Journal of Intelligent Manufacturing (2024) 35:2811–2827
https://doi.org/10.1007/s10845-023-02179-0

Smart nesting: estimating geometrical compatibility in the nesting problem using graph neural networks

Kirolos Abdou[1,4] · Osama Mohammed[2] · George Eskandar[2] · Amgad Ibrahim[1] · Paul-Amaury Matt[3] · Marco F. Huber[3,4]

Received: 14 November 2022 / Accepted: 7 July 2023 / Published online: 24 July 2023

Abstract
Reducing material waste and computation time are primary objectives in cutting and packing problems (C&P). A solution to the C&P problem consists of many steps, including the grouping of items to be nested and the arrangement of the grouped items on a large object. Current algorithms use meta-heuristics to solve the arrangement problem directly without explicitly addressing the grouping problem. In this paper, we propose a new pipeline for the nesting problem that starts with grouping the items to be nested and then arranging them on large objects. To this end, we introduce and motivate a new concept, namely the Geometrical Compatibility Index (GCI). Items with higher GCI should be clustered together. Since no labels exist for GCIs, we propose to model GCIs as bidirectional weighted edges of a graph that we call geometrical relationship graph (GRG). We propose a novel reinforcement-learning-based framework, which consists of two graph neural networks trained in an actor-critic-like fashion to learn GCIs. Then, to group the items into clusters, we model the GRG as a capacitated vehicle routing problem graph and solve it using meta-heuristics. Experiments conducted on a private dataset with regularly and irregularly shaped items show that the proposed algorithm can achieve a significant reduction in computation time (30% to 48%) compared to an open-source nesting software while attaining similar trim loss on regular items and a threefold improvement in trim loss on irregular items.

Keywords Reinforcement learning · Graph neural networks · Cutting and packing · Nesting · Geometrical compatibility · Geometrical relationship graph · Grouping for nesting

✉ Kirolos Abdou
kirolos.abdou@trumpf.com

Osama Mohammed
st170249@stud.uni-stuttgart.de

George Eskandar
george.eskandar@iss.uni-stuttgart.de

Amgad Ibrahim
amgad.ibrahim@trumpf.com

Paul-Amaury Matt
paul-amaury.matt@ipa.fraunhofer.de

Marco F. Huber
marco.huber@ieee.org

1 Research and Development, TRUMPF Werkzeugmaschinen SE + Co. KG, Ditzingen, Germany

2 Institute of Signal Processing and System Theory ISS, University of Stuttgart, Stuttgart, Germany

3 Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Stuttgart, Germany

4 Institute of Industrial Manufacturing and Management IFF, University of Stuttgart, Stuttgart, Germany

Introduction

The cutting and packing problems (in short C&P problems) are related to many areas of the operations research and they have been extensively studied for decades. The C&P problems have been referred to, in literature, with different terminologies over the past decades (Arenales et al., 1999; Dyckhoff, 1990; Wäscher et al., 2007). For example, a C&P problem was denoted by the layout design problem in Cagan et al. (2002), while it was called the spatial configuration problem in Michalek et al. (2002). Further examples of terminologies and variants of the C&P problems are cutting stock or trim loss problem, bin or strip packing problem, vehicle, pallet or container loading problem, nesting problem, knapsack problem, etc. C&P problems arise in many industries, including the wood industry, glass industry, paper industry, sheet metal cutting, ship building, garment manufacturing, shoe manufacturing, and furniture making.

The C&P problems are NP-hard problems with an identical common logical structure which can be described as follows. The inputs are two groups of geometrical data ele-

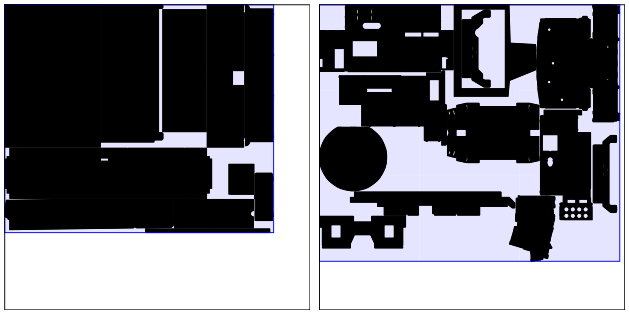
ments, namely a set of large objects, and a set of small items. The aim is then to select a set of small items, group them into one or more subsets, and finally pack each of the resulting subsets on a corresponding large object, in a pattern (a.k.a. layout), while satisfying some geometric conditions (Faina, 2020). The first geometric condition to be satisfied is that the small items of each subset must be laid entirely within the large object. The second geometric condition is that all small items lying on the same large object must not overlap. The residual unused areas on the layout, remaining uncovered by a small item, are usually treated as wasted material which is interchangeable with the term "trim loss". The objectives of the C&P problem are to reduce this waste to maximize material utilization and reduce computation time. Reducing the trim loss can be very beneficial from the economic point of view in very large-scale productions such as sheet metal, wood, and textile production as it can result in savings of material and consequently in reduction of production costs.

In general, a solution to a C&P problem may consist of using a set of some or all large objects, and a set of some or all small items depending on the problem's objective and constraints. Considering the broad constraints, a formal formulation of five sub-problems can be derived (Wäscher et al., 2007):

1. Large object selection problem: selecting the large objects on which the small items should be packed or cut.
2. Small items selection problem: selecting the small items to be packed or cut.
3. Grouping problem: grouping the selected small items into subsets.
4. Allocation problem: assigning the subsets of the small items to the large objects.
5. Layout problem: arranging the small items on each of the selected large objects, by first finding the order of the small items and then by placing them on the large object with respect to the geometric conditions.

In almost all of the C&P industries, the small items to be packed are provided by customers and hence the second sub-problem is disregarded. If similar or identical large objects are to be used, the first and fourth sub-problems will be dropped. Therefore, the C&P problem is practically reduced to only the third and fifth sub-problems, i.e., grouping the parts to be nested into clusters and finding a layout for each cluster by arranging its small items on a large object.

In most of the C&P industries, the C&P process is referred to as the *nesting* process. The small items are also usually called parts and the large objects are called sheets. The latter terms, i.e., nesting, parts, and sheets, will be henceforth used in this paper.



(a) Eleven Geometrically Compatible Parts (b) Twelve Geometrically Incompatible Parts

Fig. 1 We estimate geometrical compatibility indices (GCI) with our model and choose two sets of parts with the same total area: *left*, eleven parts with high compatibility indices and *right*, twelve parts with low indices. The highly compatible parts result in reduced trim loss

Over the past decades, the fifth sub-problem, the "layout problem", has been mostly addressed (Furini & Malaguti, 2013; Kundu et al., 2019; Labib & Assadi, 2007; Rakotonirainy & van Vuuren, 2020; Zhang et al., 2016), where it is assumed that only one sheet is used for all of the parts, and hence no clustering of parts is needed. In contrast, the third sub-problem, the "grouping problem", has often been ignored. The current existing nesting softwares known to us solve the grouping problem implicitly as a part of the layout problem.

In this paper, we hypothesize that a new nesting pipeline, where the grouping problem is directly solved before the layout problem, will positively impact both material utilization and computation time. The positive impact of this new nesting pipeline on material utilization is intuitively alleged (Fig. 1), as the material waste will be low when the clustered parts already fit together geometrically (i.e., the grouping problem) before placing them on the sheet (i.e., the layout problem). Moreover, by solving the grouping problem before the layout problem, the new nesting pipeline shortens the layout problem's computation time because the meta-heuristics of the layout problem will handle a smaller number of parts per cluster.

Therefore, in this work, we suggest solving the grouping problem explicitly with a learning-based approach. To the best of our knowledge, this learning-based approach is the first approach to solve the grouping problem explicitly in the scope of nesting. The layout problem will be addressed in a follow-up work and is out of the scope of this paper.

Despite its material-saving and time-saving potential, the realization of the new nesting pipeline is complex. The challenges are the absence of a definite mathematical formulation of the grouping problem, the entanglement of the grouping and layout problems, and the difficulty of instinctively understanding the geometrical compatibility between the parts.

Accordingly, we start by differentiating between the Geometrical Compatibility Index (GCI) concept and the contour similarity concept, a closely related concept in the field of C&P, in section "Related works". We also give a background about the algorithms used by our approach in the same section. Then, in section "Problem formulation", we draft a mathematical formulation of the grouping problem. In light of this formulation, we exhibit the entanglement intricacy between the grouping and layout problems in section "Motivation to the proposed approach", motivate the proposed new nesting pipeline, introduce the two new concepts [i.e., GCI and Geometrical Relationship Graph (GRG)], and deduce the learning paradigm. Next, we formally define the GRG and GCI concepts in section "Definitions: GRG and GCI". In section "Methodology", we concretely explain our framework. After that, we conduct multiple experiments to showcase the effectiveness of our approach on a large real-world private dataset in section "Experiments and results". Finally, we conclude our work in section "Conclusion" and suggest further future research directions.

To summarize, our main contributions in this paper are as follows:

1. We propose a new pipeline for nesting, which begins by solving the grouping problem before the layout problem.
2. We define two new concepts, namely the *Geometrical Relationship Graph* (GRG) and *Geometrical Compatibility Index* (GCI), which will allow solving the grouping problem.
3. To estimate the GCI, we propose a novel reinforcement learning-based framework that deploys two Graph Neural Networks (GNNs) in an actor-critic-like fashion.
4. We suggest to group the parts based on the GCI by solving the GRG as a Capacitated Vehicle Routing Problem (CVRP) using meta-heuristics.
5. We demonstrate that our model can train on more than 2, 000 parts with irregular shapes. We report results on two test splits from a private dataset from real-world sheet metal production. Results show that our best model can achieve a considerable improvement (30% to 48%) in computation time, a 70% reduction in material waste on one test split, and a comparable trim loss on the other test split.

Related works

As the grouping problem has not been explicitly addressed before, there is no prior work relating to it. This section, therefore, distinguishes the GCI from the contour similarity concept used in the field of C&P. We also briefly describe relevant works in other fields that are used in our methodology, like graph neural networks, CVRP, and actor-critic reinforcement learning.

Contour similarity

To increase the compactness and reduce the scrap while solving the layout problem, the *contour similarity* concept was introduced in the literature on the 2D-bin packing problem (Guo et al., 2020; Yang et al., 2022). The contour similarity aims to match the new parts with the sheet's unoccupied closed polygons having a similar contour.

The concept of geometrical compatibility differs from the contour similarity concept in the following: (1) Contour similarity is a specific case from the geometrical compatibility; if two parts are geometrically compatible, they will be nested together on the same sheet, but they will not necessarily be adjacent on the sheet, as would be the case for contour similarity. (2) Contour similarity is computed pair-wise and would be time exhausting if computed for all possible pairs inside a group of parts. On the other hand, GCI is computed graph-wise and considers relationships between all parts at once in the GRG. For instance, in Yang et al. (2022), contour similarity was calculated using an iteration-based algorithm, the longest similarity subsequence, which at its best has a time complexity of $O(j \cdot l)$, where j and l are the lengths of the two input sequences. GCI is estimated using a learning-based algorithm with a $O(1)$ time complexity at inference time. (3) Lastly, contour similarity targets the layout problem, unlike GCI, which focuses on the grouping problem.

Graph neural networks (GNNs)

There exist three typical classification tasks in a graph: node classification, edge classification, as well as joint node and edge classification. The classification task is either based on node features only [e.g., GraphSAGE (Liu et al., 2022) and DeepWalk (Perozzi et al., 2014)] or on both node and edge features [e.g., EGNN (Kim et al., 2019)]. Graph Convolutional Neural Network (GCNConv) is a convolutional operator first introduced in Kipf and Welling (2017). The majority of GNN models share a largely universal architecture that is based on using a message passing function to aggregate information from neighbor nodes into the current node. The main message passing function used in the forward path of the GCNConv is given by

$$\mathrm{X}' = \hat{\mathrm{D}}^{-1/2}\hat{\mathrm{A}}\hat{\mathrm{D}}^{-1/2}\mathrm{X}\Theta, \tag{1}$$

where

$$\hat{\mathrm{A}} = \mathrm{A} + \mathrm{I}. \tag{2}$$

In Eq. (1), A is the adjacency matrix of the input graph representing which nodes are connected together. The adjacency matrix would be a matrix of ones if the input graph is a fully connected graph, which applies to the GRG. $\hat{A}$ represents the adjacency matrix with self-loops inserted, i.e., each node is connected to itself which is depicted by the identity matrix I. The adjacency matrix is also known as the *edge weights matrix* when it contains values that indicate how closely connected the nodes are to one another rather than only values that indicate the nodes' connectivity (zeros and ones). $\hat{D}$ is the diagonal node degree matrix, where the diagonal element corresponds to the number of edges connected to the node, namely, $\hat{D}_{ii} = \sum_{j=1}^{N} \hat{A}_{ij}$. Furthermore, X denotes a matrix of node feature vectors, Θ is the trainable weight matrix for the GCNConv layer, and X' is the weighted aggregation of the features of the neighbor nodes.

The architecture of our framework's learning agent deploys GNNs, specifically the GCNConv, to estimate the GCIs.

CVRP

The capacitated vehicle routing problem (CVRP) is an NP-hard optimization problem used in supply chain management to design an optimal route for a fleet of vehicles with uniform capacities serving customers with known demands for a single commodity under three constraints. The three constraints are minimizing the global distance between customers, minimizing the number of vehicles needed to serve them all, and ensuring that each vehicle does not exceed its capacity. The CVRP has been extensively studied in literature through many heuristics and meta-heuristic algorithms surveyed in Kumar and Panneerselvam (2012).

In our suggested framework, after estimating the GCIs, the GRG is considered as CVRP and is solved using a meta-heuristics-based solver.

Actor-critic reinforcement learning

Reinforcement learning (RL) is a decision-making framework where an agent represents the solver of the problem, and an environment represents the problem to be solved. At each state of solving the problem, the agent interacts with the environment by selecting an action to be applied. By applying the action, the environment updates its state and returns the new state and a scalar reward evaluating the quality of the taken action. In any state, the agent selects the action by following a policy. The agent's goal is to optimize its policy by maximizing the total accumulated reward, also called the return. To decide on actions to be taken in future states, the agent indirectly uses past experiences by building an estimation function of the return called the value function. One widely used agent's paradigm is the *actor-critic paradigm* (Grondman et al., 2012), where the actor tries to optimize the policy, and the critic tries to optimize the value function. The central concept behind all the actor-critic algorithms is that the value function estimated and optimized by the critic is used to guide the actor's policy to the improvement direction of performance.

The learning agent's paradigm in our framework is conceptually similar to the actor-critic idea.

Problem formulation

In this section, we formally define the grouping problem. We assume there are N parts, $\{\text{part}_1, \ldots, \text{part}_N\}$, to be nested on K sheets. For part_n, where $n \in \{1, \ldots, N\}$, we denote its area by a_n and its geometry information by g_n. The geometry information encodes the inner and outer contours of each part. In this work, we consider that g_n is a vector in latent space, $\mathbb{R}^d$, where d is the dimension of the latent space. For a sheet $k \in \{1, \ldots, K\}$, the sheet's area is denoted by s_k. We seek to find $x_n^k \in \{0, 1\}$, which denotes whether part_n will be assigned to sheet k. Parts that are assigned to the same sheet are grouped together. The optimization problem can be formulated as follows:

$$\arg\min_{x_n^k} \sum_{k=1}^{K} \left[Area(BB_k) - \sum_{n=1}^{N} x_n^k a_n \right] \tag{3a}$$

$$\text{s.t.} \sum_{k=1}^{K} x_n^k = 1, \forall n \in \{1, \ldots, N\} \tag{3b}$$

$$\sum_{n=1}^{N} x_n^k a_n \leq s_k, \forall k \in \{1, \ldots, K\} \tag{3c}$$

$$\sum_{n=1}^{N} x_n^k a_n + \left(1 - x_m^k\right) a_m > s_k,$$

$$\forall \left\{x_m^k, a_m\right\} \in \left\{ \left\{x_i^k, a_i\right\} \mid i \in \{1, \ldots, N\}, x_i^k = 0 \right\}$$

$$\forall k \in \{1, \ldots, K\} \tag{3d}$$

BB_k is the rectangular bounding box enclosing all the parts nested on sheet k. BB_k is first calculated by solving the layout problem for sheet k, i.e., after nesting all parts that have $x_n^k = 1$ on sheet k. Equation (3a) denotes finding x_n^k that minimizes the material waste, $Area(BB_k) - \sum_{n=1}^{N} x_n^k a_n$, on each sheet. The constraint (3b) states that each part must be assigned to one and only one sheet. The second constraint (3c) states that the total area of all nested parts on sheet k must not exceed the sheet's area, s_k. The final constraint (3d) imposes that each sheet k must be used to its maximum capacity such that it is not possible to add any more part, part_m with $x_m^k = 0$, to the sheet without exceeding its capacity.

This optimization problem is ill-defined because $Area(BB_k)$ is an unknown non-linear function, $\mathcal{F}$, of the nested parts' geometry and areas. It can be expressed as:

$$Area(BB_k) = \mathcal{F}\Big(g_1 x_1^k, g_2 x_2^k, \ldots, g_N x_N^k, a_1 x_1^k, a_2 x_2^k, \ldots, a_N x_N^k\Big) \tag{4}$$

A solution to the grouping problem consists of finding a mapping function π, which maps the parts' geometrical information g_n, parts' areas a_n, and the areas of the sheets s_k to the coefficients x_n^k, which determine the groups.

Motivation to the proposed approach

To find a solution to the grouping problem, we seek to approximate the mapping

$$\pi : \{a_n, g_n\}_{n=1}^{N} \bigcup \{s_k\}_{k=1}^{K} \mapsto \left\{x_n^k\right\}_{k,n=1}^{K,N} . \tag{5}$$

This mapping is hard to estimate as it implies optimizing the objective function (3a) under the constraints (3b)–(3d) without any insight into the non-linear function $Area(BB_k)$ or its relationship to the nested parts' geometry (Eq. (4)). The BB_k term in Eq. (3a) unveils the entanglement of the layout problem in the grouping problem; only after solving the layout problem for sheet k, BB_k and $Area(BB_k)$ can be calculated.

To disentangle the grouping problem from the layout problem, we propose to break this mapping down into two mappings by introducing an intermediary value that can be learned from the data and can then act as a criterion for optimizing the grouping. The proposed breakdown and intermediary value pave the way for a new formulation of the objective function that does not include $Area(BB_k)$ as will be shown in section "Mapping from GCIs to groups". For the intermediary value, we define a new concept, the *Geometrical Compatibility Index* (GCI). The mapping π can be expressed as $\pi = \pi_2 \circ \pi_1$. The first function in the proposed solution, π_1, tries to estimate the GCIs between all the parts given their geometrical information, $\{g_n\}_{n=1}^{N}$. The set of all estimated GCIs is expressed as $\{GCI_{n,m}\}$, where n and m are two different parts. The second function, π_2, estimates the coefficients x_n^k given the GCIs. For instance, parts with high GCIs should be grouped together, while parts that have low GCIs should belong to different groups. In short, the mapping functions can be expressed as:

$$\pi_2 \circ \pi_1 : \{a_n, g_n\}_{n=1}^{N} \bigcup \{s_k\}_{k=1}^{K} \mapsto \{x_n^k\}_{k,n=1}^{K,N} \tag{6a}$$

$$\pi_1 : \{g_n\}_{n=1}^{N} \mapsto \{GCI_{n,m}\}_{n,m=1}^{N,N} \tag{6b}$$

$$\pi_2 : \{GCI_{n,m}\}_{n,m=1}^{N,N} \bigcup \{a_n\}_{n=1}^{N} \bigcup \{s_k\}_{k=1}^{K} \mapsto \{x_n^k\}_{k,n=1}^{K,N} \tag{6c}$$

In the following two subsections, we motivate the proposed solution to estimate the mapping functions, π_1 and π_2. Then, we motivate the proposed learning approach in section "Learning paradigm".

Mapping from parts to GCIs

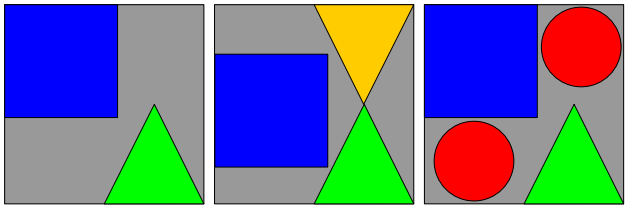
The goal of the mapping function π_1 is to estimate a quantitative criterion to cluster the parts. We hypothesize that geometrical compatibility between parts is a reasonable choice for the grouping criteria.

Since GCI is an abstract concept, we motivate it by the following simple example before providing its formal definition in section "Definitions: GRG and GCI".

To show how the selection of geometrically compatible parts can lead to more material savings, we show three different simple parts (square, triangle, and circle) in Fig. 2. In the example, we used the grey canvas to represent a square sheet of area 22,500 mm^2, on which we want to nest the parts. We also assume two previously nested parts (Fig. 2a) shown in green and blue with 2812.5 mm^2 and 7225 mm^2 areas, respectively. The resulting free area of the sheet is then 12,462.5 mm^2, which theoretically is more than sufficient to nest another two orange triangular parts with the same dimensions as the green one. By trying to nest two new orange triangles, it becomes obvious that a single one would not fit on the sheet with this layout. The only option that would allow placing a new orange triangle is to change the layout by moving the blue square a bit down (Fig. 2b). This, however, also prevents the placement of a second orange triangle, albeit the fact that theoretically the available area on the grey sheet is still enough to accommodate it. On the other hand, a red circle with a radius of 30 mm and an area of 2827.43 mm^2, i.e., almost the same area as the green and orange triangles, would fit perfectly on the sheet without even changing the layout. Furthermore, it leaves room for a second red circle to be nested (Fig. 2c). Consequently, we would say that the orange triangle is geometrically less compatible with the blue square and the green triangle than the red circle.

The red circle instead has higher geometric compatibility with the nested parts and therefore left more space for another red circle. This subsequently has increased area utilization and reduced material waste. Quantitatively, the wasted area for nesting only a single orange triangle is 9650 mm^2, while the wasted area for nesting two red circles is 6807.6 mm^2. Hence, higher geometrical compatibility saves more area for new parts to be nested and reduces trim loss.

As shown in the example above, the GCI estimation problem cannot be just estimated in a pairwise manner, without



(a) Sheet before nesting (b) Nesting one additional triangular part (c) Nesting two circular parts

Fig. 2 Demonstration of the geometrical compatibility between different parts

observing other parts in a group. To estimate GCIs between all parts, we choose to model the parts as nodes in a fully connected bidirectional weighted graph and GCIs as weighted edges between the nodes. We refer to this graph as the *Geometrical Relationship Graph* (GRG), and we use a GNN to estimate the weights of its edges.

Mapping from GCIs to groups

After estimating the GCIs, we propose to use them as a criterion for the grouping. The mapping π_2 can be seen as a solution to another optimization problem, which seeks to find out what are the optimal sets of parts that should be chosen together to achieve the maximum GCI sum for different sheets. The objective function of this optimization problem can be expressed as

$$\arg \min_{GCI} \sum_{k=1}^{K} \sum_{m=1}^{N} \sum_{n=1}^{N} (1 - GCI_{n,m}) x_n^k x_m^k. \tag{7}$$

This new formulation, while satisfying the constraints in (3b)–(3d), is analogous to a typical Capacitated Vehicle Routing Problem (CVRP) formulation. This new objective function does not depend on the unknown function $Area(BB_k)$, unlike the initial one of Eq. (3a), but rather on the GCIs estimated by π_1. To solve for π_2, we use metaheuristics instead of exact methods to find a good solution in an acceptable time.

Learning paradigm

From section "Mapping from parts to GCIs", it is clear that the GCI cannot be concretely estimated in a defined way before solving the layout problem. Moreover, it is not possible to get labels for the GCIs, because the GCIs between parts change whenever the grouping strategy changes. It is also very computationally expensive to use a brute-force approach and try out all grouping permutations to generate GCI labels for supervised learning. For this, we propose to learn the GCIs by maximizing a reward function from a nesting environment using RL. The reward function $\mathcal{R}$ is chosen to be the ratio between the area of the parts of a group and the area of the bounding box enclosing them on the sheet k, and it is computed for each group of parts using

$$\mathcal{R}_k = \frac{\sum_{n=1}^{N} x_n^k a_n}{Area(BB_k)}. \tag{8}$$

Definitions: GRG and GCI

In this section, we give formal definitions for GCI and GRG.

Definition 1 (*GRG*) The GRG is a fully connected, bidirectional weighted graph representing the parts to be nested along with their geometrical relationships.

- Graph: GRG depicts the parts to be nested as nodes in the space and their geometrical relationships as the edges connecting them. GRG has nodes $v_n \in V = \{v_1, \ldots, v_N\}$ where each node is represented by a geometrical information vector $g_n \in \mathbb{R}^d$ (details can be found in section "Input encoding").
- Fully connected weighted graph: To outline all of the geometrical relationships between the parts, all the nodes of the GRG are connected to each other and to themselves by an edge $e_{nm} \in E \,\forall\, v_n, v_m \in V$. GRG defines the affinity of the geometrical relationship between two parts v_n and v_m by a weight $w_{nm} \in [0, 1]$ assigned to their connecting edge e_{nm}.
- Bidirectional graph: In the example of section "Mapping from parts to GCIs", nesting the triangular part (green) before the square part had a high geometrical compatibility. However, nesting the triangular part (orange) after the square part had a low geometrical compatibility. This means that the order of nesting the parts affects their geometrical suitability, which is represented by directed edges between nodes in the GRG, i.e., $w_{nm} \neq w_{mn}$.

Definition 2 (*GCI*) The GCI is a scalar variable with fixed bounds that is computed graph-wise rather than pair-wise. Its value is the weight w_{nm} of the edge e_{nm} in the GRG, i.e., $w_{nm} = GCI_{n,m} \in [0, 1]$. It denotes the geometrical relationship between two parts in the sense of their geometrical compatibility, not their area compatibility, to be nested together on the same sheet.

- Scalar variable with fixed bounds: To have a concrete view of the substantial effect of the GCI on nesting, the GCI is designed to be a scalar value ranging from zero to one. A GCI of zero means no geometrical compatibility whilst a GCI of one is the highest compatibility index.

- Computed graph-wise rather than pair-wise: The GCI cannot be determined by geometrically evaluating the fitness of each pair of parts together (the green triangle looks compatible with the blue square in Fig. 2a but the orange triangle is barely compatible with the same rectangle in Fig. 2b), but rather by nesting all the parts of the GRG graph/sub-graph on a sheet and evaluating the resulting layout. Since there is no explicit calculation method for the GCI, it is instead inferred via machine learning.
- Denotes geometrical compatibility not area compatibility: Two different nodes that are area-wise compatible with a third node, i.e., their sum of areas would fit on a sheet, not necessarily need to have the same GCI with this node in the GRG. This is the case of the orange triangle and the red circle and their compatibility with the blue square in Fig. 2.

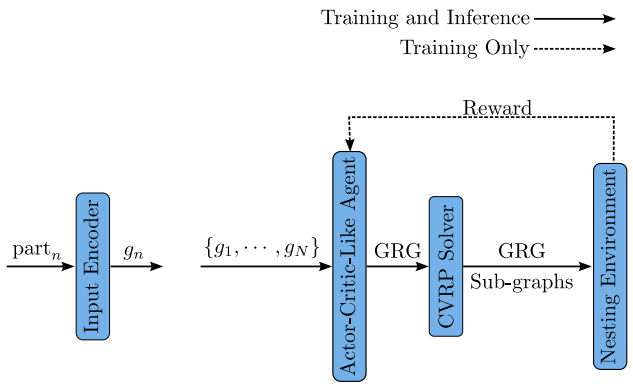


Fig. 3 A flowchart summarizing our proposed methodology: the autoencoder is pre-trained separately and generates the parts encoding g_n. Then, an actor-critic-like agent learns to generate GCIs, which are passed to a CVRP solver to group the parts. A nesting environment evaluates how well the groups will be nested and returns a reward to the agent. During inference, the critic-like module in the agent is omitted and only the actor-like module is used

Methodology

In this section, we present the proposed approach for solving the grouping problem. First, we present an overview of the different components of our approach, then we present the parts encoding, the architectures of the actor-like and critic-like modules, and the learning routine.

Overview of the proposed approach

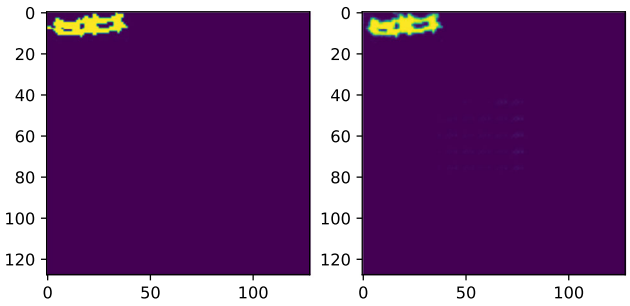
Our approach consists of finding the mappings π_1 and π_2. The proposed framework consists of four components:

1. Input encoder: It takes as input a part, part$_n$, and generates as output the part's geometrical information vector g_n.
2. Actor-critic-like agent: It represents the first policy π_1 mapping from parts to GCIs. It consists of two modules, namely, the actor-like module and the critic-like module. The input of this component is the set $\{g_1, \ldots, g_N\}$ and its output is the GRG, where the parts are the nodes and the weights of the edges are the GCIs. During the training phase, it learns the values of the GCIs, which improve the quality of the nesting, with the help of the reward signal provided by the nesting environment in an RL framework. During the inference phase, i.e., the actual nesting, only the actor-like module is used to predict the values of the GCIs to build the GRG.
3. CVRP solver: It represents the second policy π_2 mapping from GCIs to groups. This component considers the GRG, built by the previous component, as a CVRP problem and solves it using meta-heuristics. Its input is the GRG and it outputs sub-graphs of the GRG as routes. Each resulting GRG sub-graph represents a group of parts to be nested together on the same sheet.
4. Nesting environment: It solves the layout problem for the parts of each GRG sub-graph, and returns the reward of Equation (8) to the actor-critic-like agent during the training phase. During inference, it only solves the layout problem for each group of parts without returning any reward.

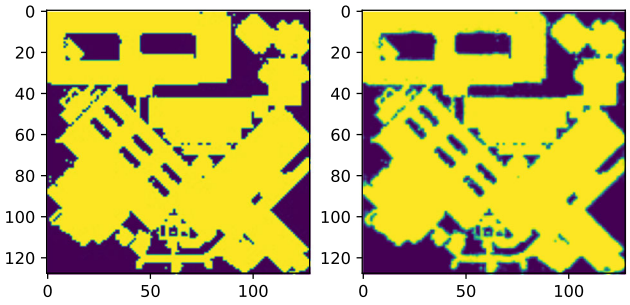
Figure 3 exhibits a block diagram of the four components constituting our methodology. In sections "Input encoding" and "Actor-critic-like architecture", the first and second components will be explained in detail, respectively. The third and fourth components will be explained in the learning routine of the last subsection as they cannot be explained alone.

Input encoding

The input to our algorithm is the set of parts to be nested $\{\text{part}_1, \ldots, \text{part}_N\}$, where each part is described by its area and its geometrical shape. To alleviate the task of the first policy, π_1, we encode the geometrical shape of each part, described as a rasterized image, into a geometrical information vector. Inspired by representation learning, the encoder of an autoencoder is employed in a pre-processing step to encode the rasterized image of each part into a vector $g_n \in \mathbb{R}^d$, where we use $d = 256$ throughout this paper. An encoding of the parts facilitates the GCI learning task and reduces the required computation resources thanks to the reduced dimensionality. The encoded geometrical information vector g_n of part$_n$ will be further used as the GRG node's features in section "Actor-critic-like architecture". A customized autoencoder based on the inception network (Szegedy et al., 2015) was trained on a different dataset of more than 10, 000 rasterized parts including some



(a) Single part reconstruction



(b) Layout reconstruction

Fig. 4 Autoencoder reconstruction results on unseen new images: *left*, the rasterized images of a single part (**a**) and a layout (**b**) and *right*, the reconstructed images after encoding

layout images. To evaluate the encoding generalization on new images, the autoencoder has been tested on unseen new images of single parts (Fig. 4a) and layouts (Fig. 4b).

Actor-critic-like architecture

Motivated by the "learning with a critic" concept (Widrow et al., 1973), which incited Sutton, Barto, and Anderson (Barto et al., 1983) to outline the fundamental concepts of all the modern actor-critic algorithms in the RL field, our model consists of two modules: actor-like and critic-like modules (Fig. 5).

Critic-like module

In the critic-like module, the nodes of a sub-graph of the GRG are passed through two consecutive GCNConv layers to perform message passing between the nodes. The rationale for using a sub-graph rather than the entire GRG will become clear in the "Data Collection" step of Sect. 6.4. Both GCNConv layers take initially a random edge weights matrix representing the GCIs. The output of the first GCNConv layer is then provided as input to the second GCNConv layer. Theoretically speaking, in Eq. (2), the matrix of node geometrical information vectors $[g_1, \ldots, g_N]$ of the input graph represents the X, while the edge weights represent the $\widehat{A}$ matrix and are learnable parameters.

To guide the improvement of the actor-like module, the final GCNConv layer's outputs are averaged to a single value estimating the environment's reward. This value is compared to the ground-truth reward signal coming from the environment using a L_1 loss function. The L_1 loss is further used in backpropagation with Stochastic Gradient Descent (SGD) to update the parameters of the two GCNConv layers and the edge weights matrix. Each parameter of this edge weights matrix stands for a GCI between its connecting nodes (i.e., parts).

Actor-like module

The actor-like module consists of one GCNConv layer, which takes the nodes of the GRG as input and an adjacency matrix of ones (as it is assumed that all nodes are connected). The goal of the actor-like module is to learn a forward mapping between the input graph nodes and the edge weights, $\widehat{A}_{estimate}$, which is a matrix of dimensions $N \times N$, where N is the number of the nodes in the graph. After updating the edge weights of the critic-like module, the L_2 loss function is computed between the output of the actor-like module $\widehat{A}_{estimate}$ and the edge weights $\widehat{A}$ learned by the critic-like module. The parameters of the actor-like module's GCNConv layer are then updated via stochastic gradient descent. In summary, the actor-like module learns an edge-labeling task.

It might be mistakenly thought that the critic-like module has achieved the whole task by learning the edge weights, and the actor-like module is redundant. Practically speaking, this is partially true. The critic-like module learns the GCIs by backpropagating on the learnable parameters of the edge weights matrix. However, the critic-like module maps the input nodes and the GCIs to a reward function and only learns the GCIs by backpropagation, which is impossible at test time.

In simple words, we can say that the critic-like module has no memory for learned knowledge. The role of the actor-like module is to generalize the knowledge learned by the critic-like module.

Actor-critic-like vs. actor-critic

The actor-critic and actor-critic-like paradigms agree on the concept but differ in its implementation. The critic in the actor-critic paradigm implicitly improves the actor, by insinuating the direction of actions update from an evaluation signal (e.g., $v - value$, $q - value$, $a - value$ or return). On the other hand, the critic-like in the actor-critic-like paradigm explicitly learns the good actions by directly estimating the reward signal and teaching the actor those actions. The actor and critic have "act then get criticized" dynamics: the actor takes action first and then the critic evaluates its perfor-

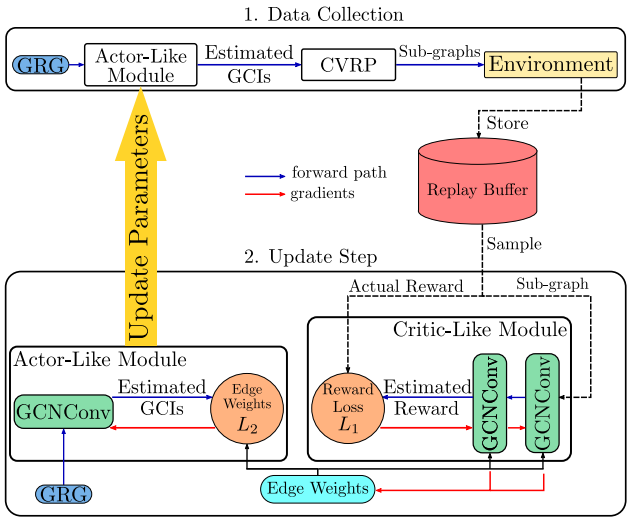


Fig. 5 Block diagram of actor-critic-like modules, the data collection and the update steps in the learning routine

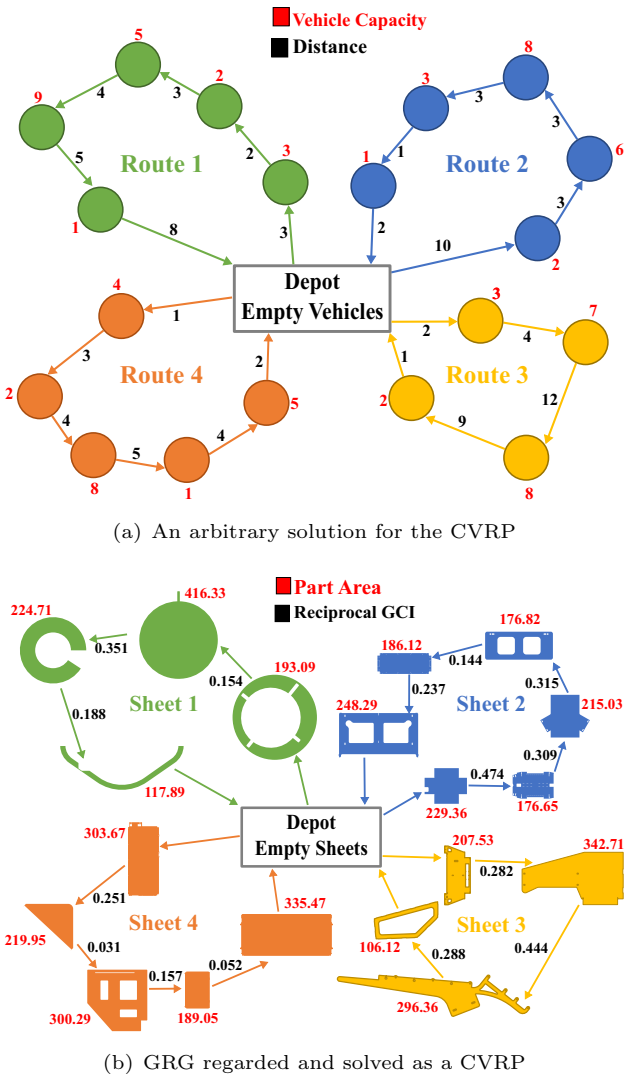


Fig. 6 Analogy between GRG and CVRP

mance. The dynamics between the actor-like and critic-like are rather "get criticized and then learn": the critic-like learns an explicit action by evaluating it, and then the actor-like learns this action.

Learning routine

The GCIs of the GRG could only be learned by nesting the parts of the GRG on sheets and evaluating the quality of the layouts in an RL framework. In this section, the GCI's, and accordingly, the GRG's learning procedure (Alg. 1) is explained in detail.

Initialization of the environment

The first step is to initialize a nesting environment. In the RL context, the action space consists of all the sets of parts that could be nested together on the same sheet, i.e., each action is a sub-graph of the GRG. The state space includes only the initial state, which is the set of all parts in the GRG. In other words, there exists only one state, which is the whole GRG graph, and each action is a sub-graph of the GRG that the agent believes that its parts will geometrically fit together on the same sheet. The environment takes a sub-graph, nests its parts on the same sheet, and returns the quality of the resulting layout (Eq. (8)) as a "reward" to the agent.

Initialization of a replay buffer

To store the experiences collected from the agent and environment interactions, a replay buffer is initialized for the GRG. Each entry of the replay buffer is a pair consisting of a binary mask and a reward. The mask filters out all the GRG's parts that are not included in the sub-graph. The replay buffer has a limited capacity and once it is full the newest experience replaces the oldest one. This allows an off-policy training since experiences collected from old policies are used to learn a new one.

Data collection

In the data collection step (Fig. 5), the GCIs, between all parts of the GRG, are inferred from the actor-like module. The GRG is then regarded and solved as a CVRP. We draw an analogy between CVRP and GRG: the reciprocal of the GCI values, i.e., $1 - GCI$, in GRG are analogous to the distances between the customers in CVRP. Both need to be minimized. On the other hand, the areas of the parts in GRG, are similar to the capacities of the customers in CVRP. Both need to meet capacity constraints. Finally, the available empty sheets to

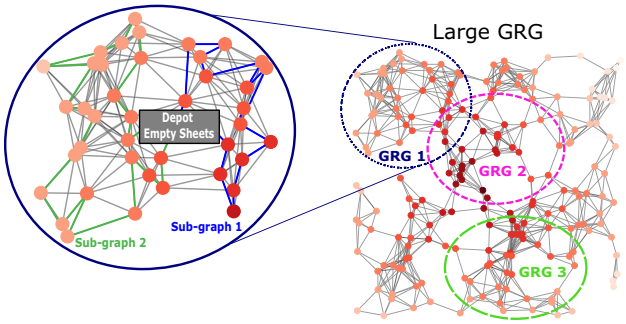


Fig. 7 Demonstration of a large GRG, GRG, and sub-graphs of a single GRG: each large GRG is divided into η GRGs. Each GRG is further divided into sub-graphs, using the CVRP solver, where each sub-graph represents a group of parts that will be nested together on the same sheet

nest the sub-graphs of the GRG correspond to the available vehicles in its CVRP counterpart. A visual demonstration of this analogy is provided in Fig. 6. Out of the several meta-heuristic algorithms that handle the CVRP, we used Google OR-Tools (Perron and Furnon) to solve the GRG into sub-graphs. Each sub-graph is then passed to the environment to be nested on the same sheet and a reward is returned. The mask of the sub-graph and the reward are then stored in the replay buffer.

Update step

To update the actor-like and critic-like modules, we sample a random batch of experiences from the replay buffer. The mask of each element in this batch is used to recover the sub-graph and the reward is used to compute the L_1 loss and update the critic-like module. After the update of the critic-like module, the whole GRG is used by the actor-like module to estimate the GCIs. Those GCIs are again compared to the edge weights matrix learned by the critic-like module, in an L_2 loss function, to update the actor-like module (Fig. 5). Note that the critic-like module has learned the GCIs of the whole GRG by only using sub-graphs from the GRG. This design choice is motivated by the fact that the geometrical compatibility should only be evaluated for clusters of parts to be nested through the reward signal. Hence, by evaluating the quality of many sub-graphs, the critic-like can infer the geometrical compatibility between all the parts of the GRG. Nevertheless, the critic-like module is a transductive model that cannot generalize its learned knowledge of edge weights to unseen GRGs. On the other hand, the actor-like module is an inductive model that can easily predict the GCIs of newly encountered GRGs.

Extension to large GRGs

To limit the computation resources, the GRG is restricted to comprise at most N parts. To extend the model on larger datasets that contain M parts with $M > N$, we divide the *large GRG* into $\eta = \lceil \frac{M}{N} \rceil$ GRGs of N parts each (Fig. 7). In the training phase, the above-described learning process (Alg. 1) will then be applied to each of the η GRGs. At the start of training for each GRG, a new replay buffer and an edge weights matrix for the critic-like module are initialized. The edge weights matrix of the critic-like module is randomly initialized only for the first GRG. However, for all the following ones, it is initialized with the previously learned edge weights matrix. This could be thought of as a simple form of transfer learning. In the inference phase, only the actor-like model is used to estimate the GCIs of the GRGs.

Algorithm 1 GCI Learning Routine

```
Initialize a nesting environment
Divide the large GRG into η GRGs of N parts each
for n ← 1 to η do
    Initialize a replay buffer B_n and Â_n
    for i ← 1 to I training steps do
        collect data from GRG_n
        store the data in B_n
        sample a random batch from B_n
        do an update step
    end for
end for
```

Experiments and results

We perform multiple experiments to showcase the effectiveness of our model. In section "Effect of different reward functions on the model generalization", we study the effect of different choices for the reward functions for a single GRG with $N = 500$. Next, in section "Comparison of GRG with open-source Nesting software", we compare the performance of the proposed model to an open-source nesting software.

Nesting environment

For our experiments, we developed a dummy environment that does not solve the layout problem but rather considers only the parts' areas to calculate the reward of the current nested parts. There are several reasons behind the choice of a dummy environment. First, real nesting environments are very slow for solving the layout problem compared to the GNNs and will increase their training time drastically by increasing the data collection time. Second, a dummy environment can be considered as a more general and abstract case of a real nesting environment as the proposed algorithm is not limited to a specific layout-solver algorithm. This design choice constitutes a proof of concept of the GCI and GRG ideas.

(a) Before preprocessing (b) After preprocessing

Fig. 8 An example part from our dataset

Realizing the reward function (Eq. (8)) in the dummy environment is impossible since this reward is a function of $Area(BB_K)$, which is a non-linear function of the geometries and the areas of the nested parts (Equation (4)) that can only be calculated after solving the layout problem. However, in the dummy environment, the layout problem is not solved and the bounding box area cannot be computed. To approximately simulate the non-linearity of the reward in (8), we designed the reward function of the dummy environment to be a non-linear function $\mathcal{F}$ of the utilization ratio U. For each subgraph (group of parts) formed by the CVRP to be nested on sheet k with area s_k, the utilization ratio U_k and the reward $\mathcal{R}_k$ are calculated according to

$$U_k = \frac{\sum_{n=1}^{N} x_n^k a_n}{s_k} ,$$
$$\mathcal{R}_k = \mathcal{F}(U_k) .$$

For the selection of $\mathcal{F}$, the effects of three different non-linear functions have been examined in section "Effect of different reward functions on the model generalization".

Dataset

Typical nesting datasets are not suitable for the grouping problem because they were originally designed to solve the layout problem. They feature a small number of different parts (typically around 20) and a single sheet. For this reason, we evaluate our model using a private dataset with a large number of parts that cannot fit on a single sheet. Our dataset consists of 2500 CAD files representing different irregular complex geometrical parts from sheet metal production (Fig. 8a). The parts used are to be nested on sheets of size 3000 mm × 3000 mm. The CAD files of the parts are translated into Scalable Vector Graphics (SVG) format. To capture the difference in dimensions between parts, the canvas is set as a constant frame of reference. Each part is placed at the top left corner of the canvas and is rasterized into a 128 × 128 pixels binary image as shown in Fig. 8b. We divide our dataset into a training set (2000 parts) and a test set of 500 parts. Two test splits are further selected from the test set and used to evaluate the models. Test Split 1 contains 113 parts with regular shapes, while Test Split 2 contains 124 parts with irregular shapes. Regular shapes contain mostly straight lines and smooth curves. On the other hand, irregular shapes have more complex concave and convex curves. The test splits were selected based on manual inspection. All the parts in the test splits are different from each other and from the training split.

CVRP solver

For optimizing Eq. (7) and solving the GRG into sub-graphs using CVRP, we use the specialized library from Google OR-Tools (Perron and Furnon) for vehicle routing. This tool uses local search and meta-heuristics on top of a constraint programming solver to solve different variants of the vehicle routing problem including CVRP. The Google OR-Tools' vehicle routing library implements a two-step solution, where the first solution is an initial solution based on heuristics and is further improved by local search meta-heuristics in the second step. We use the automatic settings to select the best first solution strategy and to let the solver select the best local search meta-heuristics to guide the search. The meta-heuristics among which the solver is selecting include greedy descent, guided local search, simulated annealing, tabu search, and generic tabu search.

Effect of different reward functions on the model generalization

In this set of experiments, we select only 500 parts from the training set to train the models, which can be grouped into one GRG. We experiment with three different non-linear functions $\mathcal{F}$ for the reward calculation to test the generalization ability of the proposed model. We train the first model using a sinusoidal function of the utilization ratio as follows: $\mathcal{R}_k = \sin((U_k - 0.5) \cdot \pi)$. The range of the utilization is set to $[-0.5, 0.5]$ to make the reward range between $[-1, 1]$. The second model uses a sigmoid function for the reward. This reward is also non-linear and represents a different challenge to the agent. The reward function is expressed as follows: $\mathcal{R}_k = \frac{1}{1+e^{-U_k}}$. In this case, the range of the reward is $[0, 1]$. In the third model, we use the hyperbolic tangent function of the utilization ratio to calculate the reward. The reward function in this case is expressed as follows: $\mathcal{R}_k = \tanh(U_k) = \frac{e^{U_k} - e^{-U_k}}{e^{U_k} + e^{-U_k}}$. The non-linear tanh function limits the range of the reward in this case between $[-1, 1]$.

Model a In this model, the reward function is a non-linear sinusoidal function. Figure 9b shows the edge weights error L_2 during the 500 training steps under the Sinusoidal reward

function. It can be noted that L_2 loss converges quickly to a relatively small value. Figure 9a shows the reward loss L_1 which exhibits the same convergence behavior.

Model b In this model, we use a sigmoid reward function. Figure 9d shows the edge weights error L_2 during the 500 training steps. Similar to model a, the L_2 loss and L_1 loss converge quickly (see Fig. 9c).

Model c In this model, we use a tanh reward function. Figure 9e and f show the quick convergence of the reward loss L_1 and the edge weights loss L_2 during the 500 training steps, respectively.

We show the estimated GCI for Models a, b, and c on test split 1 in Fig. 10a, c, and e, and on test split 2 in Fig. 10b, d, and f, respectively. Qualitatively, we can see that, for the same test split, the graphs are similar with few exceptions. The slight differences between the colors of the edges, i.e., the values of the GCIs, can be attributed to the different non-linearities used in the reward function. The values of the GCIs cannot be intuitively interpreted since they are computed in a graph-wise, not a pair-wise manner. The qualitative similarities between the graphs make it difficult to decide which model is better. For this reason, we compare the performance of the three models quantitatively in the following subsection.

Comparison of GRG with open-source nesting software

To the best of our knowledge, the nesting sub-problem of grouping the parts to be nested into clusters before doing the nesting itself has never been handled before by any nesting software. Almost all of the current nesting softwares are trying to solve two other nesting sub-problems, namely finding the best order of the parts to nest, and selecting the angle and xy-position of each part on the sheet, i.e., the nesting/packing itself. After selecting the nesting order of the parts, usually through meta-heuristics, and during packing the parts on the sheet, the parts that do not fit on the current sheet are placed on a new one. That's how the clustering sub-problem is solved in the nesting software without taking into consideration the suitability of the parts to be nested together.

To get an impression of how our approach compares to other existing frameworks, we conducted an experiment to compare the performance of our framework to the performance of an open-source nesting software called DeepNestPort. DeepNestPort is substantially a port of a browser-based vector nesting tool called SVGNest to handle different input/output image formats. We choose DeepNestPort as a baseline to compare with for three reasons: (1) it's an open-source software, which makes it easily accessible for research purposes, (2) SVGNest/DeepNestPort claims to have a similar performance to commercial softwares, and (3) other learning frameworks that solve the layout problem,

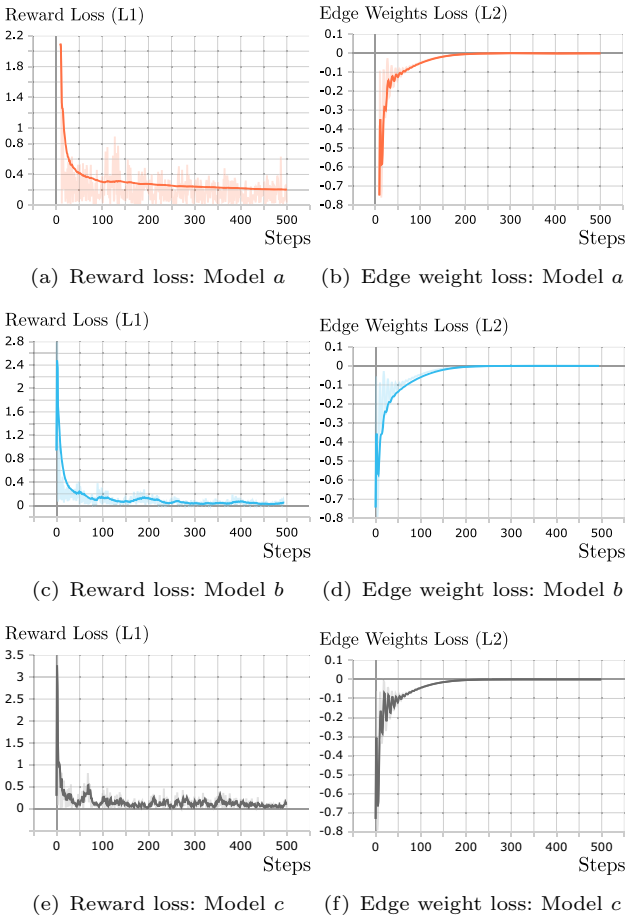


Fig. 9 Results of updating the actor-like and critic-like modules for Models a, b, and c: *left,* the L_1 loss function between the reward estimated by the critic-like module and the ground-truth reward signal coming from the environment, and *right,* the L_2 loss function between the output of the actor-like module $\widehat{A}_{estimate}$ and the edge weights $\widehat{A}$ learned by the critic-like module

operate only on a smaller number of regular parts and are not suitable to operate on a large number of regular and irregular parts. It should be noted that we used only a simple dummy environment throughout the training of our model, which we expect to affect the results negatively. Training with one of the existing nesting softwares as an environment would be impossible, as they are very slow and therefore not suitable for RL training, which needs huge amounts of samples to learn from.

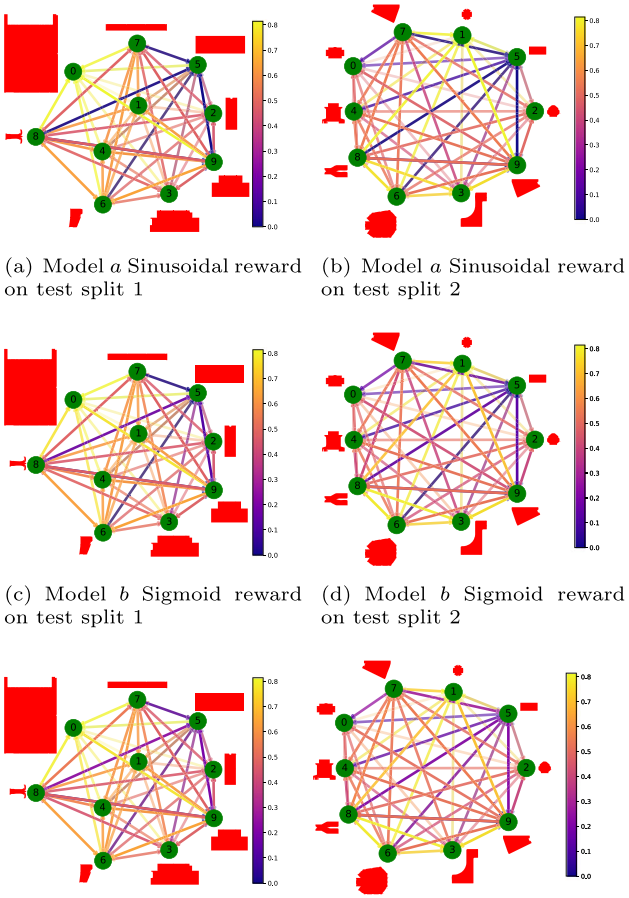
Baseline and models

In this experiment, we first give all parts in each test split to the DeepNestPort framework to nest. The performance of this experiment is considered the baseline. Then, we report the performance of six models: Models a, b, and c trained twice, on 500 parts and 2000 parts. We trained Models a, b, and c using the same reward functions of section "Effect

Table 1 Comparison between GRG and DeepNestPort

Experiments	Training	Test Split 1		Test Split 2	
		T (s)	WM (%)	T (s)	WM (%)
Deep Nest Port	NA	600	**13**	761	20
Model a	500 parts	300	15	610	12
Model b		**285**	18	521	19
Model c		290	19	490	17
Model a	2000 parts	317	16	**498**	**7**
Model b		353	18	498	20
Model c		340	20	498	16
Average		314.2	17.7	519.2	15.2

Test Split 1 contains regular parts, while Test Split 2 contains irregular parts. Best numbers are marked in bold. Models a, b, and c have the same architecture but were trained with three different non-linear reward functions: sinusoidal, sigmoid, and tanh, respectively



(a) Model a Sinusoidal reward on test split 1

(b) Model a Sinusoidal reward on test split 2

(c) Model b Sigmoid reward on test split 1

(d) Model b Sigmoid reward on test split 2

(e) Model c tanh reward on test split 1

(f) Model c tanh reward on test split 2

Fig. 10 Visualization of the estimated GCIs for three different non-linear functions $\mathcal{F}$ and on two different test splits. The nodes of each graph represent some of the parts clustered together in each test split. Beside each node stands the geometrical shape of the corresponding part visualized in red. The colored edges between the nodes show the parts' geometrical relationships, where the colors describe the values of the GCIs according to the color bars (Color figure online)

of different reward functions on the model generalization", namely, sinusoidal, sigmoid, and tanh, respectively. For each model, we estimate the GCI values of the parts and split them with CVRP. Then, we nest each group of parts, corresponding to a specific route from the CVRP problem on a sheet using DeepNestPort. The main difference is that DeepNestPort assigns the parts to each sheet on its own, while in our case, the parts are assigned according to their GCIs.

Metrics

We evaluate the performance of our model using two metrics: time (T) and trim loss expressed as the percentage of wasted material (WM). Time (in seconds) is calculated by measuring the computation time of all used sheets after the proposed grouping operation. Wasted material (expressed as a percentage) is the ratio of the sum of wasted areas in all used sheets over the total area of the sheets. The wasted area of a sheet is considered as only the unused area inside the rectangular bounding box enclosing all the parts nested on a sheet. The unenclosed area could be reused to nest new parts. Therefore it is not considered scrap.

Results and discussion

The results are reported in Table 1. In the case of using DeepNestPort, a total number of four and two sheets each of the dimension 3000 mm × 3000 mm were needed to nest the required 113 and 124 parts of test split 1 and 2, respectively. Test splits 1 and 2 were nested in an average time of 600 s and 761 s and the material wasted percentage, in this case, was 13% and 20% of the total area of the used sheets, respectively. In the case of using GCIs, a total number of four sheets was suggested for test split 1 and two sheets for test split 2. The layouts visualization is shown in Appendix: Qualitative results, For each experiment, each layout shows a group of parts that have been selected to be nested together. For exper-

iments that are using the GCIs to group the parts, the set of all layouts represents the full solution of the CVRP solver for its respective GRG.

From the table, we first notice that Model *a* generalizes better than Models *b* and *c* in all combinations. This shows empirically that the sinusoidal reward is better suited than the sigmoid and the tanh rewards. Second, all the models achieve better than the baseline on test split 2, with Model *a* resulting in only 7% material waste, almost 3 times less than DeepNestPort, when trained on 2000 parts. Models *b* and *c* still achieve a considerable reduction when trained only on 500 parts. The difference in the two results can be explained away by the number of parts during training. The three models however experience a slight degradation in performance on test split 1. We hypothesize that this effect could be mitigated by designing a more realistic nesting environment or tuning the choice of the reward function.

Overall, the framework consistently achieves a lower computation time than the classical nesting approach in all combinations. The computation time is reduced on average by 48% on test split 1 and by 30% on test split 2. The longer time in test split 2 is attributed to the irregular shapes of the parts. This stems from the fact that solving the grouping problem implicitly during the layout problem results in a large number of combinations that the nesting environment should consider. In contrast, the proposed pipeline divides the parts to be nested into small groups with more compatible parts, which in turn can be nested directly on the sheets in fewer combinations.

Limitations

One limitation of this work is that no nesting environment was used during the training. However, this is a promising result, because the model can achieve a considerable reduction in time and waste material on irregular parts. We believe that the use of a realistic nesting environment during the training will further reduce material waste. However, the design of such an environment is out of the scope of this paper and is considered in future works.

Another limitation of the proposed framework is that the use of meta-heuristics to solve the CVRP problem might lead to sub-optimal results or even no results at all in an acceptable time frame. We plan to address this issue in future works by replacing meta-heuristics with a plug-and-play learning-based approach, e.g. Nazari et al. (2018). Overall, these results validate our hypothesis that a different pipeline design (where the grouping problem is solved before the layout problem) can significantly speed up the nesting process and reduce the scrap produced.

Conclusion

In this work, we revisit the conventional nesting pipeline by directly solving the grouping problem of parts before the arrangement problem. To achieve this, we introduced and formally defined two new concepts, namely GRG and GCI. We proposed an RL-based framework to estimate the GCIs of parts in an unsupervised way. The framework consists of two GNNs trained in an actor-critic-like fashion, where the actor estimates the GCIs and the critic judges their quality by fitting the reward function of a nesting environment. The proposed framework was tested on three dummy environments with different non-linear reward functions. A comparison with an open-source nesting software was conducted to showcase the performance of the proposed grouping and nesting pipeline. We demonstrate the effectiveness of the framework by achieving a three-fold improvement in the waste material on a test split with irregular shapes and a 30% improvement in computation time. Similarly, the model achieves a 48% reduction in time with a minimal sacrifice in waste material on another test split with regular parts. In future works, we plan on completing the pipeline by developing a smart nesting algorithm for the layout problem.

Funding Open Access funding enabled and organized by Projekt DEAL.

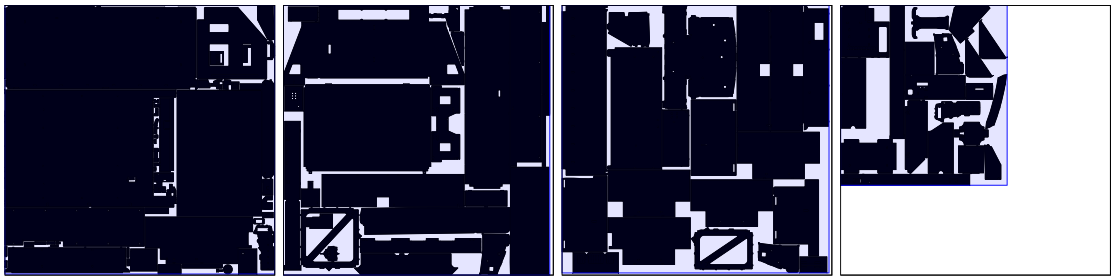
Data availability The dataset used in this study is privately owned by TRUMPF Werkzeugmaschinen SE + Co. KG and contains proprietary and commercially sensitive information. Due to contractual agreements and data protection regulations, the dataset cannot be made publicly available or shared.

Declarations

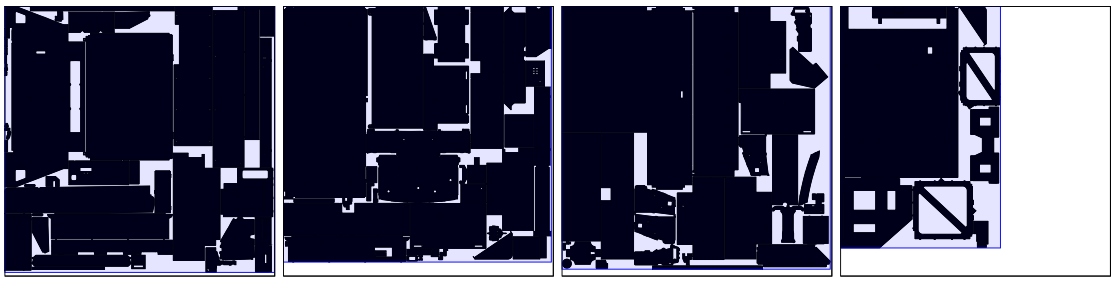
Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix: Qualitative results

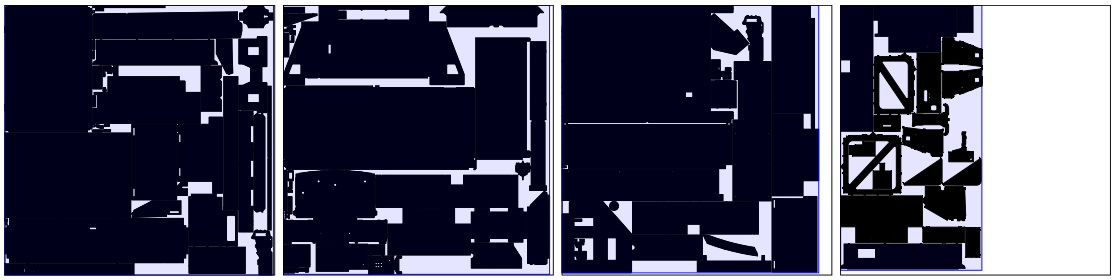
See Figs. 11, 12, and 13.



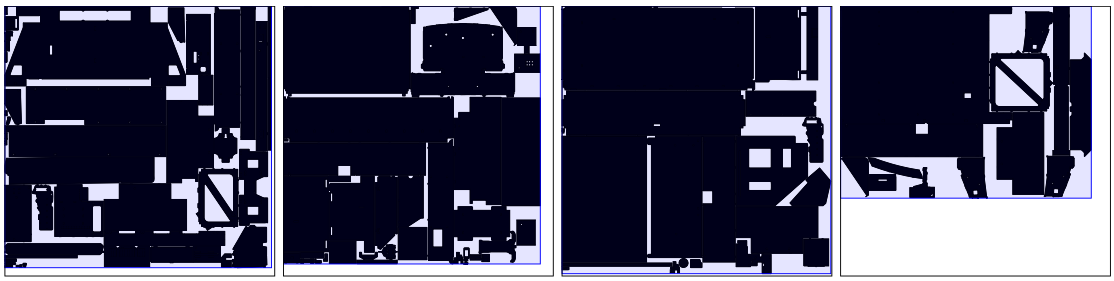
(a) DeepNestPort



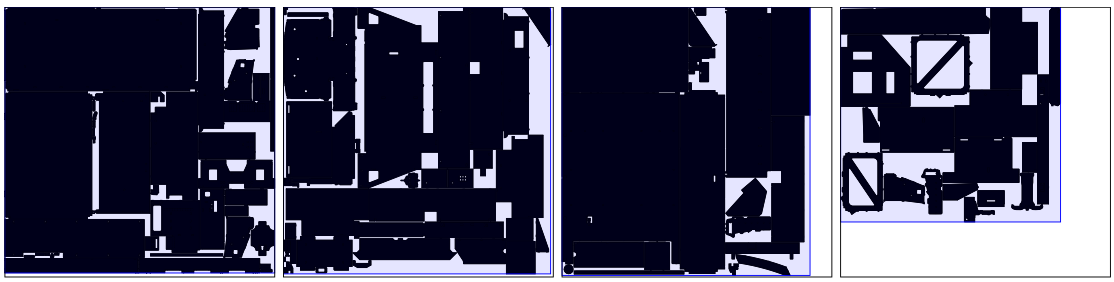
(b) Model *a* trained with 500 parts



(c) Model *a* trained with 2000 parts

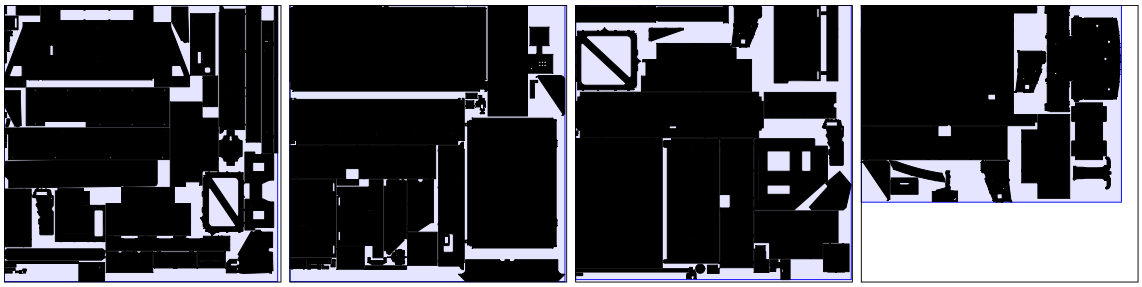


(d) Model *b* trained with 500 parts



(e) Model *b* trained with 2000 parts

Fig. 11 Nesting layouts for experiments in section "Comparison of GRG with open-source Nesting software" on Test Split 1 (regular shapes)

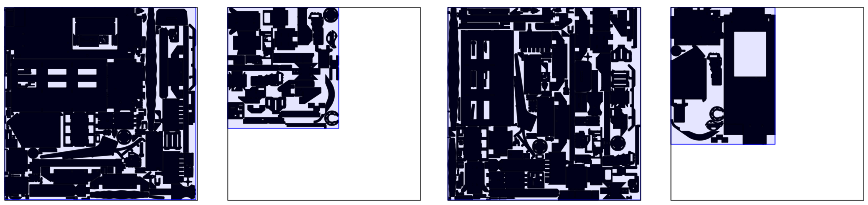


(a) Model c trained with 500 parts

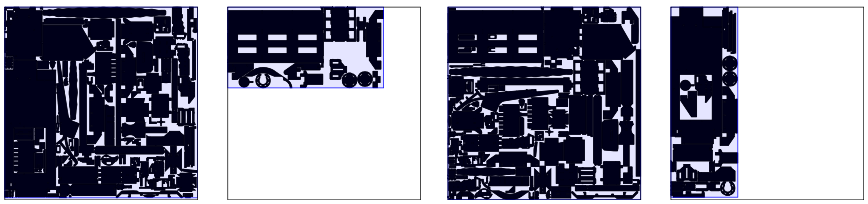
(b) Model c trained with 2000 parts

Fig. 12 Nesting layouts for experiments in section "Comparison of GRG with open-source Nesting software" on Test Split 1 (regular shapes)

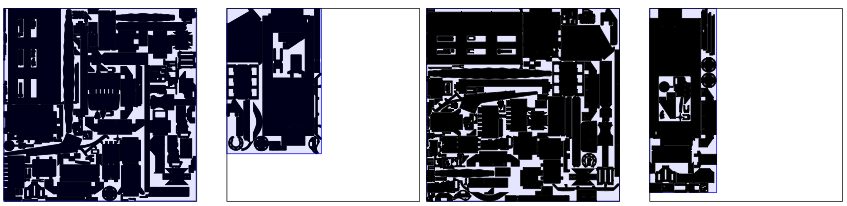
Fig. 13 Nesting layouts for experiments in section "Comparison of GRG with open-source Nesting software" on Test Split 2 (irregular shapes)



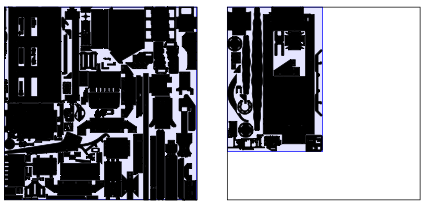
(c) DeepNestPort (d) Model a trained with 500 parts



(e) Model a trained with 2000 parts (f) Model b trained with 500 parts



(g) Model b trained with 2000 parts (h) Model c trained with 500 parts



(i) Model c trained with 2000 parts

References

Arenales, M., Morabito, R., & Yanasse, H. (1999). Special issue: Cutting and packing problems. *Pesquisa Operacional, 19*, 107–299.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics, 5*, 834–846.

Cagan, J., Shimada, K., & Yin, S. (2002). A survey of computational approaches to three-dimensional layout problems. *Computer-Aided Design, 34*, 597–611.

Dyckhoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research, 44*, 145–159.

Faina, L. (2020). A survey on the cutting and packing problems. *Bollettino dell'Unione Matematica Italiana, 13*, 567–572.

Furini, F., & Malaguti, E. (2013). Models for the two-dimensional two-stage cutting stock problem with multiple stock size. *Computers & Operations Research, 40*, 1953–1962.

Grondman, I., Busoniu, L., Lopes, G. A. D., & Babuska, R. (2012). A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, 42*, 1291–1307.

Guo, B., Hu, J., Wu, F., & Peng, Q. (2020). Automatic layout of 2D free-form shapes based on geometric similarity feature searching and fuzzy matching. *Journal of Manufacturing Systems, 56*, 37–49.

Kim, J., Kim, T., Kim, S., & Yoo, C. D. (2019). Edge-labeling graph neural network for few-shot learning. In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, (pp. 11–20).

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In: *Proceedings of the international conference on learning representations (ICLR)*.

Kumar, S. N., & Panneerselvam, R. (2012). A survey on the vehicle routing problem and its variants. *Intelligent Information Management, 4*, 66–74.

Kundu, O., Dutta, S., & Kumar, S. (2019). Deep-pack: A vision-based 2D online bin packing algorithm with deep reinforcement learning. In: *Proceedings of the IEEE international conference on robot and human interactive communication (RO-MAN)*, pp. 1–7.

Labib, R., & Assadi, R. (2007). Modified multi-layered perceptron applied to packing and covering problems. *Neural Computing and Applications, 16*, 173–186.

Liu, J., Ong, G. P., & Chen, X. (2022). Graphsage-based traffic speed forecasting for segment network with sparse data. *IEEE Transactions on Intelligent Transportation Systems, 23*, 1755–1766.

Michalek, J., Choudhary, R., & Papalambros, P. (2002). Architectural layout design optimization. *Engineering Optimization, 34*, 461–484.

Nazari, M., Oroojlooy, A., Snyder, L., & Takac M. (2018). Reinforcement learning for solving the vehicle routing problem. Advances in Neural Information Processing Systems

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. In: *Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD)*, (pp. 701–710).

Perron, L., & Furnon, V. Or-tools. Google. https://developers.google.com/optimization/

Rakotonirainy, R. G., & van Vuuren, J. H. (2020). Improved metaheuristics for the two-dimensional strip packing problem. *Applied Soft Computing, 92*, 106268.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, (pp. 1–9)

Wäscher, G., Haußner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research, 183*, 1109–1130.

Widrow, B., Gupta, N. K., & Maitra, S. (1973). Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics, 5*, 455–465.

Yang, Y., Liu, B., Li, H., Li, X., Wang, G., & Li, S. (2022). A nesting optimization method based on digital contour similarity matching for additive manufacturing. *Journal of Intelligent Manufacturing, 34*, 1–23.

Zhang, D., Shi, L., Leung, S. C. H., & Wu, T. (2016). A priority heuristic for the guillotine rectangular packing problem. *Information Processing Letters, 116*, 15–21.