as a feature. While the feature vectors extracted through deep learning are difficult to interpret, they may be better at capturing CCM structure (and do so more reliably) than the decomposition-related features (e.g., density of nonzero entries, maxwhite score) defined by Gleixner et al. (2019a).

Finally, it is impossible to define whether the relationships uncovered by this method are "better" than those revealed by the approach used in MIPLIB. Instead, we propose that ICIP is valuable as a supplement to MIPLIB's methodology, offering an alternate view of

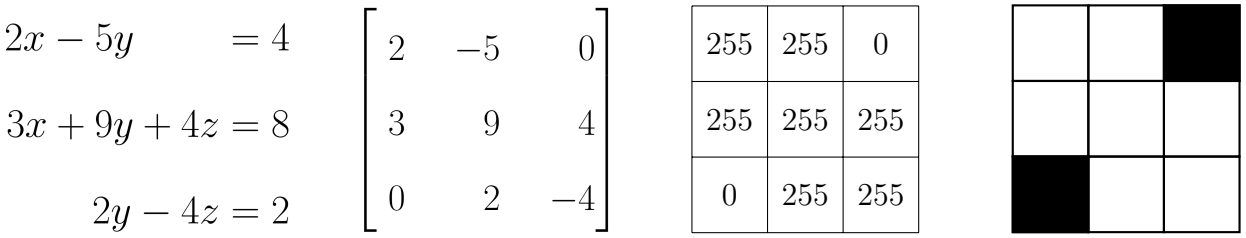
what it means for MIPs to be similar.

3 Methodology

This section provides a detailed discussion of the proposed process for representing CCM structure using images, automating the construction of feature vectors for those images, and comparing images according to their feature vectors. This discussion necessitates an overview of deep learning techniques as they relate to computer vision, which is provided in Section 3.2.

3.1 Step 1 – Generate images

The first step in the proposed approach is to represent CCM structure using images. Pixel values in a black and white image are either equal to 0 (black) or 255 (white). By casting all nonzero CCM entries to 255 and leaving all zero-valued coefficients alone, the corresponding structure can be rendered as an image. This process is depicted in Figure 3.



(a) Linear equations (b) CCM (c) Casting (d) Image

Figure 3: The stages of representing a CCM as a digital image.

3.1.1 Matrix decomposition

The initial (raw) ordering of rows and columns for a CCM is determined by the order in which constraints and decision variables, respectively, were defined by the instance author. For many of the instances examined in this paper, defined CCM structures were not readily

apparent in this raw form; this is especially true for the blended MIPLIB instances. This motivated the use of GCG 3.0 for CCM decomposition prior to image generation. MIP instances were stored and loaded into GCG as Mathematical Programming System (MPS) files (IBM 1969). The MPS file format is used to store and load linear and mixed integer programming instances, and is recognized by every major optimization software. Once loaded, GCG's default structure detection procedure was applied. For this paper, the decomposition with the greatest maxwhite score was selected for each instance.

3.1.2 Rendering CCMs as images

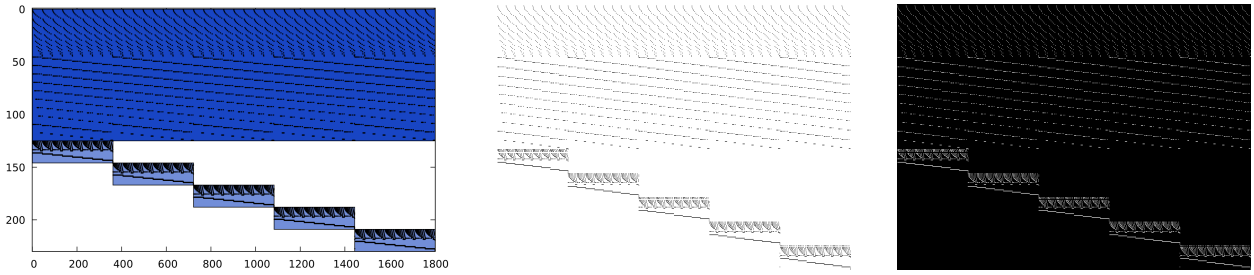
In conjunction with the command-line graphing tool *gnuplot* (Racine 2006), GCG can also help to visualize CCM structure. For a given decomposition, the "write" function in GCG will generate a gnuplot (GP) script. This script contains the code necessary for gnuplot to graph the corresponding CCM structure. There are several important things to note about these images, however.

- The images rendered by GCG's GP files contain margins, axes, and shaded blue boxes around detected block structures (e.g., Figure 4a). These elements, which do not relate to CCM structure, introduce unnecessary visual elements to the CCM images. Therefore, the GP files were edited to remove all such non-structural information.

- The color-mapping in the images produced by GCG is inverted, by default. That is, nonzero CCM entries are represented as black pixels against a white background (e.g., Figure 4b). While this is perhaps a more intuitive way for humans to view patterns, it is an inaccurate representation of the structure of nonzero coefficients in a CCM (recall that black pixels in a digital image are zero-valued). This color mapping is reverted by setting all zero-valued pixels equal to 255, and vice versa ($\text{CCM}_{ij} = 255 - \text{CCM}_{ij},\ \forall\ i, j$), as in Figure 4c.

- The GP scripts generated by GCG yield images of a standard size (640×480 pixels),

regardless of the number of rows and columns in the corresponding CCM. GCG's compression algorithm employs binary max pooling. That is, if any pixel in a macro region contains a nonzero element, the compressed, single-pixel representation of that region contains a value of 255 (and zero otherwise). It is true that a grayscale or full-color representation of coefficient values could retain more detail from the CCM. However, GCG's structure detection procedure is a critical component of this methodology. As such, we opted to work with the restriction that decomposed images will be black-and-white.



(a) GCG-generated gnuplot

(b) CCM image after axes and shading have been stripped

(c) CCM image after pixel values have been inverted

Figure 4: GCG's gnuplots include axes, as well as shading to highlight block structures which have been identified, as in Figure 4a. In Figure 4b, these elements are removed such that only the true structural information remains. Finally, in Figure 4c, the color mapping has been inverted.

3.2 Step 2 – Encode

In the second step, images are encoded as feature vectors using techniques from deep learning. There are many different methods defined within the field of computer vision for extracting features from images. Fisher Vector Encoding and Bag-of-Words models are two commonly used approaches which were considered for this application. However, these approaches are based on locally invariant visual features, which are best extracted from a rich local structure (Sánchez et al. 2013, Zhang et al. 2010). CCM images are black and white, as well as highly sparse, meaning that they lack such richness. Therefore, these methods are not well suited

for this application.

Deep learning methods also automate the feature engineering process, making them highly effective for perceptual tasks (Elsken et al. 2019). Convolutional neural networks (CNNs) are deep learning models which have been well studied (Liu et al. 2017). A wide range of optimization problems have benefited from the application of CNNs, from waveform optimization in audio enchancement tasks (Fu et al. 2018) to the design of optimal computational strategies for 3D topology models.

CNNs have also been used to study optimization models within the field of operations research. Several early works employed CNNs to develop optimal (or near optimal) solutions for integer programming formulations (Simon 1988, Ohlsson et al. 1993, Tsuchiya et al. 1996). More recently, CNNs have been applied to the optimization process, itself. Bengio et al. (2018) use CNNs to generate improved cutting planes (and hence, tighter bounds), while Gasse et al. (2019) employ CNNs to improve the variable selection process during branch-and-bound. A key difference between these applications and the one presented in this paper is that we use CNNs to *study* optimization problems, rather than to help solve them.

One type of CNN, referred to as an autoencoder, which is particularly well suited for image comparison applications like the one presented in this paper. As described by Liu et al. (2017), autoencoders are trained to encode an input into an abstract representation, and then use that representation to reconstruct the input as closely as possible.

The general structure of an autoencoder is represented in Figure 5. The inner-most layer (labeled "Coded" in Figure 5) contains values for the latent features which have been detected. Once an autoencoder is sufficiently trained (i.e., it is able to decompose and reconstruct input data with a high degree of accuracy), the values in the "Coded" layer may be extracted for use as a feature vector for the corresponding input data. The use of autoencoders to extract feature vectors from images is well documented in fields such as medical image analysis (Camlica et al. 2015, Sharma et al. 2016, Chen et al. 2017, Li et al.

2018, Cheng et al. 2018) and image retrieval (Sklan et al. 2015, Zhou et al. 2017, Rahim et al. 2018, Qin et al. 2018, Chen et al. 2018), among others. Although CCM images differ from "traditional" images (in that they do not depict tangible objects), they are fundamentally comprised of the same image primatives (e.g., oriented edges, corners). As such, an autoencoder may still be reasonably employed for feature extraction in this application.

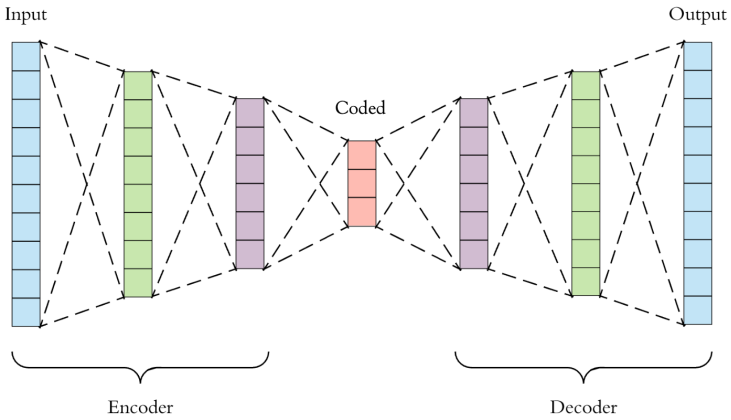
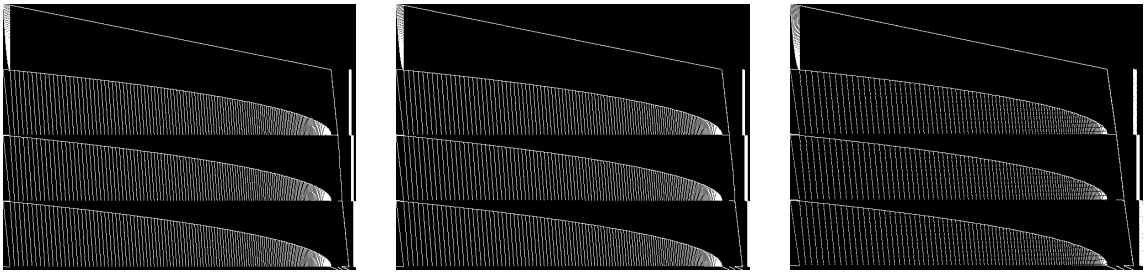


Figure 5: The general structure of an autoencoder (Derdat 2013).

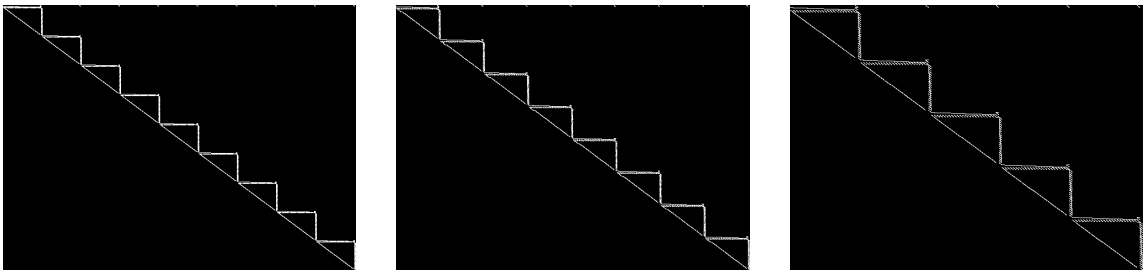
One property of CNNs which make them well suited for image-based feature extraction is their robustness to certain image transformations. In the context of decomposed CCM images, specifically, there are three primary transformations which the autoencoder is tasked with recognizing. The first of these transformations is referred to as "shading." An example of this transformation can be seen in Figure 6, which contains the CCM images for three instances from strIPlib of the container loading problem. All three images contain the same curved structure, but the shading inside that structure varies from one image to the next.



(a) `BR6_5-hifi` (b) `BR6_19-hifi` (c) `BR6_40-hifi`

Figure 6: An example of shading invariance in CCM images.

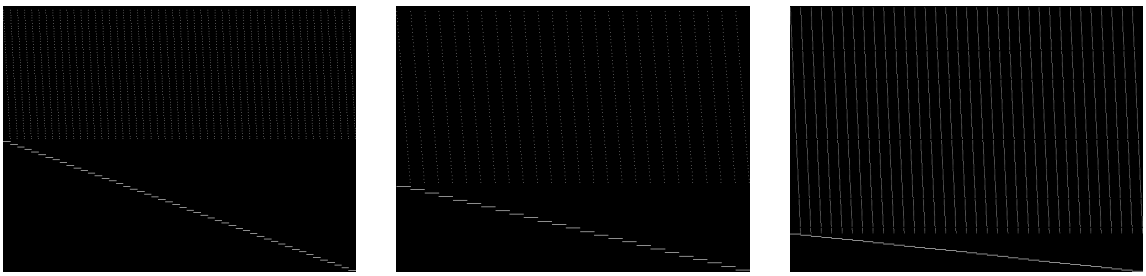
The second transformation of interest is referred to as "pattern density." Figure 7 depicts three instances of the capacitated vehicle routing problem, also from strIPlib. Each of these images contains the same pattern – a staircaise structure underscored by a solid main diagonal – in varying frequencies. For example, there are nine "steps" in the staircase shown in Figure 7a, as opposed to only five in Figure 7c.



(a) `E031-09h.cvrp` (b) `E026-08m.cvrp` (c) `E023-05s.cvrp`

Figure 7: An example of pattern density invariance in CCM images.

Finally, instances from a common model may exhibit transformations with respect to "proportionality." Figure 8 contains the CCM images for three instances of the bin packing problem, selected from strIPlib. These images are comprised of two components. The first component is a series of nearly vertical parallel lines, and the second is a block-diagonal structure. For each image, however, these components exist in different proportions. Robustness to this type of transformation is reminiscent of shift-invariance, which CNNs are known to capture (Hyvärinen and Hoyer 2000).



(a) `N1C3W2_M.BPPeq` (b) `N1W2B3R5.BPPeq` (c) `N3W4B2R8.BPPeq`

Figure 8: An example of proportionality invariance in CCM images.

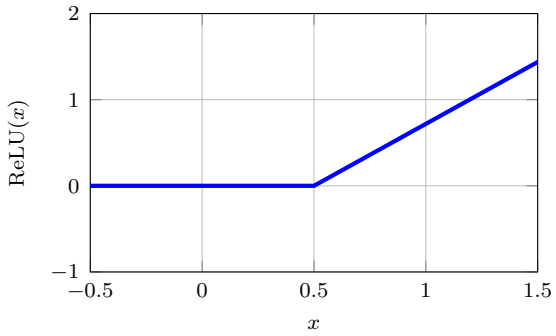
3.2.1 Autoencoder architecture

Table 2 contains a description of the autoencoder architecture used for this paper. The following principles were considered when developing this architecture.

Convolution: Convolutional layers are employed to aid in the extraction of image features. A CNN slides a number of templates (called convolutional kernels) across a digital image. The recommended size for a convolutional kernel is 3×3 (much smaller than the image itself); this kernel size was adopted here. The number of kernels in a convolutional layer represents the number of templates being checked (powers of two are recommended). Each convolutional layer in Table 2 contains either 8 or 16 kernels. The output from a convolutional layer will be an $m \times n \times k$ tensor, or multi-dimensional array. In this notation, $m \times n$ is the shape of the current image representation (either the original image or some compressed form), and k is the number of kernels being used to inspect that image (the values of m, n, and k are provided for each layer of our autoencoder architecture in Table 2). Each entry (m, n, k) in an output tensor represents the degree to which the k^{th} kernel (or template) is represented in pixel (m, n) of the current image.

Pooling: To obtain a compressed feature vector representation of an image, an autoencoder must reduce the size of the input data. The process of pooling does this by dividing a matrix into a number of disjoint sub-regions (pools). The values in each pool are then transformed according to a pooling function (in this case, max-pooling) into a single, representative value. Max-pooling was selected to retain as much structural information as possible through each stage of the compression process. Pooling occurs on the tensor that is output by a convolutional layer, and is applied to each of the k kernel dimensions separately. Using a large pool size will compress tensors too quickly, resulting in a greater loss of detail. As such, it is recommended to use a pool size of 2×2 and compress iteratively (by including more layers in the autoencoder).

Activation: The process of scaling kernel outputs in a CNN is known as activation. One of the most commonly used activation functions is known as the ReLU function. This function "shuts off" insignificant kernel outputs by casting values below a given threshold to zero, while leaving all other outputs as they are (e.g., Figure 9a). Given that all pixel values will be strictly positive, use of the ReLU activation function here is in line with the autoencoder's goal of reconstructing the inputs. Note that the ReLU function is differentiable at all points other than $x = 0$. After accounting for this singular discontinuity, differentiation of the ReLU function can be done quickly. It is therefore well suited for backpropogation, the procedure by which the network's weight structure is updated during training.



(a) ReLU

Figure 9: A graph of the ReLU activation function.

For the analyses in this paper, feature vectors were obtained by passing images as inputs to a trained autoencoder of the form defined in Table 2 and extracting the coded representation from Layer 12. It is standard procedure to normalize input data prior to training. In this setting, the pixel values in each image were scaled to between 0 and 1. Given that the images are purely black and white, this process yields binary input vectors. As such, training loss was measured using the binary crossentropy function native to TensorFlow.

3.3 Step 3 – Compare

Finally, similarity between images is measured according to the ISS metric. Let α and β be two images, and let $\boldsymbol{v_\alpha}$ and $\boldsymbol{v_\beta}$ denote their corresponding feature vector representations

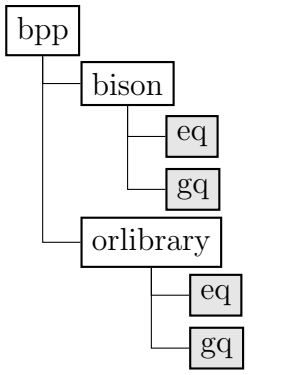
Table 2: Autoencoder architecture. Images are encoded in Layers 1–12, and decoded in Layers 13–22. Layer 12 contains the most densely encoded representation of the input data. Values are extracted from this layer for use as feature vectors.

Layer	**Type**	**Filters**	**Kernel Size**	**Output Size** (m, n, k)	**Activation**
1	Input	-	-	(480, 640, 1)	-
2	Conv2D	16	(3, 3)	(480, 640, 16)	ReLU
3	MaxPooling2D	-	(2, 2)	(240, 320, 16)	-
4	Conv2D	8	(3, 3)	(240, 320, 8)	ReLU
5	MaxPooling2D	-	(2, 2)	(120, 160, 8)	-
6	Conv2D	8	(3, 3)	(120, 160, 8)	ReLU
7	MaxPooling2D	-	(2, 2)	(60, 80, 8)	-
8	Conv2D	8	(3, 3)	(60, 80, 8)	ReLU
9	MaxPooling2D	-	(2, 2)	(30, 40, 8)	-
10	Conv2D	8	(3, 3)	(30, 40, 8)	ReLU
11	MaxPooling2D	-	(2, 2)	(15, 20, 8)	-
12	Conv2D	8	(3, 3)	(15, 20, 8)	ReLU
13	UpSampling2D	-	(2, 2)	(30, 40, 8)	-
14	Conv2D	8	(3, 3)	(30, 40, 8)	ReLU
15	UpSampling2D	-	(2, 2)	(60, 80, 8)	-
16	Conv2D	8	(3, 3)	(60, 80, 8)	ReLU
17	UpSampling2D	-	(2, 2)	(120, 160, 8)	-
18	Conv2D	8	(3, 3)	(120, 160, 8)	ReLU
19	UpSampling2D	-	(2, 2)	(240, 320, 8)	-
20	Conv2D	16	(3, 3)	(240, 320, 16)	ReLU
21	UpSampling2D	-	(2, 2)	(480, 640, 16)	-
22	Conv2D	1	(3, 3)	(480, 640, 1)	ReLU

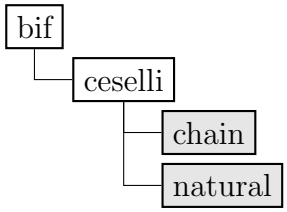
(extracted from a trained autoencoder). The ISS between these two images may be then defined as

$$\mathrm{ISS}_{\alpha\beta} = \left\| (\boldsymbol{v_\alpha} - \boldsymbol{v_\beta}) \right\|_2 . \tag{2}$$

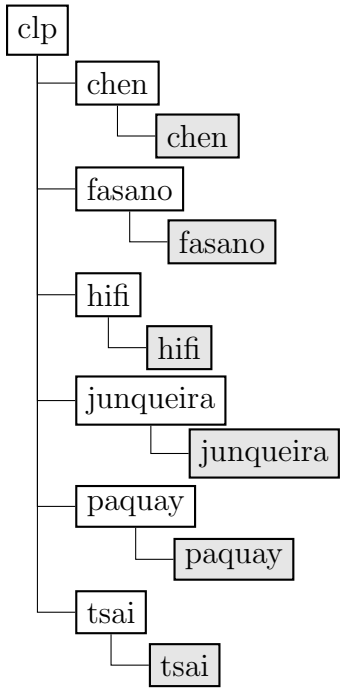
Equation (2) returns the Euclidean distance, or L2 norm, between $\boldsymbol{v_\alpha}$ and $\boldsymbol{v_\beta}$. These values are nonnegative and tend toward zero as the similarity between α and β increases. As discussed in Section 1, feature-wise comparisons such as these are less susceptible to image transformations than pixel-wise comparisons (Larsen et al. 2015).



(a) Bin packing problem



(b) Bin packing problem with item fragmentation



(c) Container loading problem

Figure 10: Example directory structures for three problem types in strIPlib, the bin packing problem (`bpp`), the bin packing problem with item fragmentation (`bif`), and the container loading problem (`clp`).

4 Data preparation and model training

There are more than 10,000 total instances contained in strIPlib. At the outer-most level, these instances are organized by their associated problem type (e.g., instances of the bin packing problem vs. instances of the cutting stock problem). Within each problem type, instances are further categorized. First, instances are separated by the author (or library) from which they originated. Then, some authors have defined additional categories by which to sort their instances. Figure 10 depicts the directory structures for three different problem types in strIPlib. For the remainder of this paper, each leaf node in a directory structure (highlighted in gray in Figure 10) is referred to as a "source." Instances were collected from 64 unique sources for this paper. The paths to these sources in strIPlib are provided in Table 10 in the Appendix.

For this paper, a set of training images were generated from a subset of instances contained in strIPlib. To ensure equal representation of each problem type in our training set, 50 instances were selected from each problem type. There were exactly 19 problem types which contained at least 50 total instances across all sources for that problem type (all other problem types were excluded from our study). These problem types, and the total number of sources for each type, are listed in Table 3. At least one instance was selected from each source for all 19 problem types. In addition, no instance was downloaded if its file size exceeded 1MB. This restriction was meant to alleviate the computational expense of GCG's structure detection procedure.

Table 3: Descriptive information for the 19 problem types included in this study. All of this information comes directly from Bastubbe et al. (2020).

Abbr.	**Full name**	**Problem description**	**Sources**
bpp	bpp	Bin packing problem	4
bp2	bpp2	Two-dimensional bin packing problem	1
bif	bppif	Bin packing problem with item fragmentation	2
clp	clp	Container loading problem	6
col	coloring	Vertex coloring problem	8
cpm	cpmp	Capacitated p-median problem	5
cut	cuttingstock	Cutting stock problem	8
cvr	cvrp	Capacitated vehicle routing problem	5
cwl	cwlp-ss	Capacitated warehouse (facility) location problem	1
gap	gatp-tmp	Generalized assignment problem	4
inr	inr	International nurse rostering problem	1
kps	kps	Knapsack problem with setups	1
lot	lotsizing	Lot sizing problem	6
map	maplabeling	Map labeling problem	2
pcp	pcp	Graph partition coloring problem	1
rel	relaxedClique	Relaxed clique covering problem	6
sch	scheduling	Scheduling problem	1
tup	tup	Traveling umpire problem	1
vrp	vrptw	Vehicle routing problem with time windows	1

In accordance with the aforementioned selection criteria, a collection of 950 MIP instances were selected at random and downloaded from strIPlib in the MPS file format. The image generation defined in Section 3.1 was then applied to generate 950 CCM images, which were

used to train the autoencoder. The trained autoencoder was employed to generate feature vectors for these images, which were used for subsequent ISS calculations.

4.1 Training statistics

The autoencoder architecture defined in Table 2 was constructed using TensorFlow version 2.1.0 (Chollet 2015) in Python 3.5.2. This architecture yielded a total of 6,721 trainable parameters. Three training epochs were conducted using a stochastic gradient descent (SGD) optimizer with Nesterov momentum, a batch size of 32 instances, and a learning rate of 0.01 (as recommended by Géron 2017). Several other optimizers (Adam, AdaGrad, and RMSProp) were also tested. These optimizers performed identically to SGD with respect to accuracy, but yielded higher average training losses. Recall that training loss was measured using TensorFlow's binary crossentropy function.

All training and testing was completed on a Dell OptiPlex 3046 running Linux Mint version 18.3, with an Intel Core i7-6700 processor and 16 GB of RAM. A summary of relevant training statistics is provided in Table 4. Note that both the training loss and training accuracy have converged within three training epochs, and that the entire training procedure was completed in just over 7 minutes. This speed results from the relative simplicity of the CCM images, which are both binary and highly sparse.

Table 4: Statistics from the autoencoder training procedure.

Epoch	**Time (min:sec)**	**Loss**	**Accuracy**
1	02:34	8.2837	0.4277
2	02:15	0.3852	0.9750
3	02:14	0.3852	0.9750

5 Results and analysis

In this section, the proposed MIP comparison approach is evaluated through a series of analyses. Recall that each of the instances in our collection belongs to a single source, which

in turn belongs to a specific problem type. In Section 5.1, similarity is examined at the problem-level. These comparisons serve primarily to validate the performance of our image-based comparison approach, as the relationships between some of these problems are known *a priori*.

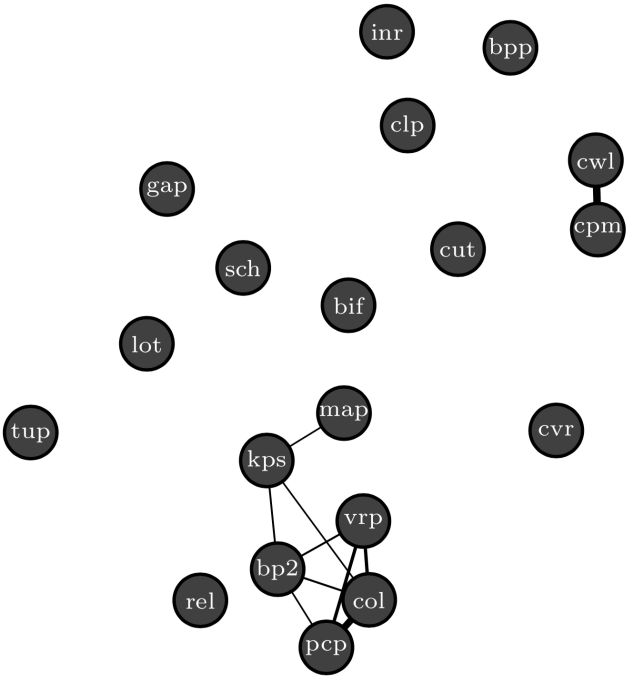
However, the results of Section 5.1 also raise questions about the purity of some problem types. Subsequently, Section 5.2 presents an analysis of similarity at the source level. Finally, a subset of instances from MIPLIB are introduced in Section 5.3. Comparisons between these blended instances and the pure strIPlib problems serve to highlight potential areas for future research. All results can also be found at `https://icip.optimatorlab.org`.

5.1 Analysis of strIPlib problem types

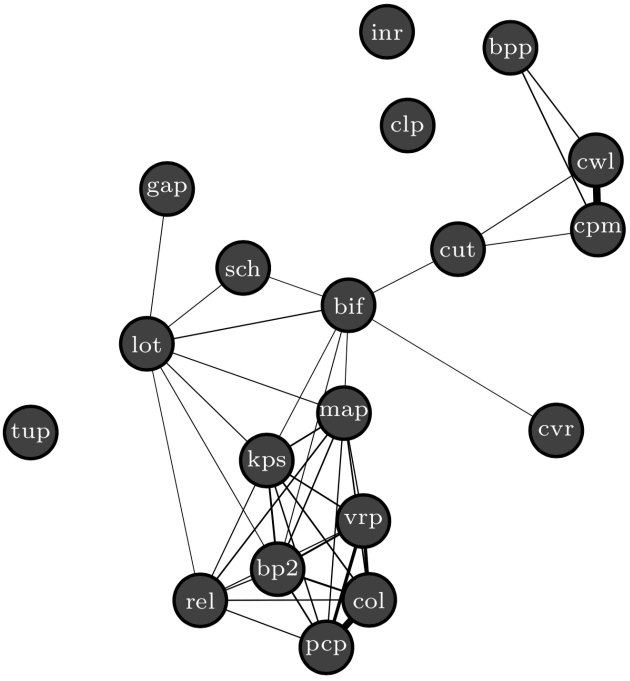
Problem-level feature vectors were constructed by averaging the feature vectors for all 50 instances of a given problem type. ISS values were then computed between pairs of problem-level feature vectors. The average ISS value between a pair of strIPlib problems is 5.81, with a standard deviation of 2.38. All ISS values were stored in a 19×19 matrix, which can be viewed as a distance matrix (recall that an ISS value is the Euclidean distance between two feature vectors in feature space). The ISS matrix was provided as input to the *qgraph* function in R (Epskamp et al. 2012), which takes an $n \times n$ distance matrix as input, and generates a set of coordinates in two dimensions for the n entities. More specifically, this two dimensional mapping seeks to (approximately) preserve the distance between each pair of entities as defined in the distance matrix.

Using the qgraph function, coordinates in 2D were generated for each of the 19 strIPlib problem types, as shown in Figure 11. In Figure 11a, edges are drawn between each pair of problems sharing an ISS value below 2.00. The ISS threshold is increased to 4.00 in Figure 11b. As the ISS threshold is increased, the graph becomes more connected, albeit by weaker relationships.

Figure 11 reveals a number of interesting relationships between problem types from



(a) ISS below 2.00

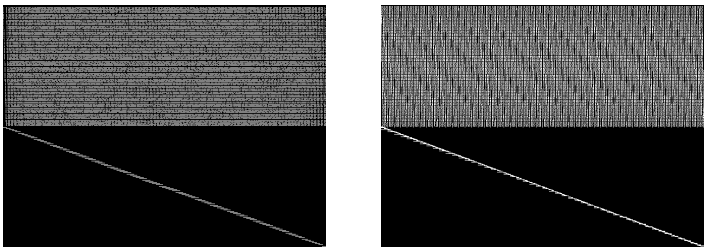


(b) ISS below 4.00

Figure 11: strIPlib problem types plotted in 2D. ISS values between pairs of problems are approximately preserved by the distance between those two problems in the plot. Edges are drawn between pairs of problems who share an ISS value below a defined threshold.

strIPlib. Consider the two initial groupings which appear in Figure 11a. In the upper right-hand portion of Figure 11a, a second cluster is formed by the `cwl` and `cpm` problem types. Recall from Table 3 that `cwl` and `cpm` represent the capacitated warehouse location problem and the capacitated p-median problem, respectively. These titles are often used interchangeably in the literature to describe the same conceptual problem (Stefanello et al. 2015, Fleszar and Hindi 2008, Maniezzo et al. 1998). These two problems share an ISS value of 0.45 – the lowest between any pair of strIPlib problems – further demonstrating the ability of ICIP to identify known relationships.

When the ISS threshold is increased in Figure 11b, both the `bpp` and `cut` problem types connect to `cwl` and `cpm`. In Figure 12, composite images for all four of these problem types are shown. A composite image is generated by "stacking" each of the CCM images for that problem type on top of one another. Composite images were not used to train the autoencoder; they serve purely as a visual aid. The ISS values between these problem types are also presented in Figure 12e.



(a) `cwl` (b) `cpm`

(c) `bpp` (d) `cut`

	cwl	cpm	bpp	cut
cwl	0.00	-	-	-
cpm	0.45	0.00	-	-
bpp	2.37	2.20	0.00	-
cut	3.18	3.42	4.51	0.00

(e) Problem-level ISS values for `cwl`, `cpm`, `bpp`, and `cut`.

Figure 12: Composite images for the `cwl`, `cpm`, `bpp`, and `cut` problem types are depicted in Figures 12a-12d. The ISS values between these problem types are presented in Figure 12e.

It can be inferred from Figures 12c and 12d that instances from the `bpp` and `cut` problem

types exhibited proportionality transformations, as defined in Section 3.2. There appears to be more variance in the proportionality of the cut instances than for the bpp instances, which is reflected in the higher ISS values for cut. However, the autoencoder was still able to identify that all four problem types share the same general structure.

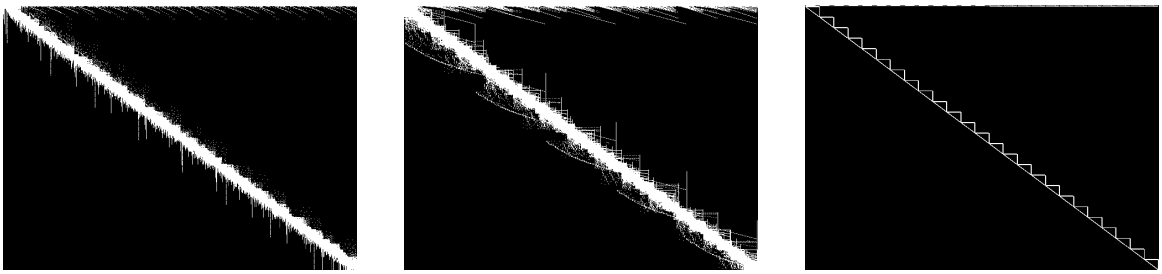
Table 5, which contains the average constraint composition for each of the four problem types, indicates that identifying similarities among these problems may be difficult when only constraint classes are considered. The cwl, cpm, and bpp problem types are comprised primarily of PAR and BIN constraints. Both cpm and bpp, however, contain a large percentage of COV constraints as well. Approximately 88% of constraints in the cut instances were classified as GEN, the catchall constraint class. This is an example of a problem for which more information is provided by ICIP than through constraint classification.

Table 5: Average constraint compositions for instances of the cwl, cpm, bpp, and cut problem types.

Problem	PAR	COV	CAR	BIN	INT	MIX	GEN
cwl	0.50	0.00	0.00	0.50	0.00	0.00	0.00
cpm	0.29	0.21	0.01	0.50	0.00	0.00	0.00
bpp	0.29	0.31	0.00	0.33	0.00	0.08	0.00
cut	0.00	0.00	0.00	0.00	0.12	0.00	0.88

The second cluster in Figure 11a contains the pcp, col, vrp, bp2, kps, and map problem types. For the sake of brevity, we will examine the first three of these problem types – pcp, col, and vrp – more closely. The composite images for these problem are shown in Figure 13. Each of these images reveals a strong main-diagonal with few nonzero entries elsewhere in the CCM.

The strongest connection in this group exists between the graph partition coloring problem (pcp) and the vertex coloring problem (col). These two problems share an ISS value of 0.49, the second lowest between any pair of strIPlib problem types. It is well established in the literature that the graph partition coloring problem is a generalization of the vertex coloring problem (Furini et al. 2018, Frota et al. 2010). Each of these problems, in turn, share an ISS value of 1.01 with vrp (the vehicle routing problem with time windows).

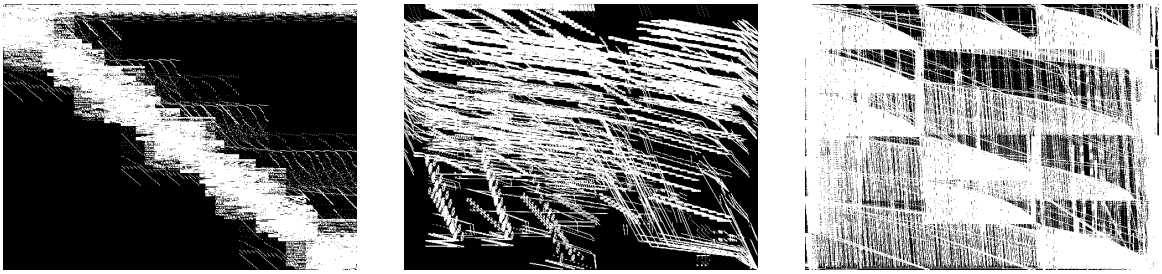


(a) pcp (b) col (c) vrp

Figure 13: Composite images for the pcp, col, and vrp problem classes.

Heuristics such as iterated local search (Xie et al. 2017), two-stage hybrid local search (Bent and Van Hentenryck 2004), ant colony optimization (Reimann et al. 2002), and simulated annealing (Chiang and Russell 1996) have all been applied to this problem with the explicit justification that they were effective at solving graph coloring problems.

Even after the ISS threshold has been increased in Figure 11b, the tup, inr, and clp problem types have no connections. Their composite images are shown in Figure 14. Unlike the other problem types in this analysis, both tup and inr originate from highly complex, real-world applications – the scheduling of umpire crews for Major League Baseball (Xue et al. 2015, Trick et al. 2012), and the International Nurse Rostering Competition (Ceschia et al. 2019, Haspeslagh et al. 2014), respectively. The complex nature of their applications necessitates the use of many different constraint types, as can be seen in Table 6, making these problems particularly unique. In a sense, then, it is a positive result that they do not resemble the other problem types included in this study.



(a) tup (b) inr (c) clp

Figure 14: Composite images for the tup, inr, and clp problem types.

Finally, consider the composite image for the clp problem type shown in Figure 14c.

Table 6: Average constraint compositions for the `tup` and `inr` problem types. An ϵ indicates a nonzero value less than 0.01.

Problem	EMP	SIN	AGG	PRE	VAR	PAR	PAC	COV	CAR	INV	BIN	MIX	GEN
inr	0.00	0.01	ϵ	ϵ	0.078	0.09	0.34	0.02	0.11	0.23	0.00	0.02	0.11
tup	ϵ	0.00	ϵ	0.00	ϵ	0.03	0.75	0.01	0.19	0.00	0.01	0.00	0.00

This image does not reveal a singular, clear structure. Instead, it appears as though several different structures are layered atop one another. This result led us to examine the purity of each problem type more closely. Theoretically, two instances of the same problem should contain the same types of constraints (although they may exist in different proportions). A simple measure of impurity, then, is the number of constraint classes which are present in one instance but not in another. The average constraint composition was calculated for each source using SCIP's linear constraint classifier, after which an impurity index was calculated for each problem type. This index is simply the total number of constraint types which are present in some, but not all, sources of a given problem type. As SCIP uses the 17 constraint classes defined in Table 1, the maximum possible impurity index is 17.

Table 7 contains the impurity index (Impurity) for all 19 problem types, along with two additional measures. Values in the "Mean ISS" column represents the average ISS value between two instances of a given problem type. As a reference, the global average ISS value between individual instances of any problem type is 7.45, with a standard deviation of 3.41. Next, we identified the 49 most similar instances to each individual instance in our collection. This number is based on the fact that, for a given instance, up to the first 49 most similar instances may belong to the same problem type. For each instance, a match rate was calculated as the percentage of the 49 most similar instance from a matching problem type. Values in the "%-Match" column of Table 7 represent the average match rate for instances of a given problem type.

It can be seen in Table 7 that 9 of the 19 problem types in this analysis have a nonzero impurity index. The various sources for any such problem type have modeled that problem using different formulations. Table 7 further reveals that problems with high impurity indices

perform poorly on the Mean ISS and %-Match measures. Notably, `clp` has the highest impurity index (9 out of 17) and Mean ISS (9.57), as well as the lowest %-Match (16.82%). These results called for an analysis of problems at the source level.

Table 7: Impurity, mean intra-problem ISS, and average Top-49 match rate for the 19 strIPlib problem types considered in this analysis.

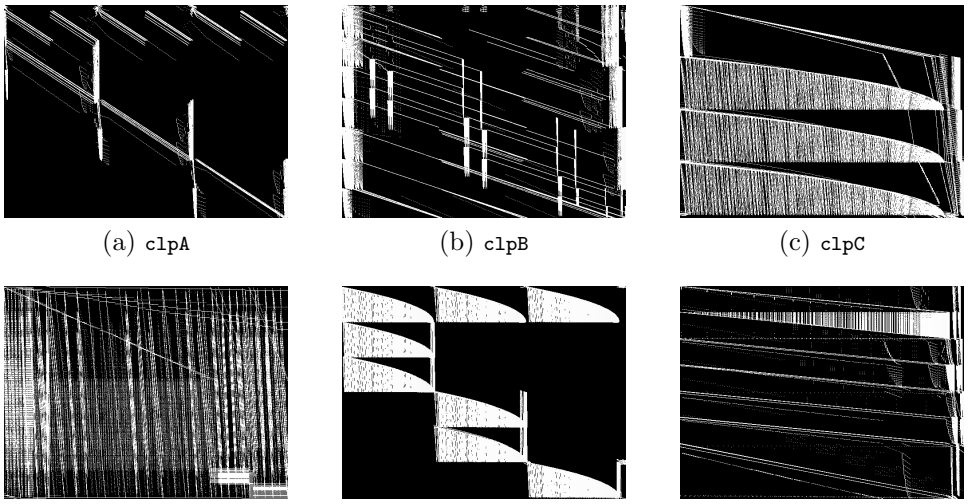
Problem	**Impurity**	**Mean ISS**	**% -Match**
`vrp`	0	0.00	100.00%
`cwl`	0	0.49	100.00%
`kps`	0	0.80	85.92%
`bp2`	0	0.79	67.59%
`pcp`	0	1.43	57.47%
`cpm`	0	3.95	38.45%
`sch`	0	4.54	45.59%
`gap`	0	4.65	48.73%
`tup`	0	4.71	78.61%
`inr`	0	7.89	22.37%
`map`	1	4.79	18.61%
`cut`	1	6.88	24.20%
`cvr`	3	5.49	41.51%
`bpp`	3	6.67	28.24%
`col`	4	1.62	40.08%
`lot`	4	3.64	40.16%
`bif`	4	4.40	35.55%
`rel`	8	6.21	23.47%
`clp`	9	9.57	18.98%

5.2 Analysis of strIPlib sources

Source-level feature vectors were constructed by averaging the feature vectors for all instances from a given source. Source-level comparisons were then made by applying the ISS metric to these feature vectors. As with our previous analysis, all results can be found on the companion website for this paper (`https://icip.optimatorlab.org`). In this section, we provide an analysis of our results in the form of a case study on the `clp` problem type.

Separate composite images were generated for each of the six `clp` sources. These composites, shown in Figure 15, reveal several unique structures. ICIP has, therefore, provided

a strong indication that these problem sources are not based on a common formulation for the container loading problem; this fact which was later corroborated by the maintainers of strIPlib (Bastubbe et al. 2020). Table 8, which contains the average constraint composition for each `clp` source, further supports this claim. It is clear in this table that the constraint compositions vary greatly across the six sources. As an example, 99% of `clpD` constraints are set packing (`PAC`), while no other `clp` source has any set packing constraints at all.



(a) `clpA` (b) `clpB` (c) `clpC`

(d) `clpD` (e) `clpE` (f) `clpF`

Figure 15: Composite images for each of the six `clp` sources.

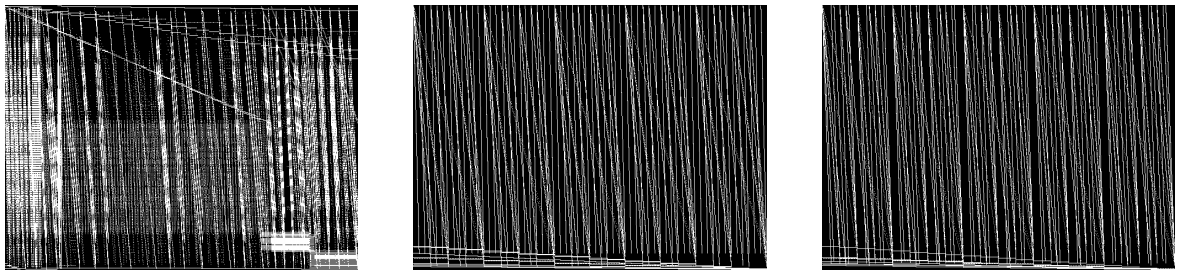
Table 8: Average constraint composition for `clp` instances, by source. An ϵ indicates a nonzero value less than 0.01.

Problem	SIN	PAR	PAC	COV	INV	KPS	INT	MIX	GEN
clpA	0.00	0.01	0.00	0.14	0.00	0.00	ϵ	0.01	0.84
clpB	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00
clpC	0.00	0.01	0.00	0.00	0.00	0.12	0.01	0.13	0.74
clpD	0.00	0.00	0.99	0.00	ϵ	0.00	0.00	0.00	0.00
clpE	ϵ	0.01	0.00	0.16	0.00	0.00	0.00	ϵ	0.82
clpF	0.00	0.01	0.00	0.12	0.12	0.00	0.01	0.01	0.74

Now consider the results of the aforementioned source-level comparisons. The ISS metric indicates that the most similar sources to `clpD` are `gapC` and `gapD` – it shares ISS values of 3.34 and 4.56, respectively, with these sources. The composite images for these three sources reveal a common CCM structure, as can be seen in Figure 16. By considering constraint

compositions at the source level, `gapC` and `gapD` are identified as having 96% and 97% set partitioning (`PAR`) constraints, respectively. Recall that 99% of constraints in the `clpD` instances are set packing (`PAC`) constraints. Moreover, it was the only `clp` source with more than 1% `PAR` *or* `PAC` constraints. These two constraint classes differ only in their sense; `PAC` constraints are inequalities, while `PAR` constraints are strict equalities.

All of these results point to the fact that the particular model used in `clpD` resembles that of a generalized assignment problem. Both the initial discrepancy within the `clp` problem type, and the link from `clpD` to the generalized assignment problem, were identified by the image-based comparison approach.



(a) `clpD` (b) `gatC` (c) `gapD`

Figure 16: Composite images for the `clpD`, `gapD`, and `gatC` instance sources.

5.3 Comparing MIPLIB instances to strIPlib sources

For this final analysis, a subset of 12 instances, defined in Table 9, were downloaded from the MIPLIB. These instances are accompanied by clear problem descriptions. CCM images were generated for these 12 instances, and the pre-trained autoencoder was used to generate feature vectors for these images.

Feature vectors for the 12 MIPLIB instances and for the 64 strIPlib instance sources were combined into a single data set. ISS values were computed between each pair of feature vectors, and 2D coordinates were generated for all 76 vectors. In Figure 17, MIPLIB instances are depicted as orange nodes, while strIPlib sources appear as gray nodes. In both Figures 17a and 17b, edges are drawn exclusively between MIPLIB instances and strIPlib sources

Table 9: MIPLIB Instances

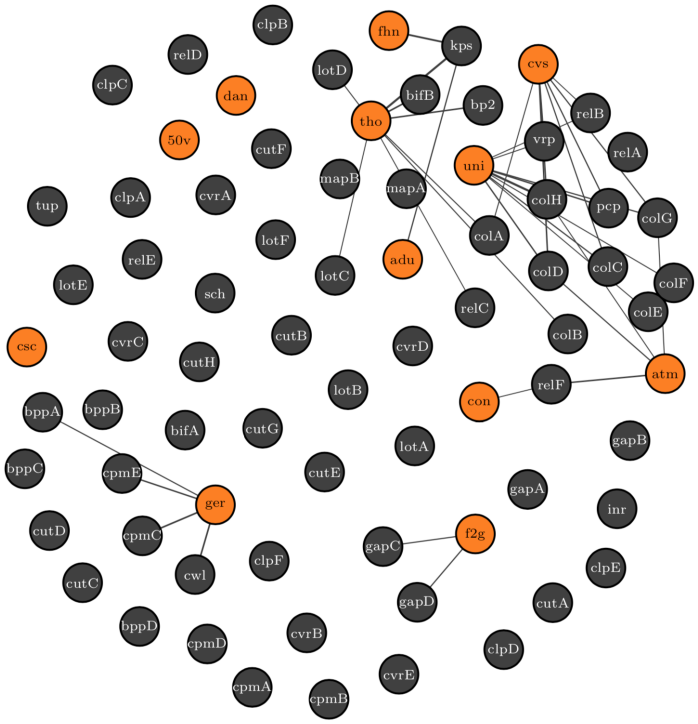
Abbr.	Instance name	Corresponding problem description
50v	50v-10	Network loading problem.
adu	adult-max5features	MIP to create optimized data-driven scoring systems.
atm	atm20-100	ATM cash management problem.
con	control30-5-10-4	Optimal control of a discrete-time switched system model.
csc	csched007	Cumulative scheduling problem.
cvs	cvs16r89-60	Capacitated vertex separator problem.
dan	danoint	Telecommunications application instance.
f2g	f2gap801600	Restriction of well-known hard generalized assignment problem.
fhn	fhnw-schedule-paira400	Continuous-time project scheduling and selection problem.
ger	ger50-17-ptp-pop-6t	Multi-layer network design problem
uni	unitcal_7	California seven day unit commitment problem.
tho	thor50dday	Steiner tree problem in graphs.

according to the specified ISS thresholds.

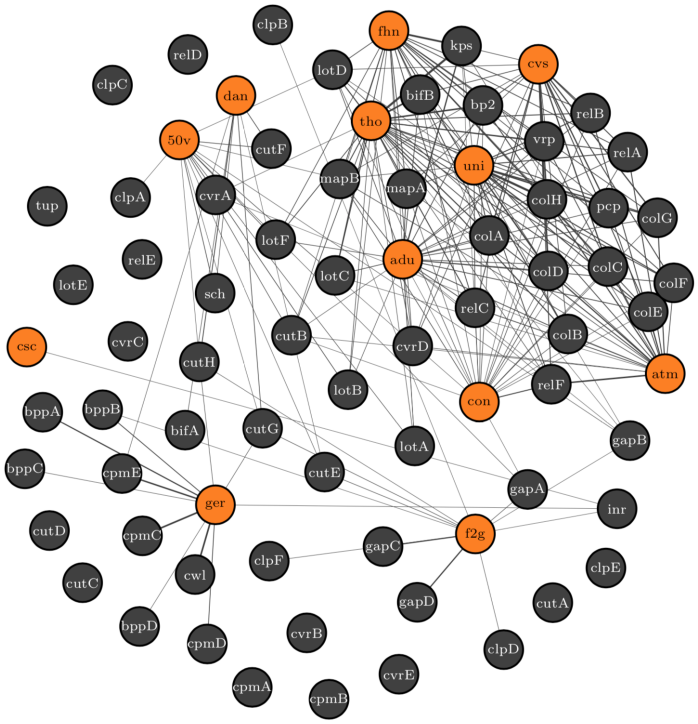
As in Section 5.1, some of the results in this section help to validate the efficacy of ICIP. For example, f2g is an instance of a restricted generalized assignment problem. In Figure 17a, f2g is connected to two sources from the gap - the generalized assignment problem type from strIPlib.

Next, consider ger, an instance of a multi-layer network design problem. This instance shares ISS values of 1.16, 1.18, 1.32, and 1.64 with the cpmC, cwl, cpmE, and bppA sources, respectively. As an aside, cwl, cpm, and bpp are also the most similar *problem types* to ger, in that order. The CCM for ger is presented alongside the composites for its four most similar sources in Figure 18. Interestingly, multi-layer network design has been studied in conjunction with both the facility/warehouse location (Shishebori et al. 2018, Contreras et al. 2012) and p-median (Tang et al. 2019, Contreras and Fernández 2012) problems. The design of airline networks and the location of medical service centers are examples of applications where these problems intersect.

At this point, it is important to note that the CCM images corresponding to the 12 MIPLIB instances were not included in the training set. The trained autoencoder provides

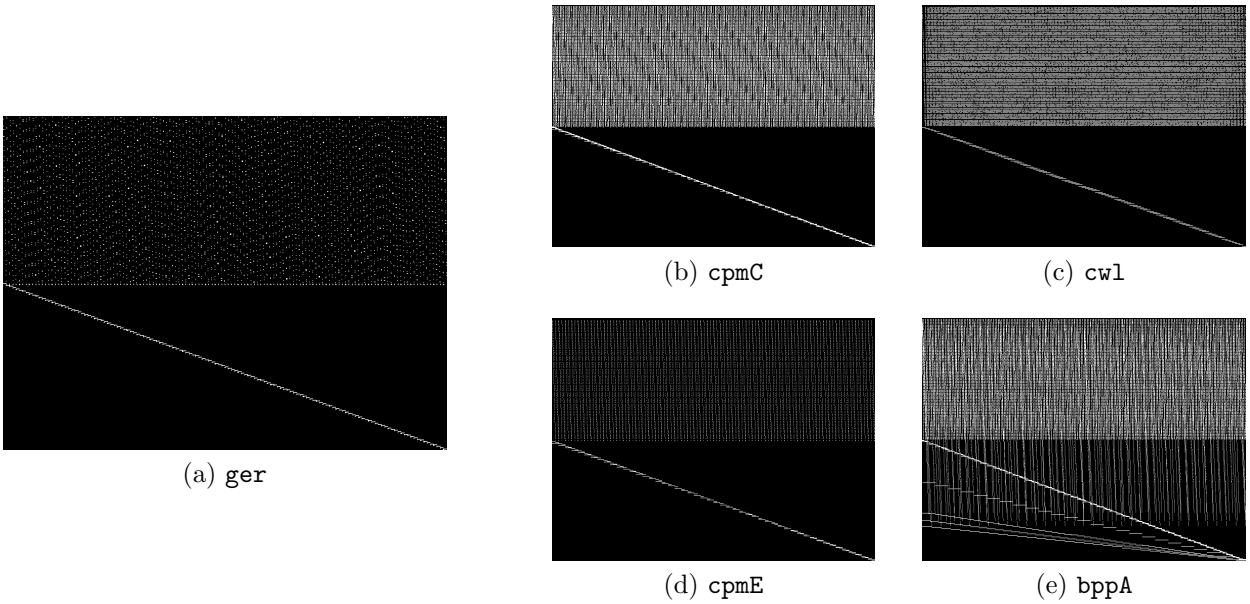


(a) ISS below 2.00



(b) ISS below 4.50

Figure 17: MIPLIB instances and strIPlib sources.



(a) ger (b) cpmC (c) cwl (d) cpmE (e) bppA

Figure 18: The CCM image for ger, an instance from the MIPLIB, is depicted in Figure 18a. Figures 18b-18e contain composite images for its four most similar strIPlib *sources*: cpmC, cwl, cpmE, and bppA.

feature vectors which describe these new CCM images in terms of the features it has learned from the strIPlib instances. Nonetheless, several clear and explainable relationships have been identified. This highlights several important results. First, while the exact meanings of the automatically-learned features are unknown, the efficacy of these features as a basis for MIP comparisons has been demonstrated through our analyses. More generally, given that previously "unseen" instances were effectively described by a previously learned set of features, it may be the case that all MIPs are definable by a unified set of image primatives. While the authors are not making this claim explicitly, we certainly hope that this notion inspires continued research in ICIP.

6 Conclusions and future research

In this paper, CCM structure is explored as a basis for comparing MIP instances and the problems they model. To this end, we propose a methodology for representing CCMs as digital images, encoding those images into feature vectors, and quantifying the similarity

between feature vectors for different images. We term this process Imaged-based Comparison of Integer Programs (ICIP).

The significance of CCM structure in the context of solving MIPs is well established. However, our analyses indicate that CCM structure is also a powerful indicator of similarity between MIPs. These analyses reveal that structurally similar MIPs often share a related underlying problem, even if those problems originate from seemingly disparate applications. These insights are made possible, in part, by the use of automated feature engineering through the application of deep learning techniques. This paper represents the first known use of automated feature engineering to compare MIPs.

Additionally, the results in this paper have raised questions regarding *dissimilarity* between instances previously thought to be related. ICIP both identified a collection of instances in strIPlib which came from different formulations of the same problem, and detected a new problem type with which those instances shared more in common.

Finally, the methodology defined in this paper is flexible in that it may be used to measure similarity between individual instances, collections of instances from a common problem, or combinations of both. In the same vein, this paper presents the first documented analyses of a quantitative measure for problem-to-problem and instance-to-problem similarity.

One key issue to be explored by future research efforts is the rearrangement of CCM rows and columns. GCG proved to be a useful CCM decomposition tool for the scope of the analyses in this paper. However, its decomposition approach is heuristic, and the analyses in this paper operate under the assumption that the decomposition with the greatest maxwhite score is always best. As a result, no claim can be made that the ICIP is invariant to row and column permutations. Moving forward, inspiration may be taken from other fields where matrix row and column permutations are known be an issue (e.g., graph Laplacian matrices).

Another interesting area for future research is in the interpretation of machine-learned features. Although autoencoders are capable of learning features which are effective in image comparison applications, they offer no explanation of what these features mean. It would

be useful to understand how the autoencoder is describing CCM structure, whether this description is similar to one that a human might offer, and what the implications are for modeling and solving MIPs. An improved understanding of these features may facilitate the development of a combined feature set, which leverages both machine-learned and human-defined features to describe problems.

The proposed methodology should also be tested on a larger data set with more problem types. This will require continued efforts by the operations research community to collect and document MIP instances. Additionally, this method can be validated empirically by testing different solution methodologies on related instances.

Other areas for future research include identifying a meaningful ISS threshold, the consideration of additional MIP components (e.g., $\mathbf{x}$, $\mathbf{c}$, and $\mathbf{b}$ in Definition (1)), and other methods for representing model structure. This work provides a foundation for future research in each of these areas, as well as on new methods for comparing and relating works in operations research.

References

Mathematical Programming System/360 Version 2, Linear and Separable Programming – User's Manual. IBM Corporation, White Plains, N.Y., 3 edition, 10 1969. Publication H20–0476–2.

M. Bastubbe, L. Kirchhart, M.E. Lübbecke, N. Rieken, and J.T. Witt. strIPlib: A library of structured integer programs. `striplib.or.rwth-aachen.de`, 2020. Accessed April 20, 2019.

J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *arXiv preprint arXiv:1811.06128*, 2018.

R. Bent and P. Van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515–530, 2004.

G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: a classification scheme and survey. *Top*, 15(1):1–31, 2007.

P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

Z. Camlica, H.R. Tizhoosh, and F. Khalvati. Autoencoding the retrieval relevance of medical images. In *2015 International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pages 550–555. IEEE, 2015.

S. Ceschia, N. Dang, P. De Causmaecker, S. Haspeslagh, and A. Schaerf. The second international nurse rostering competition. *Annals of Operations Research*, 274(1-2):171–186, 2019.

J. Chen, W.K. Cheung, and A. Wang. Learning deep unsupervised binary codes for image retrieval. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 613–619. AAAI Press, 2018.

M. Chen, X. Shi, Y. Zhang, D. Wu, and M. Guizani. Deep features learning for medical image analysis with convolutional autoencoder neural network. *IEEE Transactions on Big Data*, 2017.

X. Cheng, L. Zhang, and Y. Zheng. Deep similarity learning for multimodal medical images. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 6 (3):248–252, 2018.

W.C. Chiang and R.A. Russell. Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Annals of Operations Research*, 63(1):3–27, 1996.

F. Chollet. Keras. `https://keras.io`, 2015.

F.G. Commoner. A sufficient condition for a matrix to be totally unimodular. *Networks*, 3(4): 351–365, 1973.

I. Contreras and E. Fernández. General network design: A unified view of combined location and network design problems. *European Journal of Operational Research*, 219(3):680–697, 2012.

I. Contreras, E. Fernández, and G. Reinelt. Minimizing the maximum travel time in a combined model of facility location and network design. *Omega*, 40(6):847–860, 2012.

K.L. Croxton, B. Gendron, and T.L. Magnanti. A comparison of mixed-integer programming models for nonconvex piecewise linear cost minimization problems. *Management Science*, 49 (9):1268–1273, 2003.

G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8 (1):101–111, 1960.

A. Derdat. Applied Deep Learning – Part 3: Autoencoders, 2013. Accessed July 26, 2019. https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798.

B. Eksioglu, A.V. Vural, and A. Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483, 2009.

T. Elsken, J.H. Metzen, and F. Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

S. Epskamp, A.O.J. Cramer, L.J. Waldorp, V.D. Schmittmann, and D. Borsboom. qgraph: Network visualizations of relationships in psychometric data. *Journal of Statistical Software*, 48(4):1–18, 2012. URL `http://www.jstatsoft.org/v48/i04/`.

R.Z. Farahani and M. Hekmatfar. *Facility Location: Concepts, Models, Algorithms and Case Studies*. Springer, 2009.

K. Fleszar and K.S. Hindi. An effective vns for the capacitated p-median problem. *European Journal of Operational Research*, 191(3):612–622, 2008.

Y. Frota, N. Maculan, T.F. Noronha, and C.C. Ribeiro. A branch-and-cut algorithm for partition coloring. *Networks: An International Journal*, 55(3):194–204, 2010.

S.-W. Fu, T.-W. Wang, Y. Tsao, X. Lu, and H. Kawai. End-to-end waveform utterance enhancement for direct evaluation metrics optimization by fully convolutional neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(9):1570–1584, 2018.

F. Furini, E. Malaguti, and A. Santini. An exact algorithm for the partition coloring problem. *Computers & Operations Research*, 92:170–181, 2018.

G. Gamrath and M.E. Lübbecke. Experiments with a generic Dantzig-Wolfe decomposition for integer programs. In P. Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes*

in Computer Science, pages 239–252, Berlin, 2010. Springer. doi: 10.1007/978-3-642-13193-6_21.

M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 15554–15566, 2019.

A. Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc., 2017.

A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, R.L. Gottwald, G. Hendel, C. Hojny, T. Koch, M.E. Lübbecke, S.J. Maher, M. Miltenberger, B. Müller, M.E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, C. Schubert, F. Serrano, Y. Shinano, J.M. Viernickel, M. Walter, F. Wegscheider, J.T. Witt, and J. Witzig. The SCIP Optimization Suite 6.0. Technical report, Optimization Online, July 2018. URL `http://www.optimization-online.org/DB_HTML/2018/07/6692.html`.

A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. Christophel, K. Jarck, T. Koch, J. Linderoth, M.E. 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. July 2019a.

A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P.M. Christophel, K. Jarck, T. Koch, J. Linderoth, M.E. Lübbecke, H.D. Mittelmann, D. Ozyurt, T. K. Ralphs, D. Salvagnin, and Y. Shinano. MIPLIB 2017. 2019b. http://miplib.zib.de.

G.D. Glockner and G.L. Nemhauser. A dynamic network flow problem with uncertain arc capacities: Formulation and problem structure. *Operations Research*, 48(2):233–242, 2000.

H.W. Hamacher and S. Nickel. Classification of location models. *Location Science*, 6(1-4):229–242, 1998.

S. Haspeslagh, P. De Causmaecker, A. Schaerf, and M. Stølevik. The first international nurse rostering competition 2010. *Annals of Operations Research*, 218(1):221–236, 2014.

A. Hyvärinen and P. Hoyer. Emergence of phase-and shift-invariant features by decomposition

of natural images into independent feature subspaces. *Neural computation*, 12(7):1705–1720, 2000.

K.L. Jones, I.J. Lustig, J.M. Farvolden, and W.B. Powell. Multicommodity network flows: The impact of formulation on decomposition. *Mathematical Programming*, 62(1-3):95–117, 1993.

D.G. Kendall. Some problems in the theory of queues. *Journal of the Royal Statistical Society: Series B (Methodological)*, 13(2):151–173, 1951.

W.-Y. Ku and J.C. Beck. Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & Operations Research*, 73:165–173, 2016.

A.B.L. Larsen, S.K. Sønderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015. https://arxiv.org/abs/1512.09300.

Z. Li, X. Zhang, H. Müller, and S. Zhang. Large-scale retrieval for medical image analytics: A comprehensive review. *Medical Image Analysis*, 43:66–84, 2018.

X. Liang and Y. Xiao. Game theory for network security. *IEEE Communications Surveys & Tutorials*, 15(1):472–486, 2013.

W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

V. Maniezzo, A. Mingozzi, and R. Baldacci. A bionomic approach to the capacitated p-median problem. *Journal of Heuristics*, 4(3):263–280, 1998.

H. Marchand, A. Martin, R. Weismantel, and L. Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1-3):397–446, 2002.

R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004.

G.L. Nemhauser, M.W.P. Savelsbergh, and G.C. Sigismondi. *Constraint classification for mixed integer programming formulations*, volume 20. COAL Bulletin-Committee on Algorithms of Mathematical Programming Society, 1992.

M. Ohlsson, C. Peterson, and B. Söderberg. Neural networks for optimization problems with inequality constraints: the knapsack problem. *neural computation*, 5(2):331–339, 1993.

J. Ostrowski, M.F. Anjos, and A. Vannelli. Tight mixed integer linear programming formulations for the unit commitment problem. *IEEE Transactions on Power Systems*, 27(1):39–46, 2011.

J. Qin, E. Haihong, M. Song, and Z. Ren. Image retrieval based on a hybrid model of deep convolutional encoder. In *2018 IEEE International Conference of Intelligent Robotic and Control Engineering (IRCE)*, pages 257–262. IEEE, 2018.

J. Racine. gnuplot 4.0: a portable interactive plotting utility. *Journal of Applied Econometrics*, 21 (1):133–141, 2006.

N. Rahim, J. Ahmad, K. Muhammad, A.K. Sangaiah, and S.W. Baik. Privacy-preserving image retrieval for mobile devices with deep features on the cloud. *Computer Communications*, 127: 75–85, 2018.

M. Reimann, K. Doerner, and R.F. Hartl. Insertion based ants for vehicle routing problems with backhauls and time windows. In *International Workshop on Ant Algorithms*, pages 135–148. Springer, 2002.

J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the fisher vector: Theory and practice. *International journal of computer vision*, 105(3):222–245, 2013.

S. Sharma, I. Umar, L. Ospina, D. Wong, and H.R. Tizhoosh. Stacked autoencoders for medical image search. In *International Symposium on Visual Computing*, pages 45–54. Springer, 2016.

D. Shishebori, A.Y Babadi, and Z. Noormohammadzadeh. A lagrangian relaxation approach to fuzzy robust multi-objective facility location network design problem. *Scientia Iranica. Transaction E, Industrial Engineering*, 25(3):1750–1767, 2018.

F.Y.-P. Simon. Integer linear programming neural networks for job-shop scheduling. In *IEEE 1988 International Conference on Neural Networks*, pages 341–348. IEEE, 1988.

J.E.S. Sklan, A.J. Plassard, D. Fabbri, and B.A. Landman. Toward content-based image retrieval with deep convolutional neural networks. In *Medical Imaging 2015: Biomedical Applications in Molecular, Structural, and Functional Imaging*, volume 9417, page 94172C. International Society for Optics and Photonics, 2015.

F. Stefanello, O.C.B. de Araújo, and F.M. Müller. Matheuristics for the capacitated p-median problem. *International Transactions in Operational Research*, 22(1):149–167, 2015.

X. Tang, F. Lehuédé, O. Péton, and L. Pan. Network design of a multi-period collaborative distribution system. *International journal of machine learning and cybernetics*, 10(2):279–290, 2019.

M.A. Trick, H. Yildiz, and T. Yunes. Scheduling major league baseball umpires and the traveling umpire problem. *Interfaces*, 42(3):232–244, 2012.

K. Tsuchiya, S. Bharitkar, and Y. Takefuji. A neural network approach to facility layout problems. *European Journal of Operational Research*, 89(3):556–563, 1996.

J.P. van den Berg and W.H.M. Zijm. Models for warehouse management: Classification and examples. *International Journal of Production Economics*, 59(1-3):519–528, 1999.

F. Xie, C.N. Potts, and T. Bektaş. Iterated local search for workforce scheduling and routing problems. *Journal of Heuristics*, 23(6):471–500, 2017.

L. Xue, Z. Luo, and A. Lim. Two exact algorithms for the traveling umpire problem. *European Journal of Operational Research*, 243(3):932–943, 2015.

Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.

W. Zhou, H. Li, and Q. Tian. Recent advance in content-based image retrieval: A literature survey. *arXiv preprint arXiv:1706.06064*, 2017. https://arxiv.org/abs/1706.06064.

Appendix

The analyses presented in this paper examine a collection of 950 MIP instances from strIPlib. These instances were downloaded from 64 different strIPlib sources. Table 10 contains the abbreviations, paths, and number of instances downloaded for each of these sources.

Table 10: Abbreviations and directory paths for all 64 sources.

Abbr.	Num. Instances	Path
bppA	13	bpp/bison/eq
bppB	13	bpp/bison/gq
bppC	12	bpp/orlibrary/eq
bppD	12	bpp/orlibrary/gq
bp2	50	bpp2/orbologna/gq
bifA	25	bppif/ceselli/chain
bifB	25	bppif/ceselli/natural
clpA	9	clp/chen/chen
clpB	9	clp/fasano/fasano
clpC	8	clp/hifi/hifi
clpD	8	clp/junqueira/junqueira
clpE	8	clp/paquay/paquay
clpF	8	clp/tsai/tsai
colA	4	coloring/fixedsets/bigM
colB	2	coloring/fixedsets/disaggregiert
colC	8	coloring/mixed/bigM
colD	8	coloring/mixed/disaggregiert
colE	7	coloring/orlibrary/bigM
colF	7	coloring/orlibrary/disaggregiert
colG	7	coloring/r-set/bigM
colH	7	coloring/r-set/disaggregiert
cpmA	2	cpmp/lorena/eq
cpmB	2	cpmp/lorena/gq
cpmC	15	cpmp/optlab/eq
cpmD	15	cpmp/optlab/gq
cpmE	16	cpmp/orlibrary/eq
cutA	7	cuttingstock/18classes/eq
cutB	7	cuttingstock/18classes/gq
cutC	6	cuttingstock/hard28/eq
cutD	6	cuttingstock/hard28/gq
cutE	6	cuttingstock/schwerin/eq
cutF	6	cuttingstock/schwerin/gq
cutG	6	cuttingstock/waescher/eq
cutH	6	cuttingstock/waescher/gq
cvrA	8	cvrp/asymmetric/default
cvrB	12	cvrp/augerat/default
cvrC	9	cvrp/ce-vrp/default
cvrD	13	cvrp/symmetric/default
cvrE	8	cvrp/tsplib/default
cwl	50	cwlp-ss/sobolev/default
gapA	12	gap-tmp/yagiura/max
gapB	12	gap-tmp/yagiura/min
gapC	13	gap/orlibrary/max
gapD	13	gap/orlibrary/min
inr	50	inr/inrc/default
kps	50	kps/furini/default
lotA	9	lotsizing/derstroff/default
lotB	9	lotsizing/derstroff/setups
lotC	8	lotsizing/lotsizelib/default
lotD	8	lotsizing/pimentel/default
lotE	8	lotsizing/surie/default
lotF	8	lotsizing/trigeiro/default
mapA	25	maplabeling/lorena/alternative
mapB	25	maplabeling/lorena/default
pcp	50	pcp/furini/default
relA	9	relaxedClique/gschwind/gamma-quasi-clique_covering
relB	9	relaxedClique/gschwind/gamma-quasi-clique_partitioning
relC	8	relaxedClique/gschwind/s-clique_covering
relD	8	relaxedClique/gschwind/s-club_covering
relE	8	relaxedClique/gschwind/s-club_partitioning
relF	8	relaxedClique/gschwind/s-plex_covering
sch	50	scheduling/orlibrary/default
tup	50	tup/kuleuven/default
vrp	50	vrptw/sourceunknown/default