Graph Neural Network Comparison for 2D Nesting Efficiency Estimation

Corentin Lallier, Guillaume Blin, Bruno Pinaud, Laurent Vézard

Graph Neural Network Comparison for 2D Nesting Efficiency Estimation

Corentin Lallier[1,2*], Guillaume Blin[1], Bruno Pinaud[1] and Laurent Vézard[2]

[1]Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, 351, cours de la Libération, Talence cedex, 33405, France.
[2]Lectra, 23 Chem. de Marticot, Cestas, 33610, France.

Contributing authors: corentin.lallier@u-bordeaux.fr;

Abstract

Minimizing the level of material consumption in textile production is a major concern. The cornerstone of this optimization task is the nesting problem, whose goal is to lay a set of irregular 2D parts out onto a rectangular surface, called the nesting zone, while respecting a set of constraints. Knowing the efficiency –ratio of usable to used up material enables the optimization of several textile production problems. Unfortunately, knowing the efficiency requires the nesting problem to be solved, which is computationally intensive and has been proven to be *NP*-hard. This paper introduces a regression approach to estimate efficiency without solving the nesting problem. Our approach models the 2D nesting problem as a graph where the nodes are images derived from parts and the edges hold the constraints. The method then consists of combining convolutional neural networks for addressing the image-based aspects and graph neural networks (GNNs) for the constraint aspects. We evaluate several neural message passing approaches on our dataset and obtain results that are sufficiently accurate for enabling several business use cases, where our model best solves this task with a mean absolute error of **1.65**. We provide open access to our dataset, whose properties differ from those of other graph datasets found in the literature. This dataset is constructed on **100,000** real customers' nesting data. Along the way, we compare the performance and generalization capabilities of four GNN architectures obtained from the literature on this dataset.

Keywords: Fashion Manufacturing, 2D Bin packing, Machine Learning, Graph Neural Networks

1 Introduction

Textiles are everywhere! Several industries ranging from fashion (clothes, shoes, and accessories) to the automotive (seats, interiors, and airbags) and furniture (sofas and chairs) industries make considerable use of them. In the context of Industry 4.0 and smart manufacturing (Oztemel and Gursev, 2020; Wang et al, 2021), the optimization of resources and thus the minimization of material consumption is a major concern, formerly for economic reasons and more recently for environmental considerations (Rissanen, 2013; Henninger et al, 2016). The cornerstone of material optimization is the nesting phase. Nesting, illustrated in Figure 1, is a problem that is closely related to strip packing; however, some constraints must be respected in the output layout. The goal of nesting is to lay a set of irregular 2D parts, derived from computer-aided design, out onto a surface called the nesting zone by minimizing the occupied space (and material consumption). The constraints may concern a part itself (*e.g.*, which rotations are allowed for that part), its relation to other parts (*e.g.*, two parts must be close to each other), or the target fabric (*e.g.*, alignment with the fabric pattern). The nesting efficiency is the percentage of usable material (the parts placed) compared to the whole amount of used up material.

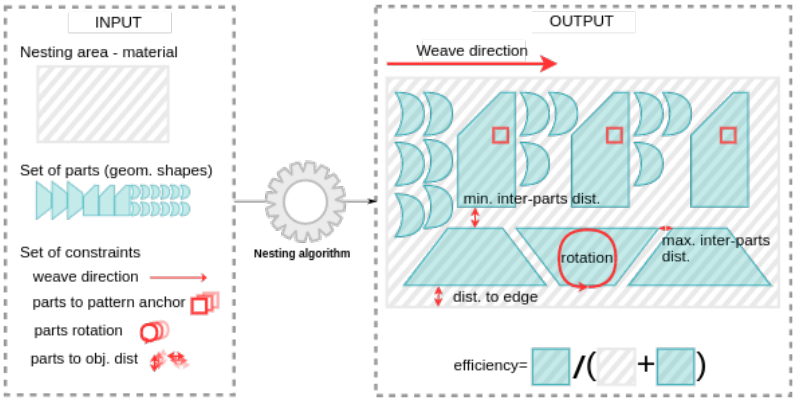

Fig. 1 Nesting algorithm: the rectangular nesting area (*i.e.*, the material sheet) is in light gray. The diagonal gray lines on the area figure the material pattern. Parts are depicted in green with a solid border, and examples of constraints are in red (arrows and squares). A nesting algorithm takes as inputs a set of parts, a set of constraints applied on the parts, and a nesting area where the parts have to be placed. Formally speaking, the output of such an algorithm is a position and a rotation angle for each part, with respect to the constraints and the nesting area. A 2D geometric view can be reconstructed using position and rotation of the parts, then the global efficiency can be computed.

Beyond the nesting phase itself, knowing the efficiency for a set of parts and constraints may be used to optimize the material consumption level in various

phases: from design (shape optimization) to planning and even to postproduction analysis. The planning phase allows us to decide how to group several items (*e.g.*, 3,000 shirts in small and medium sizes) into batches to be cut on several layers of fabrics. This activity, called *section planning*, is a combinatorial optimization problem where we have items, *e.g.*, the shirts' parts, to be placed on a resource, *i.e.*, the fabric sheets. The problem involves minimizing the number of sheets used to produce the shirts while simultaneously maximizing the number of material layers to be cut. It requires knowing the efficiency levels of the part combinations to find the batch of parts with the best possible global efficiency. Unfortunately, the class of 2D packing problems is computationally intensive. Li and Milenkovic (1995) proved it to be NP-hard, even with rectangles. Hence, nesting times, even with small sets of parts, are between a few minutes and a few hours depending on the utilized nesting algorithm and the efficiency expectation. Under these conditions, it is not viable to compute thousands of nesting operations for such optimization tasks.

A common practice is to rely on *efficiency tables* empirically built with representative nesting cases to estimate efficiency. For instance, such tables might state that performing section planning for basic shirts with a mix of small and medium sizes leads on average to an efficiency of 75%. In addition to being time-consuming to build, efficiency tables, while helpful, prevent us from reaching the most efficient combinations. For all these reasons, a new estimation method is needed. In this context, we propose a novel approach for efficiency estimation based on machine learning (ML).

The use of ML in the context of combinatorial optimization problems has gained some attraction in recent years (Bengio et al, 2021; Popescu et al, 2021), especially with deep learning. For example, graph neural networks (GNNs) and reinforcement learning have been used to optimize chip placement (Mirhoseini et al, 2021), which is close to our use case. Our work follows this path and proposes an original way to solve the efficiency prediction problem by modeling nesting data as a graph, where the nodes are image-based representations of the different parts and the edges are the constraints between parts. The efficiency is the result of regression on a vector-based embedding of the whole graph.

Contributions

To the best of our knowledge, this is the first work to propose a deep learning approach for estimating the efficiency of a 2D nesting problem through the modeling of graphs with complex relations. Our general aim is to improve the performance achieved in efficiency-dependent optimization tasks (previously introduced), such as design and section planning tasks. By combining convolutional neural networks for image-based aspects and a GNN for relationship aspects, our approach aims to estimate nesting efficiency without actually performing nesting. We solve this task with a mean absolute error (MAE) of 1.65, which permits us to unlock business usages. We also provide our original nesting dataset for the community as a benchmark for this efficiency prediction task or other regression tasks.

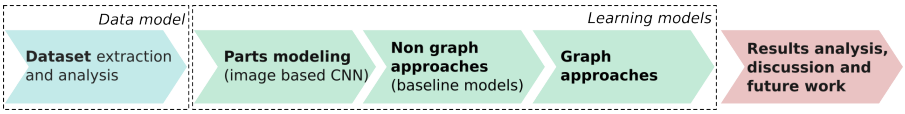

Fig. 2 Flowchart of the main steps of our work

Figure 2 presents the main steps of our work. The first step concerns dataset creation, specifically collecting, processing and exploring the raw data for the given problem, to form a first data model. This is followed by the creation of several deep learning models, which can be separated into three steps: first, parts representations, nongraph approaches, and graph approaches. The last step involves analyzing the results in relation to the planned future work.

In the remainder of this paper, we first provide some background knowledge on ML and GNNs in Section 2. Our dataset is described in Section 3. Our approach is described in Section 4, and it is experimentally evaluated in Section 5.

2 Related work

Below, we present the related approaches concerning the use of ML in optimization, an overview of the main GNN models, the use of attention in GNNs, and finally the recent field of edge information integration involving GNNs.

2.1 ML in combinatorial problems

Nesting is a notorious combinatorial optimization problem (Bennell and Oliveira, 2009), which is usually solved by applying heuristics or metaheuristic functions such as simulated annealing (Oliveira and Ferreira, 1993; Gomes and Oliveira, 1999), construction via bottom-left placement (Dowsland et al, 2002) or genetic algorithms (Goodman et al, 1994; Liu and He, 2006) to obtain a pseudo-optimal solution. The use of ML in combinatorial optimization problems has attracted attention in recent years (Bengio et al, 2021; Popescu et al, 2021), especially with the development of deep learning. ML fulfills different purposes depending on the utilized approach. Notably, it can be used to directly find the solution of a problem (Soong, 2015). Nonetheless, it can also be used, for example, to select an appropriate algorithm for finding a solution or to help a given algorithm converge faster toward a solution by providing a useful piece of information about the problem or a state.

In the context of placement, reinforcement learning has been used to improve chip layouts (Mirhoseini et al, 2021) or to solve the regular 2D packing problem (Soong, 2015). However, we did not find any approaches that specifically solve the nesting problem with ML. The closest work to that in this paper is the research of Xu et al (2020) who proposed an approach estimating the length of a marker produced by nesting with ML. This can be considered equivalent to efficiency prediction; however, their approach takes only the

lengths of the pieces into account and does not generalize to any garment. In contrast, our approach is applicable to any garment.

2.2 Graph Neural Networks

A GNN (Scarselli et al, 2009) is a general neural network (NN) architecture defined according to a graph structure that permits representational learning on graphs by taking the underlying relationships among data and graph structural information into account. Since then, GNNs have attracted interest because many problems can be naturally modeled with graphs. The use of more traditional yet well-known ML approaches on this kind of graph data implies that destructive transformations are required.

Kipf and Welling (2017) introduced a graph convolutional network (GCN) as a generalization of a convolutional network architecture on graphs using node features. Alongside this approach, Gilmer et al (2017) reformulated existing models using a common message passing framework into the so-called message passing NN. In this architecture, messages are exchanged along edges by following a neighborhood *aggregation function*, thus producing node representations. Then, the node representations are passed throughout a graph-level *readout function* to produce a vectorized graph representation. These different levels of representations permit node-, edge- or graph-level operations, for instance, node property prediction, missing edge creation, or graph property prediction. In our case, we use graph-level representations to predict a graph-level property: efficiency.

Schlichtkrull et al (2018) designed an extension of a GCN for dealing with multiple types of relationships in data by using the message passing framework referred to as relational GCN (R-GCN). More recently, graph isomorphism network (Xu et al, 2019) was designed as a generalization of the aggregation function using a multilayer perceptron instead of a single-layer perceptron, and this approach showed that a sum aggregator over all layers is more expressive as a readout function, than the other standard (mean, max) operators.

The graph isomorphism network model focuses on standard aggregators and leaves others, such as the attentional weighted sum, unexplored. In this paper, we focus on such attentional operators in GNNs (presented next) as we believe that they preserve the variability of data distributions better than standard operators.

2.3 Attention-based GNNs

Attention is a mechanism that extends the range of possible ML method input data by allowing the use of input data with variable sizes to produce vector-based embeddings. It exploits the similarities between the input data, called attention scores, which are linearly combined with the current embeddings to produce new embeddings. A review of similarity functions is given in Chaudhari et al (2021).

In the case of *self-attention*, the scores are based on the similarities between all elements of the input data. One can see it as a $n \times n$ self-similarity matrix, where n is the size of the input dataset. In *cross-attention*, the scores are computed between two distinct sets of input data (using a $n \times m$ similarity matrix, with m being the size of the second input dataset). Soft (or global) attention (Luong et al, 2015) is a specific case of cross-attention where one of the input sets has a size of 1 (*i.e.*, a $n \times 1$ matrix). Attention has gained much interest in NNs since the development of the transformer model (Vaswani et al, 2017), primarily in the field of natural language processing, then on image data with vision transformers (Dosovitskiy et al, 2021) and in more general industrial applications (Mo et al, 2021).

The gated graph sequence NN (Li et al, 2017) first applies attention to graphs using GNNs. Given a set of graph-level tasks, this method produces sequences and uses an attention mechanism that weights nodes according to their relevance to the current task. The application of attention at the node level was formalized in graph attention network (Veličković et al, 2018) with self-attention between neighboring vertices. Self-attention between nodes is also used in self-attention graph pooling (SAGPool) (Lee et al, 2019) to downsample or prune graph nodes according to their scores. Graph attention has also been used in graph transformers networks (GTNs) (Yun et al, 2019; Dwivedi and Bresson, 2021) to generate node embeddings.

These GNN methods mainly focus on producing node embeddings. As seen previously, in our approach, edges represent nesting constraints, and edge attributes encode constraint parameters. We now review how edge information can be used in GNN models.

2.4 GNNs with edge features

In GNNs, three levels of edge information can be used. The first relies on edges' weights by using a weighted adjacency matrix. The second deals with edge types. Edge features are vectors of integer values that can be represented by an integer-filled adjacency matrix, as in an R-GCN (Schlichtkrull et al, 2018) or GNN- featurewise linear modulation (FiLM) (Brockschmidt, 2020). Depending on the types of edges different transformations are applied to the nodes. The third and the most general level concerns multidimensional edge features, where each edge is represented by a vector. This can be seen as a three-dimensional adjacency matrix of shape $N \times N \times E$, where N is the number of nodes, and E is the dimensionality of the edge vector features. In message passing GNNs (Gilmer et al, 2017), the edge features can be passed directly in the message function with the source and target node embedding. This principle is also used in the crystal GCN (Xie and Grossman, 2018). These models use node and edge features to build node embeddings, as in SchNet (Schütt et al, 2017). The gated GCN (Bresson and Laurent, 2018) updates node and edge embeddings using edge gates. Another approach is to use edge features to build edge embeddings, as in edge GNNs (Gong and Cheng, 2019)

or GTNs (Dwivedi and Bresson, 2021) by combining self-attention with the node and edge embeddings.

In this paper, we compare nonattentional message passing architectures with an attention-based architecture for different edge information integration methods. In the remaider of the paper, we compare an R-GCN (nonattentional, one-dimensional integer edge type), the CF-GCN from SchNet (nonattentional, multidimensional edge features, only node embeddings are updated), a gated GCN (soft attention, multidimensional edge features, the node and edge embeddings are updated) and a GTN (self attention, multidimensional edge features, the node and edge embeddings are updated). The next section presents our general process and architecture, first by modeling the input data and then utilizing our general GNN approach.

3 Dataset

Several open nesting datasets exist (ESICUP, 2021), but we do not use them, as they do not fit our needs because 1) they lack variability in the input data (low part cardinality and no or few constraints), 2) they possess a low number of nesting instances for building a learning dataset and 3) they contain no prediction variables (their efficiency is generally unknown). We thus propose a new dataset to achieve improved performance on the efficiency prediction task.

This dataset is composed of 100,000 nesting tasks performed by about a hundred of our fashion customers with the same nesting algorithm. This dataset is freely available on Zenodo[1]. The nesting time is constant (5 minutes) for all tasks. The use of real customer data permits us to form a large dataset distribution for evaluating our approach. The 100,000 nesting tasks are composed of 7.4 million parts referencing 400,000 distinct shapes and 7.6 million constraints.

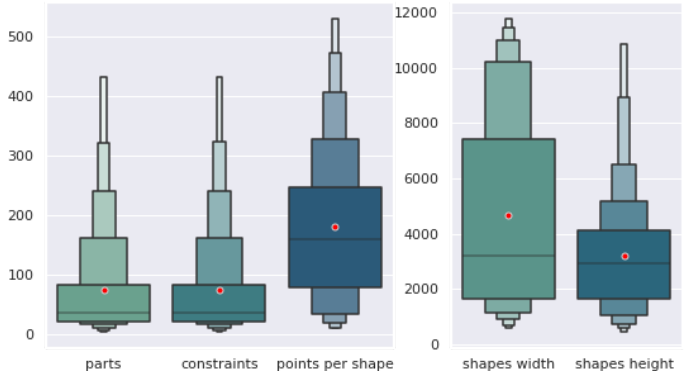

Fig. 3 Dataset distributions. From left to right: number of parts per task, constraints per task, and number of points, width, and height (in m^{-4}) per shape.

[1] https://doi.org/10.5281/zenodo.6610253

The distribution of parts and the constraint count per task are presented in Figure 3, as along with the numbers of points, width, and height descriptors. On average (the red dots in Figure 3), there are 74 parts and 75 constraints per graph, and the parts are also defined by a mean of 184 points, a width of 468 mm and a height of 320 mm.

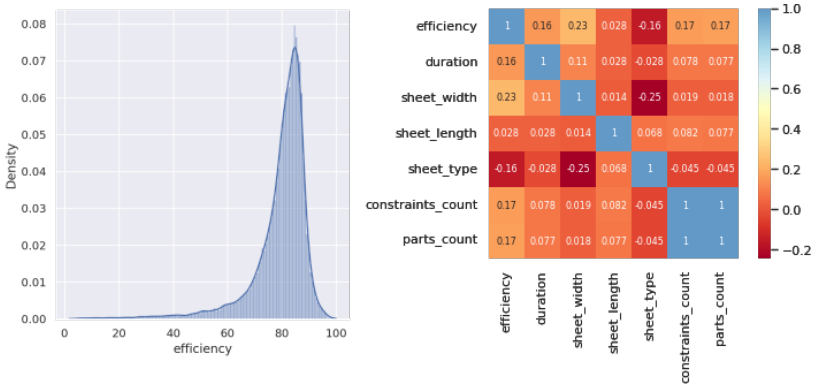

Fig. 4 Left: efficiencies distribution in the dataset. Right: main variables Pearson correlation matrix (efficiency, duration, sheet width, height and type, constraints and parts count). Detailed distributions are shown in Figure A1 of the Appendix.

Each nesting point is labeled with its computed efficiency. The efficiency distribution is shown in Figure 4 (79.2% on average with a standard deviation of 10.9%). The main descriptors of a nesting task are its duration, nesting zone (sheet width and sheet length), type and numbers of constraints and parts composing each task. Figure 4 shows no obvious correlation between any of these variables and efficiency.

4 Our approach

Let us recall that the input data of a nesting problem are 2D parts' shapes along with their associated constraints and a nesting area, as described in Figure 1. One challenge is the relational nature of these data. Constraints may relate to 1) two different parts (*e.g.*, the distance between them), 2) a part property (*e.g.*, rotations), or 3) a part relative to the nesting area (*e.g.*, distances to edge borders).

Therefore, we choose to model nesting problems as graphs, where nodes are parts, edges are constraints and edge features are constraint parameters. Our task is then a regression problem, where a graph representation must be associated with an expected efficiency level. We choose to use GNNs to extract graph representations, and several GNN models are compared in terms of the efficiency estimation performance achieved on our dataset. In the next section, we present the data modeling approach and the deep learning-based regression pipeline. This pipeline is summarized in Figure 5 and developed in the next

section with a focus on the graph data model and feature engineering; then, the message passing phase and the readout are provided and finally, the regression phase is described.

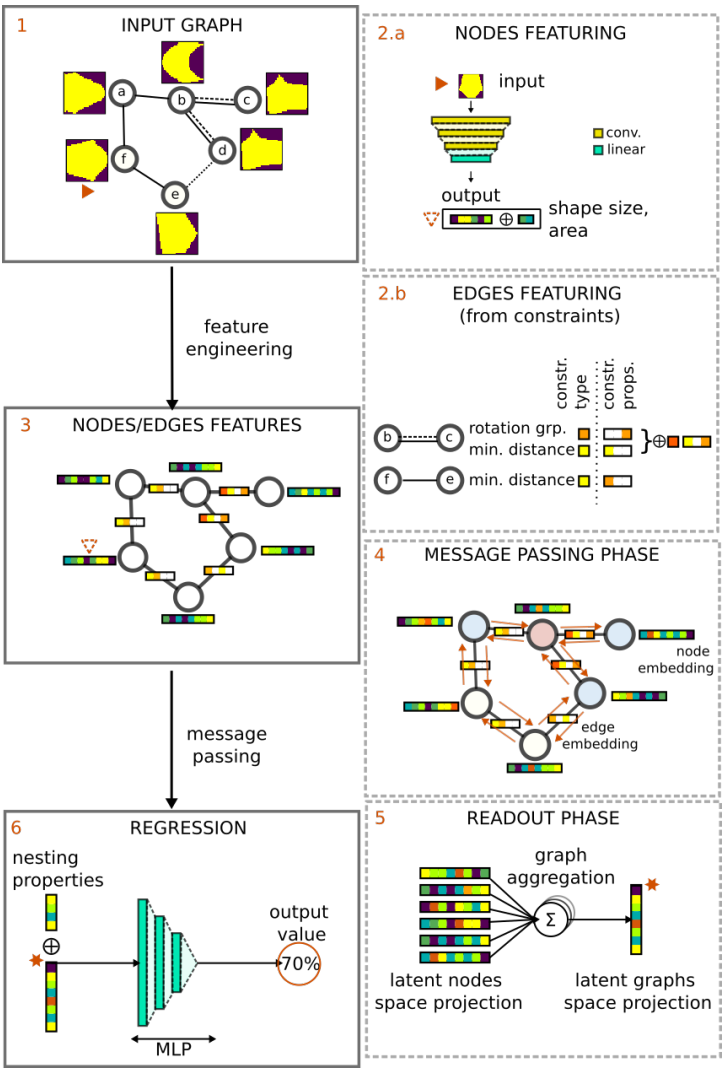

Fig. 5 General process and architecture. From an input graph (1), node features are built from the latent image space projection of an auto-encoder (2.a), and edge features are handcrafted (2.b, Section 4.1). They are both merged in (3) as inputs for (4) where node and edge embeddings are updated. Node embeddings are passed in the readout function to generate the graph embedding (5, Section 4.2). Eventually, the decoder uses the graph embedding to regress to a single output value, that is the efficiency level (6, Section 4.3).

4.1 Graph modeling

To model our input data as a graph, we propose 1) using part images to compute node-level features, 2) using constraint properties to compute edge-level features, and 3) using high-level nesting properties, *e.g.*, area dimensions, as graph-level features.

Node-level features are representations of the parts. They aim to capture the geometric representation of these parts (Figure 5 2.a). We discretize the parts' shapes into binary images with a fixed dimensionality. Then, the images are fed into a convolutional NN-based encoder to generate embeddings.

Fig. 6 Left: the convolutional NN image encoder. Right: the decoder used in the image reconstruction task

This LeNet-like (Lecun et al, 1998) convolutional NN architecture is described in Figure 6 and is validated in a side task consisting of reconstructing the input images (illustrated in Figure 7). The autoencoder is trained on an image reconstruction task using a mean squared error (MSE) loss.

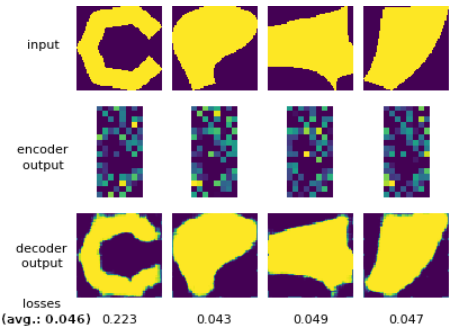

Fig. 7 Examples of image reconstruction tasks used to optimize and validate the convolutional NN encoder. Top: input images, center: encoder output, bottom: reconstructed images used to compute the loss. The encoder is then used as a convolutional NN to generate node embeddings

The width, height, and area of each of the original parts, lost during the discretization process, are concatenated to the embeddings to generate node features.

Edge-level features are representations of constraints. Our objective is to transform heterogeneous data from their original constraint parameters into normalized input edge features. The more predominant constraints are rotations, groups, and pattern alignments. Rotation constraints define rotation

ranges in degrees (*e.g.*, [−45; 45]). There is always one rotation constraint for each part, which is designed as a self-connection on the node representing that part. Group constraints define sets of parts that share a property. For instance, a group can share a position interval on one axis. They are designed by the Cartesian product of the connections between the parts in the group. Pattern alignments describe how two parts must be aligned according to their material patterns. This constraint and the remaining constraints are represented by one-to-one relations.

To properly use message passing, multiple edges are merged into one with a normalized vector of features (see Figure 5 2.b). Moreover, variables with ranges (*e.g.*, a rotation in [−45; 45]) are binarized in fixed subranges to have a constant number of attributes. The resulting vector features are sparse, only the relative attributes of the constraint are filled, and the others are left as 0.

Graph-level features are high-level information regarding the nesting problem. The width, height, and constraints of the nesting area relative to the nesting zone (*e.g.*, the *exclusion zone*) are used as graph features. *Exclusion zones* are represented as low-definition binary images inside the graph feature vector.

In the next section, we present the global GNN architecture we use to estimate the efficiency based on these extracted features.

4.2 Message passing phase

Our objective is to build high-level representations of the graph to regress from the graph-level representation to a value (the efficiency). To this end, we use a message passing framework (Gilmer et al, 2017), consisting of two phases: message passing and readout (Steps 4 and 5 in Figure 5, respectively). Message passing runs for T time steps, which are implemented as layers. During this phase, the hidden states of each node/edge in the graph are updated based on exchanged messages. We propose and compare several GNN architectures in Section 5.

The readout phase computes an embedding for the entire graph. As described in Figure 5, Step 5, the node embeddings are aggregated to obtain the final graph-level feature vector to be passed to the regression module. The readout module is implemented by a sum aggregation function, as in Xu et al (2019). The last part of the process is the regression module presented below.

4.3 The regression module

This module transforms a graph embedding into a single value (the nesting efficiency). This is done with a pyramidal multilayer perceptron with one output cell, which is defined as

$$y = W_3.\sigma(W_2.\sigma(W_1 x)) \tag{1}$$

where y is the output of the network, *i.e.*, the estimated efficiency. W_1, W_2 and W_3 are learnable projections of size $x \times 2h$, $2h \times h$ and $h \times 1$, respectively

where x is the size of the input graph embedding. It is built from the output of the readout function concatenated with the graph-level features (see Figure 5, Step 6). h is the hidden layer size. σ is an activation function.

In the next section, we present the experimental process followed and the different tested message passing methods.

5 Experiment

In this section, we first present the implementation of our benchmark, and then we provide the experimental protocol and our results.

For all experiments, the task input data are obtained at node, edge, and graph levels. We discretize the input shapes into (64×64) binary images to have a fixed dimensionality. The convolutional NN output is an image embedding with 128 dimensions. The nodes are then described by 131 features: 128 for the shape image, plus the width, height, and area. The input edge data are represented by 44 features: 13 binary features for the constraint types and, source and target nodes indices, and 31 for describing the constraints features (*e.g.*, 4 for rotations, 2 for x/y position offsets, *etc.*). The input graph data are described by 87 features: 6 for the main problem variables (duration, nesting width/height, nesting type, constraints, and part count), 2 for the nesting folding y min/max, 64 for the binary rasterized exclusion zone, and 15 for the constraint counts grouped by type. All the features are standardized (the mean is removed, and the variance is scaled to 1).

As described in Section 4, the global process is based on four main steps. First, the input shape images are fed into a convolutional NN and then combined with the image width, height, and area to create node features. Second, according to each model, the nodes and edges are updated by a message passing phase. Third, the resulting node embeddings are fed into the readout function. The output of the readout, with 128 dimensions, is concatenated with the 87 graph-level features to form a graph embedding possessing 215 features and then finally fed to the 3-layer regression module. Each layer of this module has shapes of (215×256), (256×128) and (128×1). The activation functions are rectified linear unit (ReLU) functions except for the final activation function used for the loss calculation, which is a sigmoid function. The final output is the estimated efficiency.

The models compared in our benchmark can be split into two groups: baseline models and graph-based models. On the one hand, the baseline models do not use the graph paradigm. On the other hand, in the GNN group, the models use constraint parameters as edge information and a graph convolution function as the message passing phase. The implementations are based on PyTorch (Paszke et al, 2019) and on the Deep Graph Library (Wang et al, 2020).

5.1 Baseline models

The baseline models are 1) constant-mean, 2) regression and 3) composed network models.

The **constant-mean model** is a simple baseline that returns the average of the observed efficiency for the training dataset, which is slightly different from that of the full dataset (78.9 instead of the 79.2 seen in Section 3). This approach is quite similar to the efficiency table technique presented in Section 1.

The **regression model** is a preliminary model that can be seen as a validation of the module architecture used for the regression step. It was the first baseline NN approach to predict efficiency using only high-level graph descriptors. This model uses only the regression module described in Section 4.3 and employs 87 graph features as inputs.

The **composed network** does not use the graph paradigm; *i.e.*, no edges, constraints, or message passing are used. This model uses the target architecture but without message passing. It aims to validate the mechanics related to the creation of image embeddings, readout, and regression. It builds a top-level representation using only node representations with no knowledge of the graph connectivity or edge information. The node embeddings are based on the output of the convolutional NN and are then directly passed through the readout function to build the nesting embedding.

5.2 Graph-based models

In the GNN group, the models include 1) relational GCN, which uses only the constraint types as edge information. Other models, 2) a continuous filtering GCN (CF-GCN), 3) a gated GCN and 4) a GTN, use constraint parameters as edge features in different ways, as seen in Section 2.

The **R-GCN** model integrates only the constraint types as edge information into message passing. It uses two layers of *relational graph convolutions*, as defined in (Schlichtkrull et al, 2018) by:

$$h_i^{l+1} = \sigma(W_0^l.h_i^l + \sum_{r \in R} \sum_{j \in N_i^r} \frac{1}{c_{ir}} W_r^l.h_j^l) \quad \forall i \in N, r \in R \tag{2}$$

where R is the set of relation types (15 total), c is a scale factor (set to 1), σ is the ReLU function, W_0 and W_r are learnable projections, and each W_r is specific to each type of edge, *i.e.*, 15 per layer. For each layer, the sizes of the matrices are (131×512) and (512×128). W_0 is used for the self-connections. In our case, the rotation constraints are represented as self-connections. Our implementation is based on the R-GCN layer from the Deep Graph Library.

The **CF-GCN** model integrates multidimensional edge features in the message passing phase and updates only the node embeddings. Message passing is performed with two layers of *continuous filter convolutions*, from SchNet (Schütt et al, 2017), which are described as:

$$h_i^{l+1} = O_o^l . \sum_{j \in N_i} (W^l . h_j^l \odot O_e^l . e_{ij}) \quad \forall i \in N, e \in E \tag{3}$$

where l is the layer index. $\odot$ is the Hadamard product operation (elementwise multiplication). W is a learnable projection with size of (131×256) for the first layer and (128×256) for the second layer. O_o and O_e are learnable nonlinear projections defined by

$$O_e = \sigma(W_2 . \sigma(W_1 . x)) \tag{4a}$$
$$O_o = \sigma(W_3 . x) \tag{4b}$$

where σ is the shifted softplus activation function. W_1, W_2 and W_3 are layer-specific learnable projections of size $e \times h$, $h \times h$ and $h \times o$, respectively, where e is the size of the edge features (45), h is the number of hidden features (256), and o is the number of output features (128). We use the implementation from the Deep Graph Library.

The **gated GCN** model integrates multidimensional edge features and updates node and edge embeddings. Message passing is achieved with two gated GCN layers from Bresson and Laurent (2018). For each layer l, the edge gate is denoted g_{ij}, and the node and edge feature update functions are

$$g_{ij}^l = \frac{\theta(e_{ij}^l)}{\sum_{j' \in N_i} \theta(e_{ij'}^l) + \epsilon} \tag{5a}$$
$$e_{ij}^{l+1} = e_{ij}^l + \sigma(\text{norm}(A^l . h_i^l + B^l . h_j^l + C^l . e_{ij}^l)) \tag{5b}$$
$$h_i^{l+1} = h_i^l + \sigma(\text{norm}(U^l . h_i^l + \sum_{j \in N_i} g_{ij}^l \odot V^l . h_j^l) \tag{5c}$$
$$\forall i \in N, e \in E$$

where l is the layer index and h and e are the node and edge vectors, respectively. N and E are the sets of graph vertices and edges. An edge from j to i is denoted as e_{ij}. The embeddings of the edge target and source nodes are denoted as h_i and h_j, respectively; θ and σ are the sigmoid and ReLU functions, respectively; norm represents 1-dimensional batch normalization; ϵ is a small constant used for stability. Finally, A, B, C, U and V are layer-specific learnable linear projections with compatible dimensions in this case (128×128). $\odot$ is the Hadamard product. Unlike the CF-GCN, this convolution function uses residual connections on each layer and propagates updated edge and node representations. The gate g_{ij} uses edge information as a soft attention mechanism in the node embedding update process. The implementation is based on Dwivedi et al (2020).

The **graph transformer network (GTN)** is described in Dwivedi and Bresson (2021). It builds node and edge embeddings using self-attention

between the neighbor nodes and edge projections, defined as

$$w'_{ij} = \frac{Q.h_i^l.K.h_j^l}{\sqrt{d}}.W.e_{ij}^l \tag{6a}$$

$$w_{ij} = \text{softmax}(w'_{ij}) \tag{6b}$$

$$h_i^{l+1} = O_h.\left(\oplus_c \sum_{j \in N_i} w_{ij}.V.h_j^l\right) \tag{6c}$$

$$e_{ij}^{l+1} = O_e.\left(\oplus_c w'_{ij}\right) \tag{6d}$$

$$\forall i \in N, e \in E, c \in H$$

where l is the layer index and h is a node vector. e_{ij} is the edge vector from j to i. N and E are the sets of graph nodes and edges, respectively. The embeddings of the edge target and source nodes are denoted by h_i and h_j, respectively. $\sqrt{d}$ is a normalization coefficient. For each layer, Q, K, V and W are learnable linear projections for the *query, key, value* and *edge* projections, respectively. The attention weight between nodes i and j is w_{ij}. The concatenation operation used in the multihead attention process is denoted by $\oplus$, and H is the number of heads. O_h and O_e are the nonlinear learnable projections of nodes and edges, *i.e.*, multilayer perceptron, defined as

$$x' = \text{norm}(x + W_1.x) \tag{7a}$$

$$y = \text{norm}(x' + W_3.\sigma(W_2.x')) \tag{7b}$$

where norm is a 1-dimensional batch normalization, σ is a ReLU function and W_1, W_2 and W_3 are learnable linear projections with sizes of $h \times h$, $h \times 2h$ and $2h \times h$, respectively. h is the number of features in the input vector x (here 128). y is the output vector with a size of 128.

In contrast with the gated GCN, this convolution model uses a transformer-like architecture. This model integrates edge information in the computation of the attention weights w_{ij}. As in the gated GCN, the node and edge embeddings are updated and then propagated to the next layer. The node and edge features are projected in compatible dimensional spaces (64 dimensions here, as needed by the scaled dot-product similarity) using a single linear layer. Then, these node and edge embeddings are passed through 2 graph transformer layers of size (128×16) with 8 heads to produce a final node embedding size of 128.

5.3 Experimental protocol and results

With this benchmark, our objective is to compare the baseline models (the constant-mean, simple regression and composed network models) with four GNN convolution models (the R-GCN, CF-GCN, gated GCN and GTN) to best fit our task data, given our architecture, by achieving the lowest average

error. The task dataset is split randomly into training, validation, and test subsets, with a $(70, 15, 15)$ percent split. The optimizer is adaptative moment estimation (Adam) with a batch size of 16. The optimization loss is

$$y = \alpha \mathrm{L1}(x) + \beta \mathrm{L2}(x) \tag{8}$$

where $L1$ computes the MAE and $L2$ computes the MSE. The $L2$ part of the loss can be seen as a manual regularization to penalize high errors. The α and β weights are fixed to 1 and 2, respectively. The learning rate is 0.001 with a weight decay (regularization) of 5.10^{-4}. We use an adaptative learning rate with a reduction factor of 0.5 and a patience of 5 for a total of 100 epochs. Our benchmark environment is based on Azure ML using 16 cores, 110 GB of memory and one Tesla T4 with 16 GB of memory.

On average, for each model (except the baseline models), a training step takes 35 minutes per epoch. The results are summarized in Table 1, where the presented values are averaged over 5 runs. The presented metrics are the *MAE*, *MSE*, *explained variance* and *error under* 3% (denoted as $< 3\%$). The MAE is the mean absolute error, which gives a model's average error. The explained variance gives a measure of the link between a model output and the variable to predict. For the efficiency estimation task, we want to avoid models that produce high errors. The MSE and $< 3\%$ are introduced to this end. The MSE is the mean squared error, where errors are squared and weighted: high error values are more penalized than lower values. The business-oriented metric, denoted as $< 3\%$, is defined by the percentage of the error distribution under the 3% threshold. This is a threshold beyond which the estimation is no longer meaningful. For all GNN models, we use two convolution layers with an internal representation possessing the same size and an output of 128 as the readout function input. This explains the variations in the numbers of parameters. The dispersion for each metric of the composed network, R-GCN, CF-GCN, gated GCN and GTN is shown in Figure A2.

Network	Edge feat.	Param.	MAE	MSE	Ex.Var	$< 3\%$
constant mean	–	–	7.36	119.9	0.0	26.22
simple regression	–	56.3 K	5.20	66.51	45.25	47.93
composed net.	–	193 K	1.80	12.29	89.76	85.06
R-GCN	+	2.3 M	**1.65**	**11.07**	**90.78**	**86.98**
CF-GCN	++	463 K	2.09	15.41	87.16	81.73
Gated GCN	++	696 K	2.06	14.61	87.86	81.61
GTN	++	661 K	2.05	14.75	87.76	82.04

Table 1 Comparative benchmark. +: edge type only, ++: edge features. Metrics are **Param.**: the model parameters, **MAE**: the mean absolute error, **MSE**: the mean squared error, **Ex.Var**: the explained variance, and **$< 3\%$**: the percentage of predictions under the 3% error threshold. These values are averaged over 5 runs for each model. See Figure A2 for a view of the distributions.

The *p-values* presented next are computed from test sample comparisons, *i.e.*, 75000 (5×15000) predictions per model, involving a Wilcoxon signed-rank test, which is nonparametric and specific to dependent and nonnormal distributions with a standard significance rate $\rho = 0.05$.

First, the constant-mean model with a baseline MAE of 7.36 shows that the use of complex input data, *i.e.*, parts and constraints, provides a benefit over other approaches and confirms that the efficiency is highly correlated with the nesting input data. Recall that this approach is quite similar to the use of efficiency tables.

Second, the simple regression model, which uses only the graph-level input data, achieves an MAE of 5.2, showing that the graph-level data carry meaningful information and validating the regression module of our general approach (all *p-values* $< 9.9e^{-4}$). The composed network is another validation model that uses graph-level and input shape data (images and image metadata) as inputs. Yielding an MAE of 1.8, the input shape information is critical for the efficiency estimation task (all *p-values* $< 1.56e^{-10}$).

Next, the R-GCN, which uses graph-level data, shape data, and partial edge information (only edge types) as inputs, achieves the best performance on our benchmark, with an MAE of 1.65 (all *p-values* $< 1.56e^{-10}$). It also achieves interesting performance in terms of other metrics, with over 90% of the variance explained and almost 87% of its estimations under the 3% error threshold.

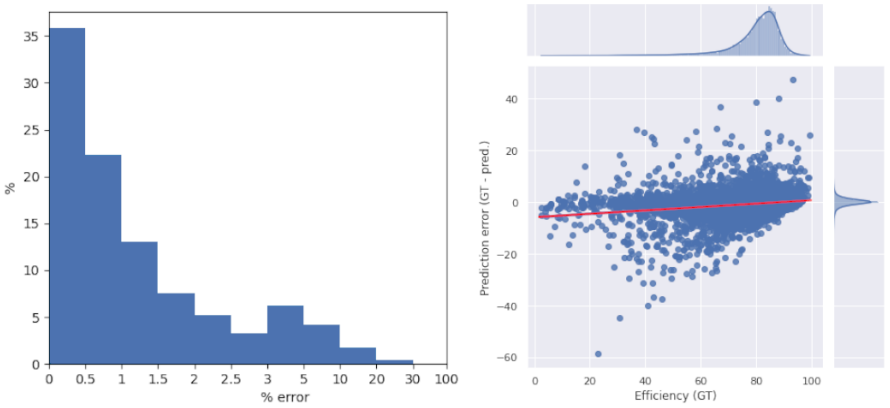

Fig. 8 R-GCN error distribution. Left: the nonuniform histogram of the error distribution for the best instance of the R-GCN. The focus is on the $< 3\%$ threshold on the left of the histogram. Here most of the error (around 50%) is under 1% threshold and approximately 87% of the error is under the 3% threshold. Right: Details of efficiency against prediction error for the same instance of the R-GCN. In red, a linear regression shows that the error is underestimated for high efficiency nestings and overestimated for low efficiency nestings, with few extremes values.

Finally, the comparison among the five main models (the composed network, R-GCN, CF-GCN, gated GCN and GTN) yields significant differences

between each pair (all *p-values* $< 1.56e^{-10}$). The approaches with complete edge information (the CF-GCN, gated GCN and GTN) perform surprisingly worse than the partial edge information approach (the R-GCN). This is particularly true for the GTN since transformer-based architectures are the state-of-the-art sequence processing approaches (related to graph processing) in deep learning. These points are discussed further in the next section with a specific focus on the GTN.

This work shows that our global approach works on the presented dataset (described in Section 3) with a validation of the part encoding module (by using an autoencoding task on the image encoding, as presented in Section 4.1), the regression module (Section 4.3) and finally the message passing module with the R-GCN. Combining these modules provides sufficiently accurate predictions (in terms of the MAE) to enable efficiency-dependent optimization use cases.

6 Discussion and limitations

Below, we discuss the important elements with respect to the encoding of the parts, the dataset, and the edge information significance within the complete edge information-based models.

Part encoding

To compute the piece shape descriptors, we choose to use embeddings extracted from a convolutional NN in an autoencoding task. Other solutions exist, such as boundary-based shape descriptors (Fourier or wavelet descriptors as in Laga et al (2006)), or region-based descriptors, representing shapes by local moment encoding (Xu and Li, 2008). Newer approaches are also available, such as GNNs (using shapes as graphs) or attentional mechanisms (using shapes as sequences of points). We could develop these approaches instead of the actual deep shape descriptors.

Dataset

First, the nesting algorithm used to generate our dataset depends on the time allocated to explore the solution space. For the collected dataset, this parameter is set to 5 minutes. This corresponds to the specific use of our customers, which can be considered a limitation of our dataset. Introducing some variability to this parameter could permit us to evaluate whether our approach generalizes to nesting operations with varying times. Additionally, we compute some descriptors on the graph dataset: the density (a measure of vertex interconnection), the clustering coefficient (the fraction of connected nodes triplets, *i.e.*, triangles (Saramaki et al, 2007)), the average global efficiency (*i.e.*, the inverse of the shortest path distances between all pairs of nodes), and the largest connected component for each graph. These metrics show that in our graphs, the densities are generally low, they contain no clustering-like structures (hardly any triangles), and they exhibit a low global efficiency. The

average global efficiency and connected component size distributions are shown in Figure 9. Most of the connected components have only very few nodes, thus confirming that our graphs are very sparse. We interpret this as the patterns learned by the models relying more on the local node and edge information than on the graph's global structural cells (Bodnar et al, 2021).

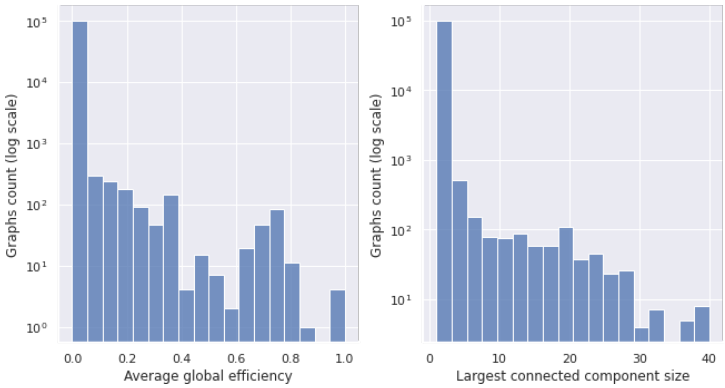

Fig. 9 Graph dataset metrics distribution. Left: average global efficiency, right: largest connected component size

Edges information uses

The performance of the edge-based GNNs (the CF-GCN, gated GCN and GTN) is quite mixed. Our interpretation is that 1) the input handcrafted edge features do not provide sufficient information for improving the model results and/or 2) we may not have found the optimal architecture/hyperparameter combination.

To test the importance of the edge information, we compare the three edge based models (the CF-GCN, gated GCN and GTN) with the protocol described in Section 5 by masking the edge information in the following ways: 1) with zero masking: all the edge information except for edge types is set to zero, 2) with random masking: all the edge information except for edge types is set to a random value following a normal distribution, and 3) only for the CF-GCN, the edge information is truncated only to edge types. The edge types are one-hot encoded features and are considered qualitative information. Other edge features are constraint parameters, as described in Section 4.1, that are encoded on numeric features and are considered quantitative information. The results are shown in Table 2.

The masking of edges slightly improves the performance of the gated GCN and GTN. For instance, with noise masking, the MAE of the gated GCN drops from 2.06 to 2.01 with a $p\text{-}value = 1.19e^{-44} < 0.05$. For the GTN, the MAE drops from 2.05 to 2.03 with a $p\text{-}value = 3.06e^{-46} < 0.05$. For the CF-GCN, edge masking has the opposite effect: we first attempt edge masking with zero noise, which shows a strong negative impact on performance (with an

Network	masking	MAE	MSE	Ex.Var	$< 3\%$
CF-GCN	zero	5.21	66.62	45.32	47.64
CF-GCN	noise	5.23	67.01	44.86	47.30
CF-GCN	truncated	5.21	66.46	45.36	47.49
gated GCN	zero	2.03	14.39	88.04	81.99
gated GCN	noise	2.01	14.23	88.16	82.33
GTN	zero	2.02	14.33	88.04	82.15
GTN	noise	2.03	14.54	87.88	81.96

Table 2 Edge masking. **zero**: edge information are set to zero; **noise**: edge information are set to a random value following a normal distribution, and **truncated**: only the edge types are kept. These values are averaged over 5 runs for each model.

MAE similar to that of simple regression at approximately 5.20). This seems to show that the CF-GCN is very sensitive to noise. To extend beyond this noise sensitivity, we then try to use truncated edge information (only edge type information is used). The performance are still low, with an MAE of 5.21. This result seems to show that the CF-GCN also must use quantitative edge information.

We conclude that

- the CF-GCN is sensitive to noise, probably because of the use of the input edge features e_{ij} at each layer (no edge embeddings are propagated from layer to layer) and the use of the Hadamard product in Equation 3 which contrasts with the gated GCN and GTN, which propagate edge embeddings and use sum or dot products in the edge update functions in Equations 5b and 6a;
- the CF-GCN needs quantitative edge information, and
- all of these models are not able to extract enough pertinent task information from the edge information to perform as well as the R-GCN which uses only edge type information. This can be an effect of our data modeling approach, where quantitative edge features seem to not be significant. As shown in Figure 9, the low number of edges for each graph also seems to have a negative impact on those models.

Regarding these points, we believe that the global performance of our approach can still be enhanced.

7 Conclusion

In this paper, we introduce a regression approach for estimating the efficiency of a nesting problem without solving it. This novel approach models a 2D nesting problem as a graph, where the nodes are images derived from parts and the edges hold the constraints. Convolutional NNs for image-based aspects and attentional GNNs for constraint aspects are combined. We propose and give to the community a large dataset with 100,000 labeled examples based

on real customer data. These dataset properties differ from those of graph-oriented datasets found in the literature. We evaluate an implementation of our approach on customer data and achieve sufficiently accurate results (MAE of 1.65) to enable several business use cases. In addition, we report on the performance and generalization capabilities of four GNN architectures on these nesting graphs. Such graph models can help the textile industry to be more sustainable by helping optimize material consumption in various phases:

- in the design phase with shape fitness representing material consumption;
- in the planning phase with automation and improved efficiency tables during the process of section planning;
- and eventually, in the postproduction phase by providing insights into a global nesting efficiency analysis.

For future work, we have three directions.

- We will use the edge embedding information in the readout function (the current version is a sum aggregation of node embeddings). By using both node and edge embeddings, we can achieve improved performance, especially for models using complete edge information (the CF-GCN, gated GCN and GTN);
- Specifically on the GTN, the performances are quite low. This seems to be a problem seen in the literature (Ying et al, 2021), where authors make the following observation: "it is still an open question whether Transformer architecture is suitable to model graphs and how to make it work in graph representation learning". The authors proposed several structural information encoding innovations to improve graph transformer models, such as *centrality encoding* as special node features that aim to capture the importance of nodes in a graph according to their degrees. *Spatial encoding* aims to add nodes' relative spatial relations to the attention similarity function, and finally, the authors provided a new specific edge encoding method where the edges on a path between nodes are considered in the correlation process of the attention method. We propose to use these innovations to see how they impact the performance of the GTN approach.
- As seen in Section 6, the graphs of our dataset are loosely connected. Another improvement could be with the use of a master node, as described in Gilmer et al (2017), which is connected to all other nodes in each graph. This could enable the generalization of the message passing phase to all nodes, even to orphan nodes and those with few connections.

Declarations

The authors declare:

- The manuscript is not submitted to more than one journal.
- The submitted work is original and has not been published.
- This work is not split in several parts to increase to quantity of submissions.

- Results are presented clearly, honestly, without fabrication, falsification or inappropriate data manipulation.
- No data, text, or theories by others are presented as if they were the author's own.

The authors did not receive support from any organization for the submitted work. However, Corentin Lallier is a PhD student (and is employed) at the Lectra company. Laurent Vézard is also employed by Lectra.

References

Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research (EJOR) 290(2):405–421. doi:10.1016/j.ejor.2020.07.063

Bennell JA, Oliveira JF (2009) A tutorial in irregular shape packing problems. Journal of the Operational Research Society 60(sup1):S93–S105. doi:10.1057/jors.2008.169

Bodnar C, Frasca F, Otter N, et al (2021) Weisfeiler and Lehman go cellular: CW networks. In: Ranzato M, Beygelzimer A, Dauphin Y, et al (eds) Advances in Neural Information Processing Systems, vol 34. Curran Associates, Inc., pp 2625–2640, URL https://proceedings.neurips.cc/paper/2021/file/157792e4abb490f99dbd738483e0d2d4-Paper.pdf

Bresson X, Laurent T (2018) Residual Gated Graph ConvNets. arXiv:171107553 [cs, stat] doi:10.48550/arXiv.1711.07553

Brockschmidt M (2020) GNN-FiLM: Graph neural networks with feature-wise linear modulation. In: Proc. of the 37th Int. Conf. on Machine Learning. JMLR.org, ICML'20, URL https://dl.acm.org/doi/10.5555/3524938.3525045

Chaudhari S, Mithal V, Polatkan G, et al (2021) An attentive survey of attention models. ACM Trans Intell Syst Technol 12(5). doi:10.1145/3465055

Dosovitskiy A, Beyer L, Kolesnikov A, et al (2021) An image is worth 16x16 words: Transformers for image recognition at scale. In: Int. Conf. on Learning Representations, URL https://openreview.net/forum?id=YicbFdNTTy

Dowsland KA, Vaid S, Dowsland WB (2002) An algorithm for polygon placement using a bottom-left strategy. European Journal of Operational Research (EJOR) 141(2):371–381. doi:10.1016/S0377-2217(02)00131-5

Dwivedi VP, Bresson X (2021) A Generalization of Transformer Networks to Graphs. Deep Learning on Graphs: Method and Applications (DLG-AAAI) doi:10.48550/arXiv.2012.09699

Dwivedi VP, Joshi CK, Laurent T, et al (2020) Benchmarking Graph Neural Networks. arXiv:200300982 [cs, stat] doi:10.48550/arXiv.2003.00982

ESICUP (2021) cutting and packing - datasets. URL https://www.euro-online.org/websites/esicup/data-sets/#1535972088237-bbcb74e3-b507

Gilmer J, Schoenholz SS, Riley PF, et al (2017) Neural Message Passing for Quantum Chemistry. Proc of Machine Learning Research (PMLR) URL https://dl.acm.org/doi/10.5555/3305381.3305512

Gomes AM, Oliveira JF (1999) Nesting irregular shapes with simulated annealing. Extended Abstracts of MIC1999—III Metaheuristics Int Conf pp 19–22

Gong L, Cheng Q (2019) Exploiting Edge Features in Graph Neural Networks. Computer Vision and Pattern Recognition Conf (CVPR) doi:10.1109/CVPR.2019.00943

Goodman E, Tetelbaum A, Kureichik V (1994) A Genetic Algorithm Approach to Compaction, Bin Packing, and Nesting Problems. Tech. rep., Michian State University, URL http://garage.cse.msu.edu/papers/GARAGe94-4.pdf

Henninger CE, Alevizou PJ, Oates CJ (2016) What is sustainable fashion? Journal of Fashion Marketing and Management 20(4):400–416. doi:10.1108/JFMM-07-2015-0052

Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: Int. Conf. on Learning Representations, URL https://openreview.net/forum?id=SJU4ayYgl

Laga H, Takahashi H, Nakajima M (2006) Spherical wavelet descriptors for content-based 3D model retrieval. Int Conf on Shape Modeling and Applications (SMI) doi:10.1109/SMI.2006.39

Lecun Y, Bottou L, Bengio Y, et al (1998) Gradient-based learning applied to document recognition. Proc of the IEEE 86(11):2278–2324. doi:10.1109/5.726791

Lee J, Lee I, Kang J (2019) Self-Attention Graph Pooling. In: Proc. of the 36th Int. Conf. on Machine Learning (ICML), PMLR, URL http://proceedings.mlr.press/v97/lee19c/lee19c.pdf

Li Y, Tarlow D, Brockschmidt M, et al (2017) Gated Graph Sequence Neural Networks. In: Int. Conf. on Learning Representations (ICLR), doi:10.48550/arXiv.1511.05493

Li Z, Milenkovic V (1995) Compaction and separation algorithms for non-convex polygons and their applications. European Journal of Operational Research (EJOR) 84(3):539–561. doi:10.1016/0377-2217(95)00021-H

Liu Hy, He Yj (2006) Algorithm for 2D irregular-shaped nesting problem based on the NFP algorithm and lowest-gravity-center principle. J Zhejiang Univ - Sci A 7(4):570–576. doi:10.1631/jzus.2006.A0570

Luong T, Pham H, Manning CD (2015) Effective approaches to attention-based neural machine translation. In: Proc. of Conf. on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Lisbon, Portugal, pp 1412–1421, doi:10.18653/v1/D15-1166

Mirhoseini A, Goldie A, Yazgan M, et al (2021) A graph placement methodology for fast chip design. Nature 594(7862):207–212. doi:10.1038/s41586-021-03544-w

Mo Y, Wu Q, Li X, et al (2021) Remaining useful life estimation via transformer encoder enhanced by a gated convolutional unit. Journal of Intelligent Manufacturing 32(7):1997–2006. doi:10.1007/s10845-021-01750-x

Oliveira JFC, Ferreira JAS (1993) Algorithms for Nesting Problems. In: Fandel G, Trockel W, Vidal RVV (eds) Applied Simulated Annealing, vol 396. Springer Berlin Heidelberg, Berlin, Heidelberg, p 255–273, doi:10.1007/978-3-642-46787-5_13

Oztemel E, Gursev S (2020) Literature review of industry 4.0 and related technologies. Journal of Intelligent Manufacturing 31(1):127–182. doi:10.1007/s10845-018-1433-8

Paszke A, Gross S, Massa F, et al (2019) Pytorch: An imperative style, high-performance deep learning library. In: Proc. of the 33^{rd} Int. Conf. on Neural Information Processing Systems. Curran Associates Inc., Red Hook, NY, USA, 721, URL https://dl.acm.org/doi/10.5555/3454287.3455008

Popescu A, Polat-Erdeniz S, Felfernig A, et al (2021) An overview of machine learning techniques in constraint solving. Journal of Intelligent Information Systems doi:10.1007/s10844-021-00666-5

Rissanen TI (2013) Zero-waste fashion design : a study at the intersection of cloth, fashion design and pattern cutting. Thesis, University of Technology Sydney, URL http://hdl.handle.net/10453/23384

Saramaki J, Kivela M, Onnela JP, et al (2007) Generalizations of the clustering coefficient to weighted complex networks. Physical Review E 75(2):027,105. doi:10.1103/PhysRevE.75.027105

Scarselli F, Gori M, Tsoi AC, et al (2009) The Graph Neural Network Model. IEEE transactions on neural networks 20(1):61–80. doi:10.1109/TNN.2008.2005605

Schlichtkrull M, Kipf TN, Bloem P, et al (2018) Modeling relational data with graph convolutional networks. In: European semantic web conf., Springer, pp 593–607, doi:10.1007/978-3-319-93417-4_38

Schütt KT, Kindermans PJ, Sauceda HE, et al (2017) SchNet: A continuous-filter convolutional neural network for modeling quantum interactions. In: Proc. of the 31st Int. Conf. on Neural Information Processing Systems. Curran Associates Inc., Red Hook, NY, USA, NIPS'17, p 992–1002, URL https://dl.acm.org/doi/abs/10.5555/3294771.3294866

Soong ZW (2015) Reinforcement learning for the 2D - packing problem. PhD thesis, The Hong Kong University of Science and Technology, doi:10.14711/thesis-b1514577

Vaswani A, Shazeer N, Parmar N, et al (2017) Attention is all you need. In: Guyon I, Luxburg UV, Bengio S, et al (eds) Advances in Neural Information Processing Systems, vol 30. Curran Associates, Inc., URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Veličković P, Cucurull G, Casanova A, et al (2018) Graph Attention Networks. Int Conf on Learning Representations (ICLR) p 12. doi:10.17863/CAM.48429

Wang B, Tao F, Fang X, et al (2021) Smart manufacturing and intelligent manufacturing: A comparative review. Engineering 7(6):738–757. doi:10.1016/j.eng.2020.07.017

Wang M, Zheng D, Ye Z, et al (2020) Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. Computing Research Repository (CoRR) URL https://www.dgl.ai/

Xie T, Grossman JC (2018) Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties. Physical Review Letters 120(14):145,301. doi:10.1103/PhysRevLett.120.145301

Xu D, Li H (2008) Geometric moment invariants. Pattern Recognition 41(1):240–249. doi:10.1016/j.patcog.2007.05.001

Xu K, Hu W, Leskovec J, et al (2019) How powerful are graph neural networks? In: Int. Conf. on Learning Representations, URL https://openreview.net/forum?id=ryGs6iA5Km

Xu Y, Thomassey S, Zeng X (2020) An application of machine learning to marker prediction in garment industry: Marker length estimation by neural network for the exponentially increasing magnitude of possible size combinations. In: Proc. of the 3^{rd} Int. Conf. on Applications of Intelligent Systems. ACM, pp 1–5, doi:10.1145/3378184.3378219

Ying C, Cai T, Luo S, et al (2021) Do transformers really perform badly for graph representation? In: Beygelzimer A, Dauphin Y, Liang P, et al (eds) Advances in Neural Information Processing Systems, URL https://openreview.net/forum?id=OeWooOxFwDa

Yun S, Jeong M, Kim R, et al (2019) Graph transformer networks. In: Wallach H, Larochelle H, Beygelzimer A, et al (eds) Advances in Neural Information Processing Systems, vol 32. Curran Associates, Inc., URL https://proceedings.neurips.cc/paper/2019/file/9d63484abb477c97640154d40595a3bb-Paper.pdf

Appendix A Supplementary Materials

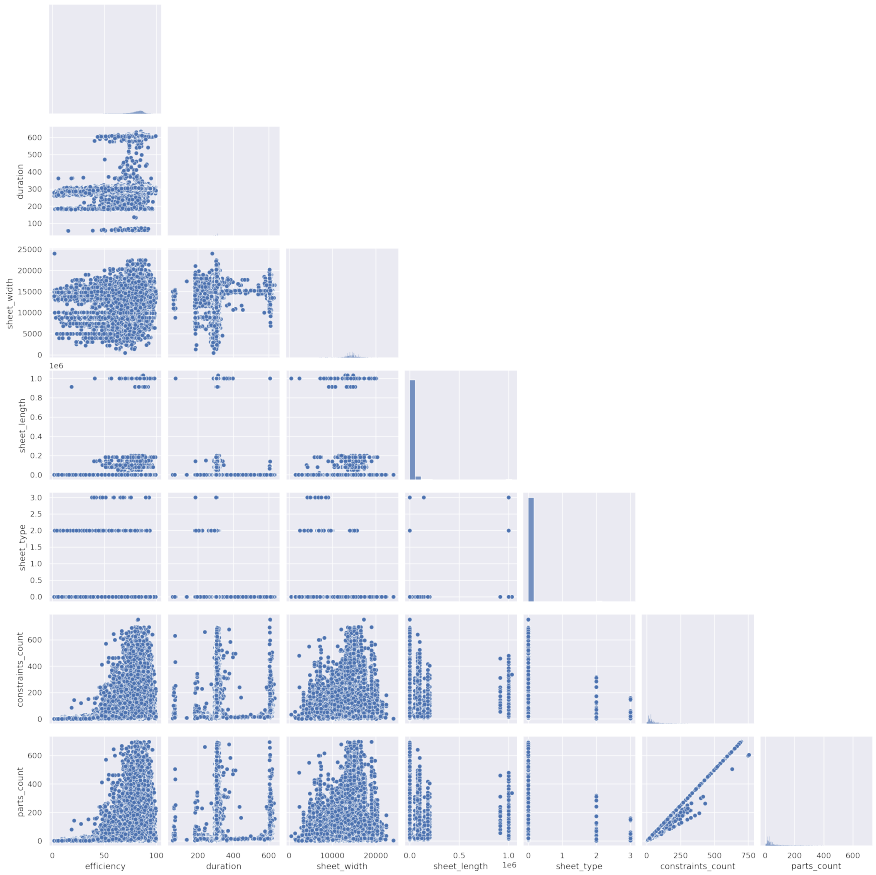

Fig. A1 Pair plot between the main variables

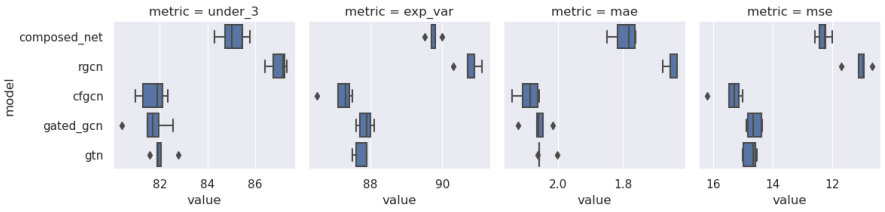

Fig. A2 Distribution of the benchmark results for the composed network, R-GCN, CF-GCN, gated GCN and GTN for the $< 3\%$ error threshold, the explained variance, the MAE and the MSE (the further to the left the better for all plots).