INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2025, VOL. 71, NO. 2, PP. 387–396
Manuscript received February 19, 2025; revised May, 2025. doi: 10.24425/ijet.2025.153584

Neural-Driven heuristic for strip packing trained with Black-Box optimization

Mariusz Kaleta, Tomasz Śliwiński

***Abstract*—We address the well-known NP-hard problem of packing rectangular items into a strip, a problem of significant importance in electronics (e.g., packing components on printed circuit boards and macro-cell placement in Very-Large-Scale Integration design) and telecommunications (e.g., allocating data packets over transmission channels). Traditional heuristics and metaheuristics struggle with generalization, efficiency, and adaptability, as they rely on predefined rules or require extensive computational effort for each new problem instance. In this paper, we propose a neural-driven constructive heuristic that leverages a lightware neural network trained via black-box optimization to dynamically evaluate item placement decisions. Instead of relying on static heuristic rules, our approach adapts to the characteristics of each problem instance, enabling more efficient and effective packing strategies.**

To train the neural network, we employ the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), a state-of-the-art derivative-free optimization method. Our method learns decision policies by optimizing fill factor improvements over a large dataset of problem instances. Unlike conventional heuristics, our approach dynamically adapts placement decisions based on a broad set of features describing the current partial solution and remaining items.

Through extensive computational experiments, we compare our method against well-known strip packing heuristics, including MaxRects and Skyline-based algorithms. The results demonstrate that our approach consistently outperforms the best traditional heuristics, achieving up to 6.74 percentage points of improvement in packing efficiency. Furthermore, our method improves 87.87% of tested instances. Our study highlights the potential of machine learning-driven heuristics in combinatorial optimization and opens avenues for further research into adaptive decision-making strategies in packing and scheduling problems.

***Keywords*—strip packing problem; algorithm selection problem; heuristics; neural networks; reinforcement learning**

I. Introduction

A. Background

The two-dimensional orthogonal rectangular strip packing problem (2D-SPP) involves arranging a set of rectangular items on a large rectangular strip of fixed width and variable length. The objective is to minimize the strip's length while ensuring that all items are placed entirely within the strip in a non-overlapping manner. The 2D-SPP arises in various practical applications, such as packing components on printed circuit boards, macro-cell placement in Very-Large-Scale Integration (VLSI), allocation of data packets over transmission channels, and the cutting of wood boards, steel plates, or paper rolls.

Since the problem is known to be an NP-hard problem, it is typically tackled using heuristics and metaheuristics [1]. However, heuristics often suffer from poor generalization; there is no single best heuristic, as different heuristics tend to perform best on different problem instances. Metaheuristics, such as genetic algorithms, simulated annealing, and variable neighborhood search, repeatedly explore the solution space for each problem instance, even when the problems are similar. These methods risk getting trapped in local optima and frequently make suboptimal local decisions, and each time they are run, they try many local decisions that later appear to be poor. Their inability to learn from past attempts results in redundant computations, making them inefficient in terms of runtime.

In this paper, we propose a novel hybrid framework that addresses both the challenge of selecting the right heuristic and the inefficiencies of metaheuristics. Our approach leverages a neural network to evaluate available alternatives and uses these evaluations to construct solutions, improving efficiency and adaptability.

B. Related literature

Heuristic approaches to the two-dimensional strip packing problem typically involve two main tasks: selecting the next item to be packed and determining its placement on the strip. A common approach to item selection is to pre-sort items based on a specific feature, such as descending size. More sophisticated approaches have been explored; for instance, Xusheng et al. apply a Q-learning approach to predict the packing sequence while assuming a fixed heuristic for placement [2]. However, most of the research has focused on the placement problem, assuming a predefined ordering of items.

Various greedy heuristics have been proposed to construct solutions by selecting placements based on some scoring function [3]. Constructive heuristics typically choose placements from a set of candidate positions, which can be generated using simple rules such as Bottom-Left or Best-Fit [4], Best Short Side Fit [5], and Occupied Area Ratio [6]. These methods apply to rectangles derived from horizontal or guillotine cuts (shelf and guillotine algorithms) or overlapping rectangles in

Mariusz Kaleta and Tomasz Śliwiński are with Faculty of Electronics and Information Technology, Warsaw University of Technology, Warsaw, Poland (e-mail: mariusz.kaleta, tomasz.sliwinski@pw.edu.pl).

MaxRects algorithms [5]. The Skyline algorithm simplifies MaxRects by tracing only the top envelope of packed items [7]. A similar envelope-based approach was introduced by Martello, Pisinger, and Vigo, who defined Corner Points as non-dominated locations where an item can be placed [8]. Extreme Points, an extension of this idea, further exploits free space within the current packing [9].

Although numerous constructive heuristics exist [10], no universal method for selecting the best heuristic for a given problem instance is known. This challenge is a specific case of the Algorithm Selection Problem, introduced by Rice [11]. Some studies attempt to discover the best-fitted heuristic from data instances. For example, Jipan proposed using a genetic algorithm to learn coefficients for a linear combination of greedy heuristics, improving 93% of cases in the 2D Bin Packing Problem, assuming certain similarities in input distributions [12]. Rakotonirainy used data mining techniques and a dataset of solved strip packing instances to predict the best-performing heuristic based on instance characteristics [13].

The selection and combination of multiple heuristics fall under hyper-heuristic optimization. Beyaz et al. proposed a hyper-heuristic method for 2D packing problems inspired by evolutionary algorithms, incorporating crossover and mutation operators to optimize heuristic selection [14].

Recently, machine learning has emerged as a promising tool for packing problems. One approach involves discretizing the strip into a pixel-like representation and applying reinforcement learning (RL) to make decisions. However, these methods are limited by problem size [15]. In general, solving decision problems with neural networks often involves reinforcement learning, where the network selects an action from a predefined set of decisions. For example, in Atari games [16], the network chooses actions like moving forward, jumping, or staying still based on the game state. However, in combinatorial optimization, the decision set size varies at each step, making direct RL application difficult. One workaround is designing networks with a large superset of possible decisions, but this fails when the decision space is huge or infinite.

Instead of directly solving the problem, machine learning can be used to guide heuristic selection. Fang et al. proposed a pointer network with an encoder-decoder structure to optimize packing sequences, combined with MaxRects-BL for placement [17]. Xu et al. introduced a dual graph neural network, where one network selects the next item to pack, and the other encodes free space geometry [18]. Kai Zhu et al. used a neural network and reinforcement learning as a ”scorer”, evaluating placement candidates [19]. Another approach is supervised learning to classify problem instances and recommend a suitable heuristic. Álvaro et al. developed such a framework, achieving reduced computation time compared to testing all heuristics, though it does not allow for dynamic heuristic switching during solution construction [20].

C. Research gap and contribution

To the best of our knowledge, there is limited research on the use of machine learning, particularly deep reinforcement learning, in the context of constructive heuristics for solving the strip packing problem. Based on our review of the literature, we have identified the following research gaps:

- There is a lack of systematic knowledge regarding which heuristic should be chosen for specific problem instances.
- Specifically, it remains unclear whether deep learning can efficiently address the algorithm selection problem for strip packing. No prior work has explored deep learning from data to determine correlations between instance characteristics and the most suitable heuristics.
- Current research on heuristic selection primarily considers static decisions, meaning that once a heuristic is chosen, it remains fixed throughout the construction of the solution.
- A major challenge in combinatorial optimization is that the number of possible decisions at each step of an algorithm varies significantly, making the direct application of standard reinforcement learning (RL) approaches difficult.

Beyond these research gaps, we are also motivated by the observation that each time a metaheuristic runs, it repeatedly explores a vast search space and evaluates numerous partial solutions—many of which ultimately lead to dead ends. This redundancy could be mitigated by skipping moves that historically (on average) do not contribute to good solutions.

In this paper, we propose a novel framework for strip packing that outperforms traditional constructive heuristics in both efficiency and solution quality. Our main contributions are:

- A new constructive heuristic based on a neural network optimized via an evolutionary algorithm.
- Instead of directly solving the strip packing problem, our approach leverages a neural network to score placement and item selection decisions, making it independent of the variable-size description of the current state as the solution progresses.

By integrating machine learning with heuristic optimization, our method addresses the limitations of existing approaches and offers a more adaptive and efficient solution to the strip packing problem.

D. Paper organisation

The rest of this paper is structured as follows. Section II formally defines the strip-packing problem and introduces the notation. Section III provides a detailed description of our neural-driven constructive heuristic. Then, in section IV, we present numerical experiments, evaluating our approach against standard heuristics. Section V summarizes our findings.

II. Definitions and notations

A. Problem statement

Let $\mathbf{R} = \{r_1, r_2, \ldots, r_n\}$ denote the set n of rectangular items, each defined by a width $w_j > 0$, and a high $h_j > 0$, $j \in \mathbf{R}$. We are also given a strip of width W, a bin of infinite height. The aim is to pack all the items into the

strip, minimizing the total height of all the items packed. We consider orthogonal packing, where rectangular items are placed into the strip such that every edge of each rectangle is parallel to the edges of the strip. The items can be rotated 90 degrees. They must be placed entirely within the strip boundaries and they cannot overlap with each other.

In the problem instances encountered in practice, there are often many duplicates of a given item. It is therefore reasonable to introduce $t \in \mathbf{T}$ the set element types and d_t number of the set items of type t that are to be packed. Then, the set of items R can alternatively be defined as $\mathbf{R} = \{r_1^t, r_2^t, \ldots, r_{d_t}^t\}$, $\forall t \in \mathbf{T}$.

B. Constructive heuristics

Our approach is based on the general principle of constructive heuristics, which iteratively builds a solution. Let $h \in \mathbf{H}$ represent a specific heuristic from the set of constructive heuristics $\mathbf{H}$, where each heuristic satisfies the assumptions outlined in this section. A constructive heuristic h starts with an empty solution, i.e., an empty strip, and iteratively chooses an item and its placement on the strip.

At the beginning of the k-th iteration, the current partial solution is denoted as $s_k \in \mathbf{S}$, where s_k represents the packing configuration after placing k items. The set $\mathbf{S}$ contains all possible partial solutions, i.e., all possible packings of any subset of items.

In iteration k-th a new item must be chosen, possibly rotated, and placed on the strip. Let $y_k \in \mathbf{Y_k} \subseteq \mathbf{Y}$ denote a potential decision in iteration k-th, where $\mathbf{Y_k}$ is the set of all feasible decisions given the current partial solution s_k, and $\mathbf{Y}$ is the set of all possible decisions across any partial solution.

Each decision made by a constructive heuristic transforms the current partial solution into an updated solution. This transformation is represented by a function:

$$T : (\mathbf{S}, \mathbf{Y}) \to \mathbf{S} \tag{1}$$

where $T(s_k, y_k)$ adds one item according to the decision y_k to the current packing s_k, producing a new partial or final solution, s_{k+1}.

The selection of an item and its placement is specific to each constructive heuristic $h \in \mathbf{H}$ and is guided by heuristic rules or criteria, which are typically chosen based on prior knowledge or computational experiments. These criteria often take into account the item's properties (e.g., width, height) and/or characteristics of the current packing state (e.g., wasted space).

Let

$$p_h : (S, Y) \to \mathbb{R}^m \tag{2}$$

denote a property function that assigns an m-dimensional vector to a partial solution s_k and a candidate decision y_k under heuristic $h \in \mathbf{H}$. Typical properties considered in the strip packing problem include item size, aspect ratio, and space utilization.

Each heuristic h evaluates a decision y_k based on properties p_h and greedily selects the decision that maximizes the evaluation function:

$$E_h(p_h(s_k, y_k)) : \mathbb{R}^m \to \mathbb{R}^1 \tag{3}$$

where $E_h(p_h(s_k, y_k))$ assigns a scalar score to the decision y_k at state s_k. For instance, if $p_h(s_k, y_k)$ computes the item's area and the wasted space after the placement, the heuristic might choose the largest item and place it at the position that minimizes wasted space.

The generic constructive heuristic algorithm $h \in \mathbf{H}$ is illustrated in figure 1 and follows these steps:

1. **Initialize**: Set $k = 0$ and initialize the partial solution s_0 (empty strip).
2. **Select a Decision**: Find the decision $\hat{y}_k$ that maximizes the evaluation function E:
$$\hat{y}_k = \arg\max_{y \in Y_k} E_h(p_h(s_k, y_k))$$
3. **Update Solution**: Apply the decision $\hat{y}_k$ to extend the current partial solution:
$$s_{k+1} = T(s_k, \hat{y}_k)$$
4. **Iterate**: Set $k = k + 1$. If the solution is not complete, return to step 2.

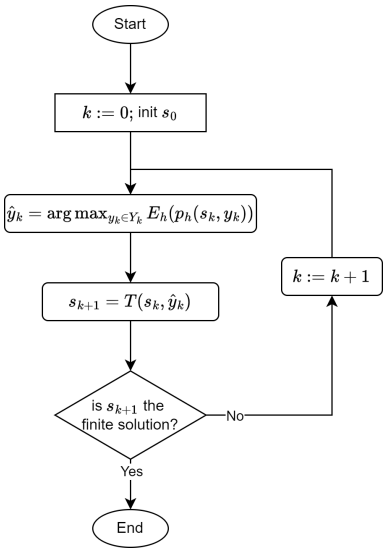

Fig. 1. Algorithm chart of the general constructive heuristic flow

III. Methods

A. General idea

Typically, constructive heuristics rely on a very limited set of properties of the items and the current partial solution. In most cases, this is restricted to one property of the items and one property of the partial solution. For instance, the Maximal Rectangles Bottom-Left (MAXRECTS-BL) heuristic uses the x and y coordinates of all possible placements [21]. The function E selects the placement with the smallest y-coordinate, and if multiple placements share the same y-coordinate, the smallest x-coordinate is used to break the tie. Clearly, the relative importance of these properties varies depending on the specific problem instance.

Furthermore, the function E, which evaluates the properties (e.g., transforms those properties into a scalar), is typically

based on simple and static rules for selecting an item and determining its placement and orientation. For example, in the RectsMax heuristic, items can be sorted according to various criteria, such as descending area, descending perimeter, difference between rectangle sides, shortest/longest side, or aspect ratio (ratio between sides). Similarly, the placement and orientation of an item in the RectsMax heuristic can be based on specific selection criteria, such as minimizing the y-coordinate of the top side of the rectangle (Bottom-Left), picking the smallest available place (Best Area Fit) or choosing the place with minimum length of the longer leftover side (Best Long Side Fit). However, once the sorting order and placement rules are chosen, they remain static throughout the algorithm's execution and do not adapt to the current partial solution. This means that for different problem instances, some evaluation criteria may be more or less effective, depending on the specific characteristics of the input data.

Based on the above observations we are aiming at a heuristic that:

- uses a wide range of properties of the current partial packing and uses these properties that are more relevant in a particular problem instance, and
- is characterized by dynamic element selection, orientation and placement, adapted to the current partial solution.

The fundamental idea of the proposed approach is to use a neural network as a function E to evaluate the properties of the available alternatives at each step of the constructive heuristic. The neural network should be able to learn when and which properties are important and when and which orientation and placement lead to a good solution. It is applied separately to each alternative decision in the current step described by the current partial solution. That makes it possible to consider a variable number of potential decisions with the constant size of the neural network input.

The input to the neural network is the vector of properties $p_h(s_k, y_k) \rightarrow \mathbb{R}^m$. A proper set of properties is discussed in section IV where numerical results are presented; however, we expect that size m of property vector is greater than in standard simple heuristics. The neural network output is a scalar evaluation of the decisions y_k at state s_k, assuming that the higher the evaluation, the better the decision. The actual value returned by the network is used to choose the best decision, however the value by itself has no other interpretation, for instance, it is not a probability assigned to a given decision.

The procedure of constructing the final solution is depicted in Figure 2, where the role of evaluator E_h is taken over by $N_{\bar{w}}$, the trained neural network with the weights $\bar{w}$.

Note that in this approach, having the trained network, we do not make any search of the solution space as many metahuristics do, but instead, we perform just a single pass of the construction heuristic that builds the solution.

B. Detailing the constructive heuristic for the strip–packing problem

In the most general case, at iteration k an item can be placed all over the empty space of the strip. That makes the decision space of the constructive heuristic huge. To make

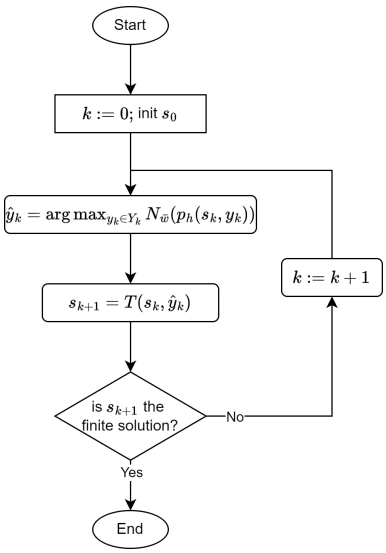

Fig. 2. General view of the neural-driven construction heuristic

it computationally feasible, we follow one of the common approaches that limit the set of possible placing positions to certain, sensible alternatives. In the algorithm proposed by Martello, et al. [8], the authors utilize a set of placing positions called *corner points*. These are bottom- and left-aligned non-dominated locations, resulting from the top envelope of items already placed on the strip, see Figure 3.

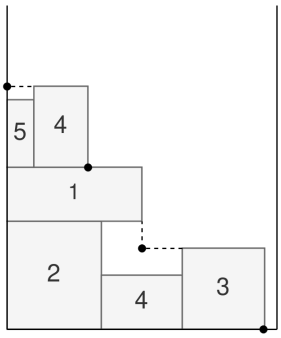

Fig. 3. Corner points (black dots) for a given partial solution.

Although, in the general case, there is no guarantee that the optimal solution of every problem instance can be constructed by placing properly ordered items in the right corner points (see [9], [22]), the approach gives very good results when combined with the proposed neural–driven constructive heuristic.

In each iteration, the neural network $N_{\bar{w}}$ is used to find evaluations for all possible decisions. We define the space of possible decisions $\mathbf{Y_k}$ at iteration k as the set of the following triplets: item type, rotations (no rotation or 90 degrees rotation), and feasible corner point. Only those element types for which there are still unpacked elements in iteration k are taken into account. The number of feasible corner points depends on the current partial solution s_k and the item and varies through

the algorithm execution. So, the decision $\hat{y}_k$ is to choose a triplet y_k consisting of item type, its rotation, and a placement position that maximizes the evaluation $N_{\bar{w}}(p_h(s_k, y_k))$. For example, in the situation depicted in Figure 4, there are six item types illustrated at the top of the figure. However, all items of types 2 and 4 are already packed. There is still one item of types 1, 3, and 5, and three items of type 6 left. In the current iteration, four item types are considered together with the three corner points (the rightmost corner point is ruled out, as the accompanying free space is smaller than any of the dimensions of the small items). Thus, the total number of alternative decisions equals $4 \cdot 2 \cdot 3 = 24$.

As discussed in this section, the algorithm considers a population of neural networks that evolves. Therefore, we want the evaluation procedure to be very efficient, so we use a small-scale and simple feed-forward neural network for $N_{\bar{w}}$. The network has a simple four-layer feed forward architecture with 22 inputs and one output. While the output is a numerical evaluation, the inputs are discussed in depth in the following subsection. The hidden layer sizes are 32 and 12, respectively. The hidden layers and the output are equipped with bias values. Only hidden layers utilize the $tanh$ activation function. The total number of network parameters (weights) is $1,145$. Initially, all the weights are set to 0.

C. Inputs to the neural network

The properties evaluated by the neural network try to reflect the state of the strip and the remaining items that are to be placed, and the actual decision. Selecting the right properties is an important design element that deeply impacts on the quality of solutions. A neural network should know a wide set of properties that allow it to take into account different features depending on the current situation and thus provide generalization. On the other hand, too many properties makes the neural network too complex and less efficient.

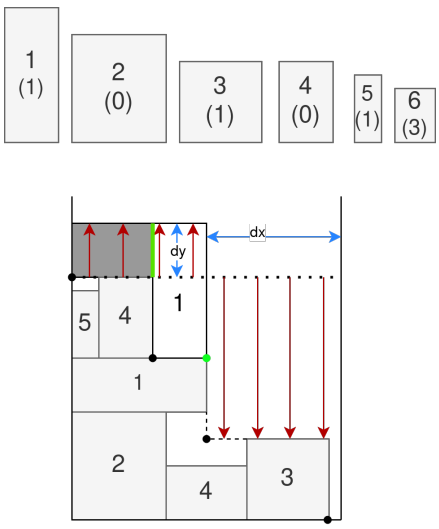

Fig. 4. Making decisions for a partial solution: item types with number of instances to pack given in brackets, current packing with corner points (black dots), and various properties visualized with arrows.

The selection of properties resulted from expert knowledge of packaging problems and a series of experiments with different property configurations. Finally, we selected 22 properties that resulted in the best outcomes. To introduce the properties, let us consider the situation illustrated in Figure 4. The current height of the strip (before inserting the next item) is marked with a dotted line. The decision that needs to be evaluated is the insertion of the new (white) item of type 1 into one of the corner points. Figure 4 illustrates one of the possible placements of the item. The vector of properties quantifying this decision consists of the following properties:

I. Information on the item being placed in the current step
   - Width
     w of the new item (after possible rotation).
   - Height
     h of the new item (after possible rotation).

II. Information on the remaining items
   - Remaining area.
     The total area of all the remaining items.
   - Type's remaining area.
     The area of all the remaining items of a type that is about to be inserted (type of the new item).
   - # items
     of all Number of remaining items.

III. Information on the state after the placement
   - 1D view of the strip state.
     This is the eight-long element vector containing the distance of the non-dominated envelope of packed items (including the new item that is about to be inserted) from the current height of the strip. The distances are marked with red arrows in Figure 4. Arrows pointing down express positive, and arrows pointing up negative values. The distance is sampled in positions evenly spread over the width of the strip.
   - Horizontal mismatch.
     This property quantifies how well the envelope of the new item is aligned with adjacent items horizontally. In Figure 4, this value is represented by the length of the green section connecting the corners of the new item and the item of type 4. Its value may be positive or negative, depending on the alignment.
   - Vertical mismatch.
     This property quantifies how well the envelope of the new item is aligned with adjacent items vertically. In Figure 4, this value is represented by the green dot in the corner of both items of type 1. The dot means they are perfectly aligned.
   - Wasted space.
     The total space wasted when placing the new item, colored with dark gray. This waste of space is due to the utilized *corner points* approach to the generation of possible placement positions.
   - Horizontal size fit.
     Let dx be the distance between the right side of the new item and the right wall of the strip,

and let w be the width of the new item (after possible rotation). Then the horizontal size fit is computed as $(dx \bmod w)/w$, where w is the width of the item. The measure tries to quantify how well other items of the same type as the new item would fit into the remaining space if there were enough of them and they were inserted into this space.

Vertical size fit.
: Let dy be the distance between the top side of the new item and the current height of the strip. If dy is negative, the property is equal to -1. Otherwise, the vertical size fit is equal to $(dy \bmod l)/l$, where l is the length of the item. The measure tries to assess the possibility of placing items of the same type on top of the new item without changing the current strip height.

Horizontal distance.
: The horizontal distance of the new item from the strip's right wall, denoted with dx in Figure 4.

Vertical distance.
: Similarly, the vertical distance from the current strip's height, marked with the vertical blue arrow in the Figure.

Horizontal and vertical positions.
: Similar to horizontal and vertical distance, but in relation to the bottom and left sides of the new item placed at certain corner point rather than to the top and right sides of the new item.

Values of each property are scaled, and whenever the network should focus on smaller values, but larger ones (outliers) can also occur, we apply a hyperbolic tangent function to fit values to $[-1, 1]$ range (these include the following properties: remaining area, number of the remaining items, 1D view of the strip state, horizontal and vertical mismatch, wasted space, vertical distance and position).

D. Network training with black-box optimization

Since the neural network is embedded within a combinatorial construction heuristic, and there is no explicit way to determine the correct network outputs, typical backpropagation learning schemes cannot be applied. However, black-box optimization methods for neural network training have gained significant attention in the machine learning community [23]–[25]. Therefore, we chose black-box optimization as the most versatile approach for computing the neural network's parameters $\bar{w}$ (weights).

Among various evolutionary strategies, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is widely regarded as a state-of-the-art derivative-free optimization method. It is particularly suitable for black-box optimization since it is derivative-free and relies solely on function evaluations [26].

The general training process is illustrated in Figure 5. The algorithm iteratively processes a population of individuals, where each individual represents a set of neural network weights and, consequently, a distinct neural network. Each individual from the population of neural networks is evaluated using the black-box function, meaning that the associated neural network is embedded in our constructive heuristic. The result of this application is an average fill factor, which estimates how well the individual performs. CMA-ES utilizes these evaluations, along with estimations of the mean and covariance matrix, to generate an offspring population and update the distribution parameters. The sole objective of the optimization process is to maximize packing efficiency, i.e., the fill factor.

Since evaluating the black-box function for every training instance would be time-consuming, we employ an approach where only a subset of training instances (a batch) is used in each iteration. In the experiments described in the following section, we randomly select 100 problem instances from a total of 500,000 instances for evaluation in each CMA-ES iteration. Consequently, in every iteration, the population is assessed on a new subset of problem instances.

Randomly selecting a relatively small subset in each iteration implies that the objective function effectively changes during optimization. Despite this, the algorithm maintains convergence while keeping computational effort relatively low. Moreover, this unconventional approach helps prevent the algorithm from getting stuck in local optima.

It is important to note that, unlike many other optimization algorithms [27], CMA-ES is based on the ranking of evaluation values rather than their absolute values. This means that the specific numerical values of evaluations are less relevant; instead, the ordering of individuals based on performance drives the optimization process. This property enhances the algorithm's robustness and facilitates convergence even when the objective function varies during training.

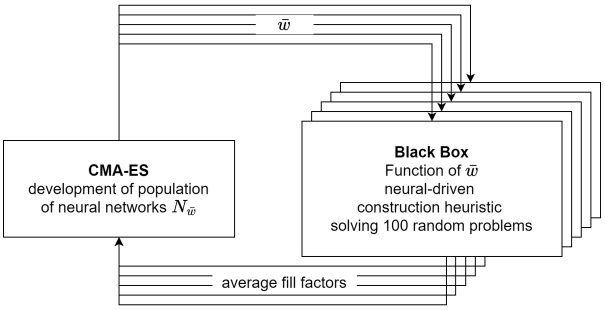

Fig. 5. Black-box optimization with CMA-ES

Every 10th iteration, the best solution found in that iteration is evaluated against a predefined validation set of 10,000 problems. The best-performing solution on the validation set so far is stored as the final solution of the algorithm.

IV. Computational Experiments

A. Test bed and data generation

Since the proposed neural-driven heuristic aims to replace simple rules for item selection and placement with neural-based rules, we compared it against the best solutions obtained from well-known single-pass heuristic families: MaxRects and Skyline [5].

The original versions of these algorithms assume that items are pre-sorted before packing. To benchmark our algorithm against the best possible solution achievable within these heuristic families, we check each heuristic with all basic ordering such as:

- descending by
  - item area,
  - item perimeter,
  - the absolute value of the difference in the lengths of the sides,
  - shorter side,
  - longer side,
  - the ratio of sides,
- and natural order.

Selection is integrated with placement decisions. The following constructive heuristics were considered for placement [5]:

- MaxRects-Based Heuristics:
  - Maximal Rectangles Bottom-Left (MaxRectsBl),
  - Maximal Rectangles Best Area Fit (MaxRectsBaf),
  - Maximal Rectangles Best Short Side Fit (MaxRectsBssf),
  - Maximal Rectangles Best Long Side Fit (MaxRectsBlsf),
- Skyline-Based Heuristics:
  - Skyline Bottom-Left (SkylineBl),
  - Skyline Bottom-Left Wast Map Improvement (SkylineBlWm),
  - Skyline Min Waste Fit (SkylineMwf),
  - Skyline Min Waste Fit with low profile (SkylineMwfl),
  - Skyline Min Waste Fit with Waste Map Improvement (SkylineMwfWm),
  - Skyline Min Waste Fit with low profile and with Waste Map Improvement (SkylineMwflWm).

To ensure a rigorous comparison, we always compare our results against the best outcome achieved across all of the above heuristics and item ordering.

To generate diverse problem instances, we used 2DCPackGen, a problem generator for two-dimensional rectangular cutting and packing problems [28]. Following the typology of Wäscher et al. [29], we generated problem instances of the *Open Dimension Problem/S* (ODP/S) type.

Each problem instance is defined by the tuple:

$$(W, w_t, l_t, d_t)$$

where:

- $t \in \mathbf{T}$ is the set of item types,
- W is the width of the strip,
- w_t and l_t are the width and length of item type t,
- d_t is the demand (number of items of type t).

The parameters relevant to the ODP/S problem type required by the generator 2DCPackGen are listed in Table I (irrelevant parameters for strip packing were omitted).

Without loss of generality, the strip width was fixed to $W = 1000$ (parameter #4). We tested three different numbers of item types (d_t) (parameter #8):

TABLE I
PARAMETERS OF THE 2DCPACKGEN PROBLEM INSTANCES GENERATOR

#	Parameter name	Value ([min, max])
1	Number of Dimensions:	2
2	Integer seed:	100 or 10000
3	Number of instances:	100 or 510000
4	Width of the strip:	[1000, 1000]
5	Items' size dimensions:	[100, 500]
6	ID of the size and shape characteristic of items:	1, 6, 2 or 16
7	Number of bins (strips):	[1, 1]
8	Number of different item types:	[5, 5], [10, 10] or [15, 15]
9	Number for item type demand:	[1, 5], [1, 10] or [1, 20]
10	ID of the characteristic for the item type demand:	5

- 5 item types,
- 10 item types,
- 15 item types.

Item dimensions (w_t, l_t) were randomly generated as integers in $[100, 500]$ (parameter #5) following one of the four predefined shape and size characteristics:

- small and square (ID=1),
- big and square (ID=6),
- long and narrow (ID=2),
- mixed sizes, i.e. small and square, short and tall, long and narrow or big and square (ID=16),

where the ID corresponds to the 2DCPackGenitem item shape classification (parameter #6).

The item demand distribution was set to a uniform random variable based on the range specified in parameter #9.

To sum up, we tested problems with various numbers of item types (parameter #8), different ranges for the generation of item type demands (parameter #9), and different size and shape characteristics of the generated small item types (parameter #6). Table II provides basic statistics for generated problem instances, showing the average number of items and a heterogeneity measure (defined as the number of item types divided by the average number of items per instance).

TABLE II
STATISTICS OF PROBLEM INSTANCES

num. of item types	item type demand	avg. num. of items	heterogeneity measure
5	1–5	15.0	0.33
	1–10	27.5	0.18
	1–20	52.5	0.09
10	1–5	30.0	0.33
	1–10	55.0	0.18
	1–20	105.0	0.09
15	1–5	45.0	0.33
	1–10	82.5	0.18
	1–20	157.5	0.09

For each problem type and size, we initially generated 510,000 instances with a random seed of 10,000 (parameter #2), out of which 500,000 instances were used exclusively for training, and 10,000 instances were reserved for testing.

During training, the neural network was trained purely on the training set, while the testing set was used to select the best-performing model across training iterations.

To compare our neural-driven heuristic with benchmark heuristics, we generated 100 problem instances (instead of

510,000, parameter #3) for each problem type and size, using 2DCPackGen utility, with a seed value of 1,000 (parameter #2).

For black-box optimization, we used the CMA-ES library developed at Laboratory for Computer Science, Université Paris-Sud (https://github.com/CMA-ES/libcmaes), authored by Emmanuel Benazera and supported by the coauthor of the original method, Nikolaus Hanse. The population size is set to 384 individuals (which is well suited to used CPU with 12 cores and 24 threads), and the initial value of the step-size parameter of the method $\sigma = 0.4$. As the stopping condition, we limit the number of black-box function evaluations to 1,000,000, which is equivalent to 2,605 iterations. We use the Sep-CMA-ES variant of the CMA-ES algorithm, which uses a diagonal covariance matrix for linear computational complexity [30].

B. Results

The average improvements of the neural-driven heuristic compared with competitive heuristics are presented in Table III. For each dataset, our algorithm improved the average fill factor by 0.41 to 6.74 percentage points. It can be observed that the lowest improvement occurs in the case of mixed sizes (ID=16). This is an expected result since this dataset is characterized by the least regularity. Consequently, the higher entropy (irregularity) in the instances mitigates the potential for improvement in our data-driven approach. On the contrary, the best improvement is obtained for large and square items (ID=6). This can be explained by the fact that the neural network considers both horizontal and vertical size alignment, whereas other heuristics neglect the ability to align similar items efficiently.

In most cases, the improvement is higher when the number of items increases from the range 1–5 to 1–10. This is reasonable since a larger number of items provides more opportunities to arrange elements better through flexible neural network rules. Further increasing the number of items also leads to greater improvement. However, interestingly, this trend does not hold for the largest number of item types (15 types). In that case, a moderate number of item types results in the best improvement, particularly for small and rectangular items (ID=1). Intuitively, having a large number of small and rectangular items reduces the opportunity for further enhancement, as many good arrangements are already possible and possibly found by some of the benchmarking heuristics.

Tables IV and V present how often our algorithm is strictly better and not worse, respectively, than the best benchmarking heuristics. In all cases except one, our algorithm is strictly better in more than 50% of problem instances, highlighting its clear advantage. Even in the worst case, our algorithm is, on average, not worse than the best heuristic in 57% of cases, which clearly makes it a better choice than the ensemble of benchmarking heuristics.

In general, as the number of item types increases, the advantage of the neural-driven heuristic becomes more pronounced. For 15 item types, the percentage of improved cases is around 98%, reaching 100% in some instances. Overall, the total percentage of improved cases is 87.87%.

TABLE III
AVERAGE IMPROVEMENT IN PERCENTAGE POINTS OF THE FILL FACTOR ACHIEVED BY THE NEURAL-DRIVEN HEURISTIC COMPARED TO COMPETITIVE HEURISTICS.

num. of item types	item type demand	Size and shape ID 1	6	2	16	avg.
5	1–5	1.83	3.94	4.10	0.41	2.57
	1–10	2.27	4.65	4.00	1.07	3.00
	1–20	2.91	5.02	3.74	1.50	3.29
10	1–5	4.21	4.39	4.07	1.39	3.51
	1–10	3.53	5.22	3.73	1.43	3.48
	1–20	3.58	6.01	4.06	1.70	3.84
15	1–5	3.89	5.56	2.87	1.41	3.43
	1–10	3.64	6.38	3.20	1.81	3.76
	1–20	3.27	6.59	3.06	1.80	3.68
avg.		3.24	5.31	3.65	1.39	3.39

Interestingly, the neural-driven heuristic is almost always (95%–100% of instances) better for large and square items (ID=6) and when the number of items is at its maximum, which also corresponds to the highest improvement rates. However, a high ratio of improved cases is also observed for mixed sizes (ID=16) and 15 item types (84%–97%), despite the relative improvement being modest (1.41%–1.81%).

TABLE IV
SHARE OF TEST PROBLEMS FOR WHICH THE NEURAL-DRIVEN HEURISTIC IS STRICTLY BETTER THAN THE BENCHMARKING HEURISTIC [%]

num. of item types	item type demand	Size and shape ID 1	6	2	16	avg.
5	1–5	66	77	77	50	67.50
	1–10	68	86	83	67	76.00
	1–20	89	92	90	77	87.00
10	1–5	92	91	91	74	87.00
	1–10	89	95	94	80	89.50
	1–20	93	100	97	92	95.50
15	1–5	94	95	91	84	91.00
	1–10	99	100	97	96	98.00
	1–20	98	100	99	97	98.50
avg.		87.56	92.89	91.00	79.67	87.77

TABLE V
SHARE OF TEST PROBLEMS FOR WHICH THE NEURAL-DRIVEN HEURISTIC IS NOT WORSE THAN THE BENCHMARKING HEURISTIC [%]. CHANGES RELATIVE TO TABLE IV ARE HIGHLIGHTED.

num. of item types	item type demand	Size and shape ID 1	6	2	16	avg.
5	1–5	66	77	78	57	69.50
	1–10	69	86	84	68	76.75
	1–20	92	92	91	79	88.50
10	1–5	92	91	92	76	87.75
	1–10	90	96	94	80	90.00
	1–20	94	100	97	93	96.00
15	1–5	96	95	92	85	92.00
	1–10	99	100	97	96	98.00
	1–20	98	100	99	97	98.50
avg.		88.44	93.00	91.56	81.22	88.56

The average fill factors are presented in Table VI. Cases with the lowest improvements, namely mixed sizes (ID=16) and particularly 15 item types, exhibit a relatively high fill ratio. This explains the relatively small improvement achieved by the

neural network heuristic —- these problems are already well-handled by benchmarking heuristics, making further improvement difficult or, in some cases, even impossible. Conversely, cases with the greatest improvements have a relatively low fill factor, indicating significant potential for optimization. This suggests that benchmarking heuristics perform poorly on these problem instances, whereas our neural-driven heuristic successfully overcomes their limitations, leading to greater improvements.

TABLE VI
AVERAGE FILL FACTOR OF 100 TEST PROBLEMS AS COMPUTED BY THE NEURAL-DRIVEN HEURISTIC [%]

num. of item types	item type demand	Size and shape ID 1	6	2	16	avg.
5	1–5	86.14	87.08	89.14	89.37	87.93
	1–10	90.51	89.58	92.32	92.96	91.34
	1–20	93.82	91.13	94.37	93.83	93.54
10	1–5	92.43	90.19	93.33	94.40	92.58
	1–10	94.17	91.79	94.78	95.68	94.11
	1–20	96.21	93.04	96.41	96.91	95.64
15	1–5	93.77	92.19	94.45	95.34	93.94
	1–10	95.72	93.52	96.20	96.78	95.56
	1–20	97.28	94.25	97.02	97.54	96.52
avg.		93.34	91.42	94.22	94.87	93.46

V. CONCLUSIONS

In this study, we proposed a novel constructive heuristic for the two-dimensional offline strip packing problem with rectangular shapes. Unlike traditional constructive heuristics that rely on static, predefined rules, our approach leverages a neural network to evaluate candidate placements dynamically during the solution construction process.

By utilizing a broad range of properties of the partial solution and candidate item, our neural-driven heuristic not only generalizes existing heuristic families but also makes the Algorithm Selection Problem irrelevant. More importantly, it allows for dynamic adjustment of selection criteria, including item choice, rotation, and placement, providing greater adaptability and performance potential compared to static heuristics.

One of the main challenges in applying neural networks to combinatorial optimization problems is handling variable input sizes. We overcome this by using the neural network purely as an evaluator within a general constructive heuristic framework. However, this introduces another challenge: the difficulty of training the network via backpropagation, since the decision-making process is non-differentiable. To address this, we employ an evolutionary algorithm for training, which is well-suited for optimizing small neural networks while avoiding local optima in weight updates.

Our experimental results demonstrate that the proposed method consistently outperforms the best heuristics from the Skyline and MaxRects families across different problem instance characteristics. In 24 out of 36 data sets our algorithm is the best in at least 90% of instances. In only one specific case, our method does not strictly outperform competing heuristics in most instances. However, even in that case, it is still better on average, improving the average packing quality.

In conclusion, our proposed neural-driven constructive heuristic successfully mitigates the problem of heuristic selection and achieves better packing solutions on average. This approach opens the door for further improvements in machine-learning-driven combinatorial optimization and suggests that similar methods could be extended to other packing and scheduling problems.

REFERENCES

[1] A. Lodi, S. Martello, and M. Monaci, "Two-dimensional packing problems: A survey," *European Journal of Operational Research*, vol. 141, no. 2, pp. 241–252, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221702001236

[2] X. Zhao, Y. Rao, and J. Fang, "A reinforcement learning algorithm for the 2d-rectangular strip packing problem," *Journal of Physics: Conference Series*, vol. 2181, no. 1, p. 012002, jan 2022. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/2181/1/012002

[3] J. Oliveira, A. Neuenfeldt Júnior, E. Silva, and M. Carravilla, "A survey on heuristics for the two-dimensional rectangular strip packing problem," *Pesquisa Operacional*, vol. 36, pp. 197–226, 08 2016.

[4] E. K. Burke, G. Kendall, and G. Whitwell, "A new placement heuristic for the orthogonal stock-cutting problem," *Operations Research*, vol. 52, no. 4, pp. 655–671, 2004. [Online]. Available: http://www.jstor.org/stable/30036614

[5] J. Jylänki, "A thousand ways to pack the bin-a practical approach to two-dimensional rectangle bin packing," *retrived from http://clb. demon. fi/files/RectangleBinPack. pdf*, 2010.

[6] J. L. da Silveira, F. K. Miyazawa, and E. C. Xavier, "Heuristics for the strip packing problem with unloading constraints," *Computers & Operations Research*, vol. 40, no. 4, pp. 991–1003, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054812002353

[7] L. Wei, A. Lim, and W. Zhu, "A skyline-based heuristic for the 2d rectangular strip packing problem," in *Modern Approaches in Applied Intelligence*, K. G. Mehrotra, C. K. Mohan, J. C. Oh, P. K. Varshney, and M. Ali, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 286–295.

[8] S. Martello, D. Pisinger, and D. Vigo, "The three-dimensional bin packing problem," *Operations Research*, vol. 48, no. 2, pp. 256–267, 2000. [Online]. Available: http://www.jstor.org/stable/223143

[9] T. G. Crainic, G. Perboli, and R. Tadei, "Extreme point-based heuristics for three-dimensional bin packing," *INFORMS Journal on Computing*, vol. 20, no. 3, pp. 368–384, 2008. [Online]. Available: https://doi.org/10.1287/ijoc.1070.0250

[10] A. Neuenfeldt Júnior, E. Silva, M. Francescatto, C. B. Rosa, and J. Siluk, "The rectangular two-dimensional strip packing problem real-life practical constraints: A bibliometric overview," *Computers & Operations Research*, vol. 137, p. 105521, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054821002616

[11] J. R. Rice, "The algorithm selection problem," ser. Advances in Computers, M. Rubinoff and M. C. Yovits, Eds. Elsevier, 1976, vol. 15, pp. 65–118. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245808605203

[12] J. Huang, "Optimal linear combination of heuristic strategies for 2d rectangular bin packing algorithms," in *2024 IEEE 2nd International Conference on Image Processing and Computer Applications (ICIPCA)*, 2024, pp. 1867–1876.

[13] R. Rakotonirainy, "A machine learning approach for automated strip packing algorithm selection," *ORiON*, vol. 36, pp. 73–, 02 2021.

[14] M. Beyaz, T. Dokeroglu, and A. Cosar, "Robust hyper-heuristic algorithms for the offline oriented/non-oriented 2d bin packing problems," *Applied Soft Computing*, vol. 36, pp. 236–245, 2015.

[15] *Progress in Polish Artificial Intelligence Research 5*, 1st ed. PL: Warsaw University of Technology, 2024. [Online]. Available: https://doi.org/10.17388/WUT.2024.0002.MiNI

[16] J. Fan, "A review for deep reinforcement learning in atari: Benchmarks, challenges and solutions," EasyChair Preprint 6985, EasyChair, 2023.

[17] J. Fang, Y. Rao, and M. Shi, "A deep reinforcement learning algorithm for the rectangular strip packing problem," *PLOS ONE*, vol. 18, no. 3, pp. 1–20, 03 2023. [Online]. Available: https://doi.org/10.1371/journal.pone.0282598

[18] Y. Xu and Z. Yang, "Graphpack: A reinforcement learning algorithm for strip packing problem using graph neural network," *Journal of Circuits, Systems and Computers*, vol. 33, no. 08, p. 2450139, 2024. [Online]. Available: https://doi.org/10.1142/S0218126624501391

[19] K. Zhu, N. Ji, and X. D. Li, "Hybrid heuristic algorithm based on improved rules & reinforcement learning for 2d strip packing problem," *IEEE Access*, vol. 8, pp. 226 784–226 796, 2020.

[20] A. Neuenfeldt Júnior, J. Siluk, M. Francescatto, G. Stieler, and D. Disconzi, "A framework to select heuristics for the rectangular two-dimensional strip packing problem," *Expert Systems with Applications*, vol. 213, p. 119202, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422022205

[21] Chazelle, "The bottomn-left bin-packing heuristic: An efficient implementation," *IEEE Transactions on Computers*, vol. C-32, no. 8, pp. 697–707, 1983.

[22] E. den Boef, J. Korst, S. Martello, D. Pisinger, and D. Vigo, "Erratum to "the three-dimensional bin packing problem": Robot-packable and orthogonal variants of packing problems," *Oper. Res.*, vol. 53, no. 4, p. 735–736, Jul. 2005. [Online]. Available: https://doi.org/10.1287/opre.1050.0210

[23] J. Arabas and D. Jagodziński, "Toward a matrix-free covariance matrix adaptation evolution strategy," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 84–98, 2020.

[24] P. Carvalho, N. Lourenço, and P. Machado, "How to improve neural network training using evolutionary algorithms," *SN Computer Science*, vol. 5, no. 6, p. 664, Jun 2024. [Online]. Available: https://doi.org/10.1007/s42979-024-02972-5

[25] D. Jagodziński, Ł. Neumann, and P. Zawistowski, "Deep neuroevolution: Training neural networks using a matrix-free evolution strategy," in *Neural Information Processing*, T. Mantoro, M. Lee, M. A. Ayu, K. W. Wong, and A. N. Hidayanto, Eds. Cham: Springer International Publishing, 2021, pp. 524–536.

[26] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[27] M. Hüttenrauch and G. Neumann, "Robust black-box optimization for stochastic search and episodic reinforcement learning," *Journal of Machine Learning Research*, vol. 25, no. 153, pp. 1–44, 2024. [Online]. Available: http://jmlr.org/papers/v25/22-0564.html

[28] E. Silva, J. F. Oliveira, and G. Wäscher, "2dcpackgen: A problem generator for two-dimensional rectangular cutting and packing problems," *European Journal of Operational Research*, vol. 237, no. 3, pp. 846–856, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221714002112

[29] G. Wäscher, H. Haußner, and H. Schumann, "An improved typology of cutting and packing problems," *European Journal of Operational Research*, vol. 183, no. 3, pp. 1109–1130, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S037722170600292X

[30] R. Ros and N. Hansen, "A simple modification in cma-es achieving linear time and space complexity," in *Parallel Problem Solving from Nature – PPSN X*, G. Rudolph, T. Jansen, N. Beume, S. Lucas, and C. Poloni, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 296–305.