Computing Maximum Polygonal Packings in Convex Polygons using Best-Fit, Genetic Algorithms and Integer Linear Programs

Alkan Atak
TU Dortmund, Germany

Kevin Buchin
TU Dortmund, Germany

Mart Hagedoorn
TU Dortmund, Germany

Jona Heinrichs
TU Dortmund, Germany

Karsten Hogreve
TU Dortmund, Germany

Guangping Li
TU Dortmund, Germany

Patrick Pawelczyk
TU Dortmund, Germany

Abstract

Given a convex region C and a set of simple polygons with associated profits, the *Maximum Polygon Packing Problem* seeks a non-overlapping packing of a subset of the polygons (without rotations) into C, such that the total profit of the packed polygons is maximized.

To handle instances of various sizes and properties, we present a collection of algorithms for this problem. For large instances, we utilize a greedy best-fit placement strategy. For instances consisting exclusively of rectilinear polygons, we use a specialized greedy best-fit algorithm which handles rectilinear shapes efficiently. For medium-sized instances, we provide a genetic algorithm. Finally, for the smallest instances, we employ an integer linear programming model to obtain near-optimal solutions.

Keywords and phrases Polygon Packing, Nesting Problem, Genetic Algorithm, Integer Linear Programming

Digital Object Identifier 10.57717/cgt.v5i5.87

Supplementary Material Source code: `https://github.com/ls11ae/MPP`

1 Introduction

Packing problems ask to pack a set of items into a larger container such that items do not overlap each other. Packing is a classic and extensively studied geometric optimization problem [15, 17].

In the *Maximum Polygon Packing Problem* (MPPP), the input consists of a convex region C in the plane and a set of simple polygons $\mathcal{P} = \{P_1, \ldots, P_n\}$ as items, each associated with a profit π_i. The objective is to determine a subset $\mathcal{P}' \subseteq \mathcal{P}$ along with corresponding translation vectors $t_i \in \mathbb{R}^2$ for each $P_i \in \mathcal{P}'$ such that the translated polygons satisfy the following constraints:

- The selected polygons lie entirely within the region C, i.e., $P_i^{t_i} \cap C = P_i^{t_i}$ for all $P_i \in \mathcal{P}'$,
- no two selected polygons overlap, i.e., $\text{int}(P_i^{t_i}) \cap \text{int}(P_j^{t_j}) = \emptyset$ for all distinct $P_i, P_j \in \mathcal{P}'$,

where translated polygons are denoted by $P_i^{t_i} := \{p + t_i \mid p \in P_i\}$, and $\text{int}(P_i)$ denotes the interior of P_i. The objective is to maximize the total profit of the selected polygons, i.e., $\sum_{P_i \in \mathcal{P}'} \pi_i$.

Most existing practical algorithms for packing simple polygons have been developed for the strip-packing problem. In this problem, all polygons must be packed into a vertical strip of a given width while minimizing the height. See Hu et al. [13] for a survey of such algorithms.

We adapt several of these algorithms to the MPPP and experimentally evaluate them on the instances of the 2024 Computational Geometry Challenge (CG:SHOP 2024) [6]. To solve small instances, we adjust integer linear programming formulations for the strip-packing problem [3, 16, 20] to incorporate selecting polygons and scoring based on profit. For medium-sized instances we adapt genetic algorithms for strip-packing [9, 11], while for large instances we utilize simple best-fit strategies [1, 2], including a dedicated approach for rectilinear polygons [13].

For more details on the MPPP, see the survey paper by Fekete et al. [6], and for other proposed solutions for this challenge, we refer to [5, 10, 18].

2 Algorithmic methods

In the following, we describe the different solutions used to solve the MPPP. A crucial component of our approaches is evaluating pairwise overlap using the *no-fit polygon* (NFP) to construct a complete *forbidden region* for partially solved instances.

More formally, let P_* be a polygon to be packed into a partially solved instance. To evaluate pairwise collisions, we reflect P_* at the origin, obtaining $-P_* = \{-p \mid p \in P_*\}$. The NFP between P_* and another polygon P is given by their Minkowski sum:

$$\mathrm{NFP}(P_*, P) = P \oplus (-P_*).$$

The interior of this NFP represents the translation vectors that result in the interior of P_* intersecting the interior of P.

We extend this concept to evaluate an entire partially solved instance. Let $\mathcal{P}' \subseteq \mathcal{P}$ be the subset of polygons already packed within a container polygon C. As defined previously, each $P_i \in \mathcal{P}'$ has a corresponding translation vector t_i. We want to determine the complete *forbidden region*, that is the set of translations where P_* would overlap with an already packed polygon or would not lie completely in the container. For this, we construct the union of all placed polygons along with U_C, a large enclosing bounding polygon containing a hole corresponding to the interior of C:

$$C_{\mathcal{P}'} = \left(\bigcup_{P_i \in \mathcal{P}'} P_i^{t_i} \right) \cup U_C.$$

Finally, the forbidden region for the instance is obtained by taking the Minkowski sum of this combined obstacle space and the reflected polygon:

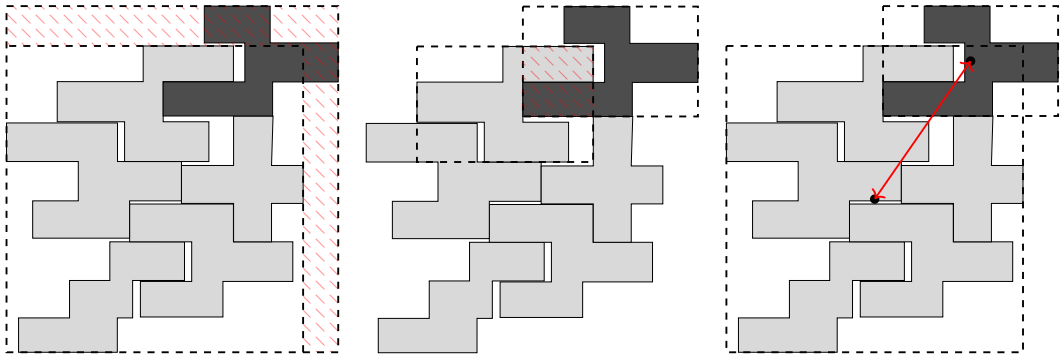
$$\mathrm{FR}(P_*, C_{\mathcal{P}'}) = C_{\mathcal{P}'} \oplus (-P_*).$$

2.1 Best-fit algorithm

A simple heuristic for packing problems is the *bottom-left* method [1], which takes a sequence of items and iteratively places them into the bottom-most, and subsequently left-most, available position. This approach can be combined with a *best-fit* strategy [2], which aims at packing the next item that fits "best" into the current partial solution. This strategy has been used successfully for rectilinear items in particular [12].

We adapt these approaches to our setting. A specific challenge in the Maximum Polygon Packing Problem is balancing geometric compactness with item profit. To aim to pack items as compactly as possible, we evaluate three different placement strategies:

1. **Bottom-Left (BL):** The standard approach of selecting the lowest valid coordinate, breaking ties by the leftmost coordinate.



(a) Waste **(b)** Overlap **(c)** Distance

Figure 1 TOPOS Placement Criteria.

2. **Minimum Distance:** Selecting the position that minimizes the Euclidean distance d_{P_i} from the item's reference point to the container's origin.
3. **TOPOS Strategy:** Based on Oliveira et al. [19], this strategy minimizes a combination of three criteria (see Figure 1): *Waste* (the increase in the bounding box of the partial solution), *Overlap* (between the bounding boxes of the new and existing items), and *Distance* (between the bounding box centers).

Because the original TOPOS algorithm was designed for strip packing, applying it directly to our knapsack setting often results in the undesirable early placement of large, low-value items. To mitigate this, we adapted the TOPOS evaluation function to incorporate the item's raw profit value π_i and its area:

$$\text{Score}_{\text{TOPOS}} = \frac{\text{Waste} + \text{Distance}}{\pi_i} - \frac{\text{Overlap}}{\text{area}(P_i)}$$

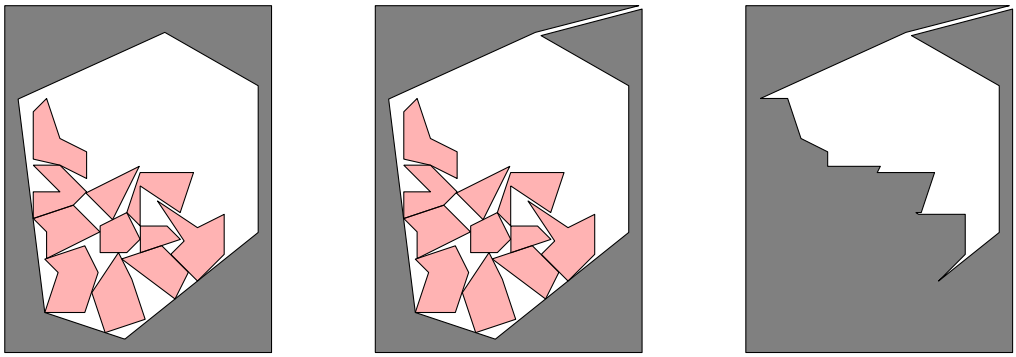
This normalization rewards items with high value density rather than strictly large sizes.

For rectilinear problem instances, we further refine the *Minimum Distance* strategy to explicitly balance geometry and profit. We define r_{P_i} as the area-to-profit ratio of an item P_i, with a smaller ratio being preferable. Let $d_{\min}$ and $d_{\max}$ be the minimum and maximum d_{P_i} among all unplaced items. Likewise, let $r_{\min}$ and $r_{\max}$ be the minimum and maximum area-to-profit ratios among all unplaced items. From the remaining items, we place the item P_i with the lowest score s_{P_i}, where

$$s_{P_i} = \rho \cdot \frac{r_{P_i} - r_{\min}}{r_{\max} - r_{\min}} + \frac{d_{P_i} - d_{\min}}{d_{\max} - d_{\min}},$$

assuming the denominators are non-zero. The parameter $\rho \in [0, 2]$ is optimized experimentally and adjusts the weight assigned to the area-to-profit ratio. If $r_{\max} - r_{\min}$ or $d_{\max} - d_{\min}$ equals 0, we simply choose the item with the smallest d_{P_i} or r_{P_i}, respectively.

The sequential arrangement of items requires a meticulous evaluation of each item within the given problem instance. To further refine the packing configuration, we introduce a preprocessing step wherein we sort the items according to their area-to-profit ratio and selectively remove a proportion of the least valuable items. This strategic elimination ensures a greater utilization of more valuable items during the packing process. To adjust the proportion of items removed, we introduce a parameter, *percentage-removed (PR)*, which is optimized experimentally.



(a) Exact representation of the instance $C_{\mathcal{P}'}$.

(b) Representation with approximated U'_C.

(c) Approximated representation $C'_{\mathcal{P}'}$.

Figure 2 Three ways of representing a partially solved instance.

For problem instances with rectilinear items only, we avoid using generic Minkowski sum computations. Instead, for these instances, we simplify the computation by determining NFPs for rectangles and subsequently combining these. Rectilinear polygons can be partitioned into rectangles. For a pair of rectangles B and B', the NFP(B, B') is simply a rectangle with width and height equal to the combined width and height of B and B', respectively. Afterwards, these NFPs can be combined with simple operations that take the union of two rectilinear polygons. This simplification proves to be effective in improving the running time without compromising the quality of our packing solutions.

For general polygons, to simplify updating feasible placement positions, we first decompose each polygon into convex parts using CGAL's implementation [21] of Greene's dynamic programming method [7]. This speeds up the calculation of the NFPs for two given polygons by taking the union of the Minkowski sums of their convex parts. Moreover, for the bottom-left placement strategy, the forbidden region computation can be further optimized. First, we approximate the container exterior U_C, which is a polygon with a hole, as a simple polygon U'_C. This simple polygon is constructed by connecting the highest point on the right of U_C to the highest point on the right of the enclosing polygon with a narrow corridor (see Figure 2b).

The second optimization transforms the exact combined obstacle space $C_{\mathcal{P}'}$ into a single simple polygon $C'_{\mathcal{P}'}$ of reduced complexity, at the cost of some placement precision. The underlying idea is that in a bottom-left strategy, the space strictly to the left of an already packed item is effectively inaccessible. Therefore, we can include any point within the container that lies left of a packed polygon in the forbidden space.

Specifically, for each packed polygon $P_i^{t_i}$, we construct a *shadow polygon* S_i by projecting $P_i^{t_i}$ horizontally to the left until it meets the boundary of the container. We then take the union of the simple container exterior U'_C and all shadowed polygons. Finally, we obtain $C'_{\mathcal{P}'}$ from this union by removing any remaining holes. These holes may occur because a polygon touching the right boundary can create enclosed empty spaces that are not covered by the leftward shadow projections.

2.2 Genetic algorithm

When iteratively packing items, the sequence in which the items are inserted plays a crucial role in the overall quality of the solution. Finding a good insertion sequence is computationally challenging due to the large number of possible permutations to consider. Consequently,

metaheuristics are employed to navigate the complex solution space efficiently. In particular, genetic algorithms (GAs) have been used successfully for packing rectangular shapes [9] as well as general polygons [11]. Since the genetic algorithm in these approaches is only used to determine the order in which the objects are considered, it is easily adapted to our setting.

A genetic algorithm requires defining *individuals*, an initial *population*, a *fitness* function for the selection procedure, a *crossover* and a *mutation* method. The individuals in our GA's population are the permutations of all polygons of a given problem instance. The permutation is used as the insertion order. Given such an order, our basic approach uses a bottom-left strategy to pack the items. Since the bottom-left strategy may give suboptimal solutions independently of the insertion order, we extend the solution space by assigning a packing strategy to each individual item. The additional packing strategies are the bottom-right, top-left, and top-right strategies, which are analogous to the bottom-left strategy. We refer to this approach, which allows for a different packing strategy for each item, as the *corner strategy*.

For the initial population, we use the following permutations: Firstly, the items sorted according to their profits as well as their area in ascending and descending order; secondly, the permutation obtained by running a modified best-fit algorithm from Section 2.1, where the score function minimizes the vertical position of the bottom-left corner of the bounding boxes; thirdly, random permutations. If the corner strategy is used, we additionally randomize the placement strategy for each item, except for the individuals from the modified best-fit algorithm. As a fitness function, we simply use the final packing score. After each generation's fitness is evaluated, tournament selection is performed to populate the following generation.

We use two standard crossover methods, namely Order Crossover (OX) and Partially Mapped Crossover (PMX). The OX operator generates offspring by copying a randomly selected subsequence from one parent to its children and subsequently filling the remaining positions with the items from the second parent in the order in which they appear, excluding duplicates. In contrast, the PMX operator selects a random sequence of items of one parent solution and establishes a positional mapping between the exchanged elements, thereby ensuring that each item appears exactly once and that valid offspring permutations are produced.

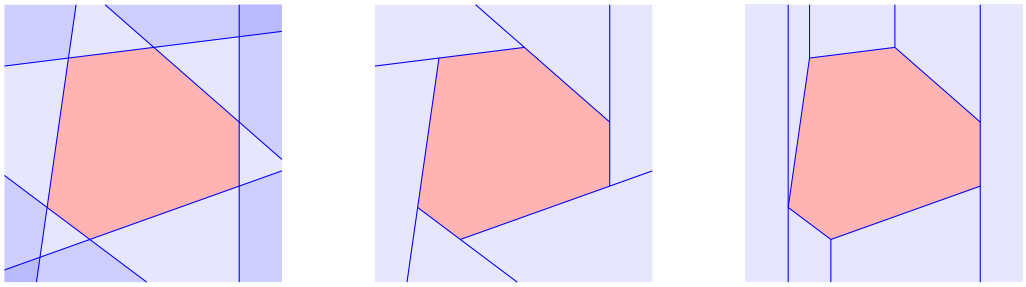
Mutation is applied either by swapping the positions of two items within an individual's insertion order or by altering the current packing strategy, for instance, changing from a bottom-left to a top-right placement heuristic. We conduct experiments with mutation probabilities ranging from 1% to 40%, and for crossover, ranging from 50% to 100%.

For instances with less than 50 items we observe that providing an initial solution by a best-fit algorithm does not guarantee a better overall fitness, even within the initial population. Results vary for different instances regarding crossover and mutation. For instances with more than 50 items, however, providing better initial solutions leads to better overall results with generally higher profits achieved by higher mutation probabilities.

2.3 Integer linear program

For very small instances we make use of an integer linear programming (ILP) formulation based on similar models for the strip-packing problem by Cherri et al. [3], Rodrigues et al. [20] and Díaz & Ortuño [16]. More specifically, we use non-overlapping constraints based on NFPs, see [3, Constraints (16)-(20)] for details. These models require a convex decomposition of the region outside of an NFP, and the main difference between the models is how this is achieved.

The NFP covering model (NFP-CM) [3] extends the edges of the NFP to form overlapping



(a) NFP-CM **(b)** I-NFP-CM **(c)** NFP-CM-VS

Figure 3 Different types of constraints (blue) around an NFP (red).

regions (Figure 3a), resulting in a small amount of constraints but symmetric solutions. To address this, its improved version I-NFP-CM [20] (Figure 3b) adds an additional constraint for each region to make them disjoint. In contrast, NFP-CM-VS [16] (Figure 3c) divides the regions into vertical slices by projecting vertical lines from the NFP's vertices, which are also disjoint but require fewer constraints.

We adapt all of these models to the MPPP. Compared to the strip-packing formulation, we first need to incorporate the convex container. Secondly, we need to handle the selection of items by introducing corresponding decision variables. Because the input instances also specify how often a certain item may be used, we introduce variables $p_{i,k} \in \{0, 1\}$, where $p_{i,k} = 1$ if the k-th copy of item i is packed. The continuous coordinates of a packed item's reference point are given by $(x_{i,k}, y_{i,k})$. In the following, we refer to copies of the same item as items of the same (item) type.

Our objective is to maximize the value of the packed items:

$$\max \sum_{i=1}^{n} \sum_{k=1}^{q_i} \pi_i \cdot p_{i,k}$$

where n is the number of distinct item types, q_i is the available quantity of item i, and π_i is its profit value.

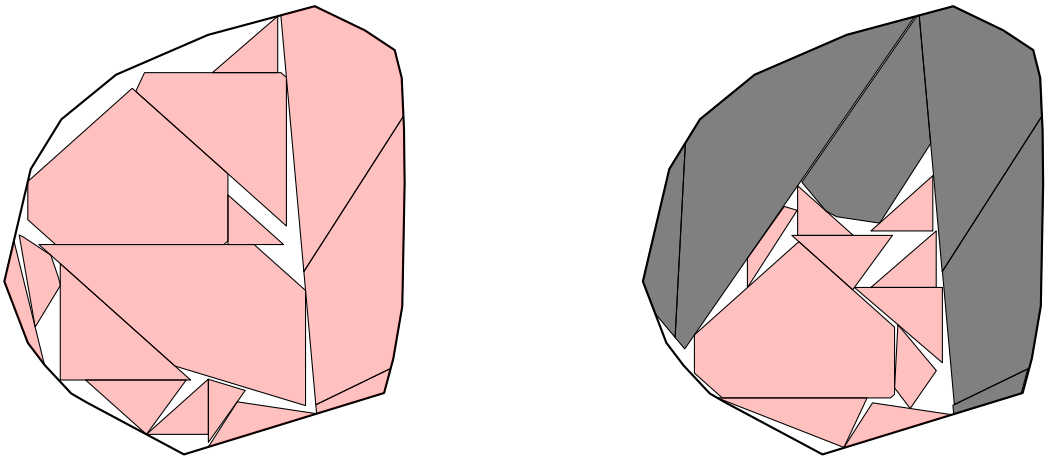
Non-overlapping constraints only need to be fulfilled in case the corresponding items are selected. To handle this conditional nature of the non-overlapping constraints, we utilize the big-M method [4] to trivially satisfy the geometric constraints for any item where $p_{i,k} = 0$. We also introduce an inequality enforcing that the total area of packed items cannot exceed the container's area, which is used to quickly discard solutions that are infeasible based on their total area.

Having items of the same type may lead to a large number of equivalent solutions, which in turn leads to performance degradation. To avoid this, we integrate symmetry-breaking constraints: For any two items of type i, we enforce a lexicographical packing order:

$$p_{i,k} \geq p_{i,k+1} \quad \forall i, \forall k \in \{1, \ldots, q_i - 1\}$$

Furthermore, we break geometric symmetry by enforcing a positional ordering along the main axis, requiring $x_{i,k} \leq x_{i,k+1}$ (for the NFP-CM and I-NFP-CM models) or $y_{i,k} \leq y_{i,k+1}$ (for the NFP-CM-VS vertical slice model).

As the instances used in this setting are generally too large to be solved optimally by our exact formulation, in Section 3 we report on the best feasible solutions obtained after a fixed time limit. We also use our ILPs locally to complete partial solutions (Figure 4).



(a) No initial partial solution, obtaining a score of 1446.

(b) Initial partial solution (dark grey items), obtaining a score of 1518.

Figure 4 Two solutions obtained by the ILP for `jigsaw_rcf4_45bf920f_31`. In the second solution, the dark grey items are packed manually, the ILP is used to select the remaining items.

3 Computational experiments

This section details the computational setup, benchmark instances, and experimental results used to evaluate our algorithms. We first describe the implementation and system specifications, then outline the benchmark data derived from the CG:SHOP Challenge 2024. Finally, we present and discuss the results, highlighting the performance of our methods across different instance types and sizes.

3.1 System specification

The best-fit algorithm for rectilinear items is written in Java, whereas the genetic and best-fit algorithms for arbitrary items are written in C++ using the CGAL 5.6 [21, 22] and Clipper2 [14] libraries, in particular for computing Minkowski sums. Our code repository (`https://github.com/ls11ae/MPP`) also includes several exploratory implementations of other algorithms that were ultimately omitted from this paper as they did not provide an advantage over the presented methods. Moreover, the ILP models were solved using Gurobi 11 [8]. The computations presented were performed on a set of 8 similarly equipped computers, all with an 8-core AMD Ryzen 7 3700X processor running at 4.4 GHz.

3.2 Benchmark data

We tested our implementations using the instances provided for the CG:SHOP Challenge 2024. We divide the instances into rectilinear instances and other instances, sorted by size. We denote *small* instances as having fewer than 100 items, *medium* instances as having 100 up to (but not including) 500 items, and *large* instances as having 500 items or more.

Furthermore, the instances can be divided into four categories:

- `Random`: These instances are created by generating random polygons within a random container. Points are sampled to form convex or concave hulls based on random ratios.
- `Jigsaw`: The container is divided by random lines into convex pieces, which are then merged to create more complex polygons. Combining multiple jigsaw sets produces a

surplus of items, forming challenging packing instances.

- `Atris`: The items are based on Tetris-like shapes; these instances use a rectangular container. Shapes are randomly varied in scale, arm length, and symmetry, and can be flipped or rotated.
- `Satris`: Similar to `atris`, but with an added shearing transformation that skews items based on a random parameter. More heavily sheared items receive higher profits.

For more detailed information on the types of instances, see the survey paper of the CG:SHOP Challenge 2024 [6].

3.3 Experimental results

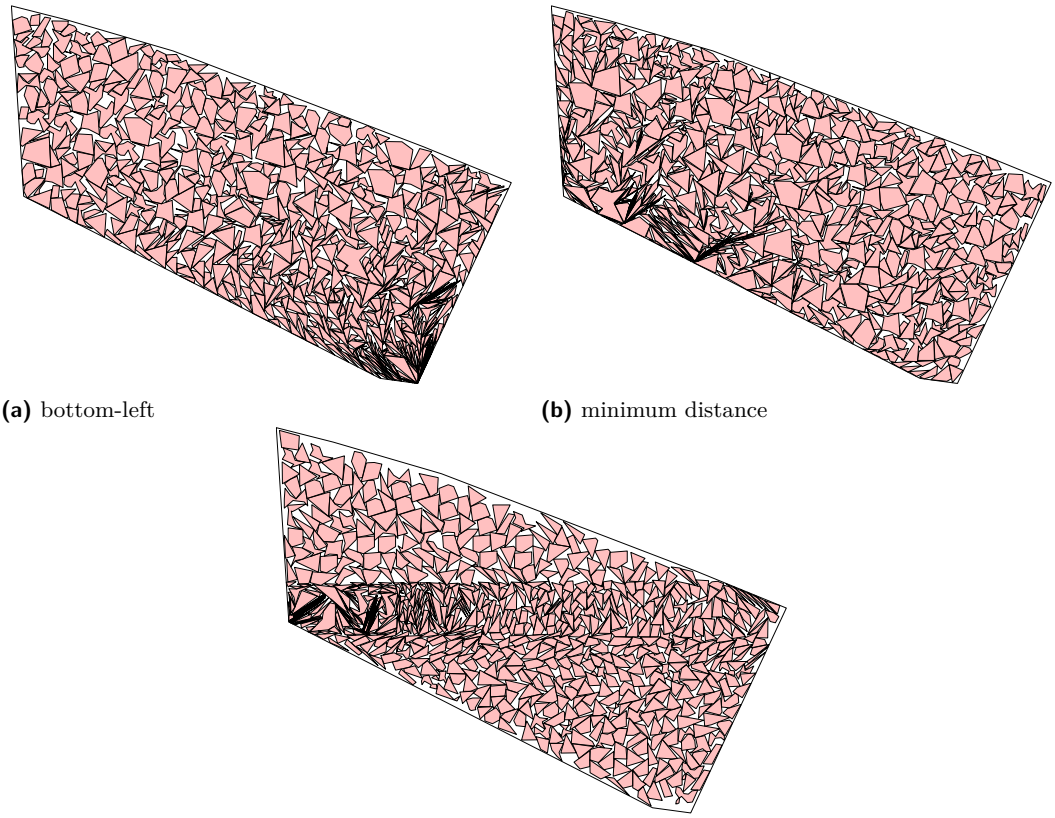
The findings presented in this section are the result of systematic experiments and, therefore, these results vary slightly from the results presented in the official CG:SHOP 2024 rankings, which were obtained through exploratory experimentation.

As explained in Section 2.1, the best-fit algorithm utilizes different methods for calculating NFPs depending on the specific instance types. Table 1 presents the mean relative scores of the different strategies. Each value represents the percentage of the highest score for an instance that was achieved using a specific strategy on average. A value of 1 indicates that a strategy achieved the highest score for all instances within that category.

As the results show, no single placement strategy dominates universally. For highly structured shapes like the `atris` and `satris` instances, the *Minimum Distance* strategy strictly outperforms the others, frequently achieving the optimal known packing. However, for uniformly valued instances (e.g., `cf1` and `ref1`), the *TOPOS* algorithm consistently yields the highest scores. Furthermore, all three best-fit strategies that search for the best-fitting next item give significantly better solutions than the baseline *Bottom-Left* approach without best-fit selection. Figure 5 shows the results of the various placement strategy for one of the instances.

Table 1 Comparison of mean relative score compared to the best result for each best-fit strategy.

Category	Bottom-Left	Minimum Distance	TOPOS	Bottom-Left (no best-fit)
`atris`	0.9671 ± 0.0086	1 ± 0	0.9040 ± 0.0342	0.8602 ± 0.0146
`satris`	0.9702 ± 0.0099	1 ± 0	0.8931 ± 0.0184	0.8198 ± 0.0121
`jigsaw_cf1`	0.9608 ± 0.0449	0.8086 ± 0.0914	0.9994 ± 0.0016	0.8408 ± 0.0694
`jigsaw_cf2`	0.9432 ± 0.0432	0.9901 ± 0.0260	0.9053 ± 0.0710	0.9398 ± 0.0373
`jigsaw_cf3`	0.8688 ± 0.0619	1 ± 0	0.7781 ± 0.1331	0.8324 ± 0.0883
`jigsaw_cf4`	0.9558 ± 0.0626	0.9629 ± 0.0525	0.8689 ± 0.0820	0.9313 ± 0.0081
`jigsaw_ref1`	0.9447 ± 0.0302	0.8059 ± 0.1086	1 ± 0	0.8115 ± 0.0844
`jigsaw_rcf2`	0.9025 ± 0.0516	0.9872 ± 0.0257	0.9169 ± 0.0708	0.8894 ± 0.1794
`jigsaw_rcf3`	0.9645 ± 0.0378	0.9707 ± 0.0597	0.8941 ± 0.0449	0.9087 ± 0.0550
`jigsaw_rcf4`	0.8970 ± 0.0881	0.9516 ± 0.0669	0.9245 ± 0.0786	0.8611 ± 0.0913
`random_cf1`	0.9232 ± 0.0438	0.8221 ± 0.0685	0.9973 ± 0.0093	0.7651 ± 0.0661
`random_cf2`	0.9287 ± 0.0381	1 ± 0	0.8974 ± 0.0812	0.9657 ± 0.0377
`random_cf3`	0.9655 ± 0.0174	1 ± 0	0.9005 ± 0.0249	0.8757 ± 0.0216
`random_cf4`	0.9602 ± 0.0412	0.9740 ± 0.0324	0.9287 ± 0.0515	0.9076 ± 0.0720
`random_ref1`	0.9215 ± 0.0510	0.8590 ± 0.0900	0.9954 ± 0.0139	0.7756 ± 0.1033
`random_rcf2`	0.9194 ± 0.0449	0.9798 ± 0.0470	0.9680 ± 0.0290	0.8914 ± 0.0768
`random_rcf3`	0.9488 ± 0.0030	1 ± 0	0.8505 ± 0.0375	0.9266 ± 0.0896
`random_rcf4`	0.8936 ± 0.0869	0.9699 ± 0.0269	1 ± 0	0.8774 ± 0.1318



(a) bottom-left

(b) minimum distance

(c) TOPOS

Figure 5 Results of different strategies applied to `random_cf1_x53b606b_1000`.

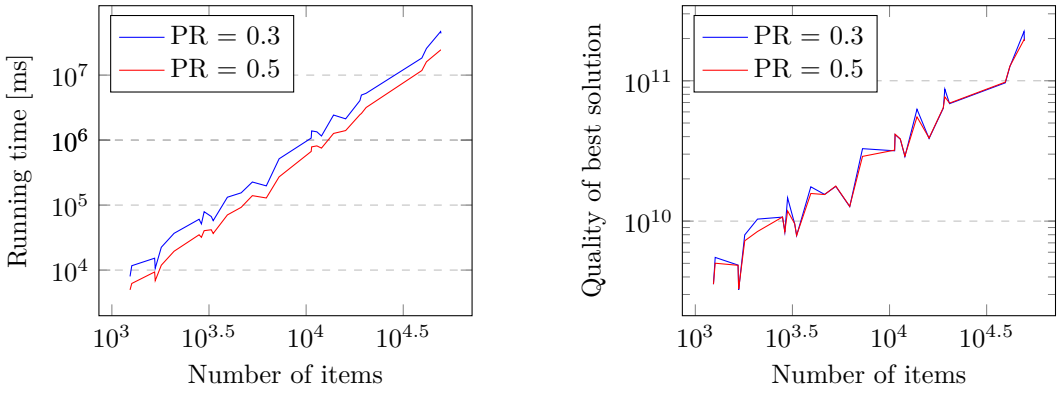


Figure 6 Running time and respective quality of the best-fit algorithm on rectilinear instances (here PR stands for percentage removed as described in Section 2.1).

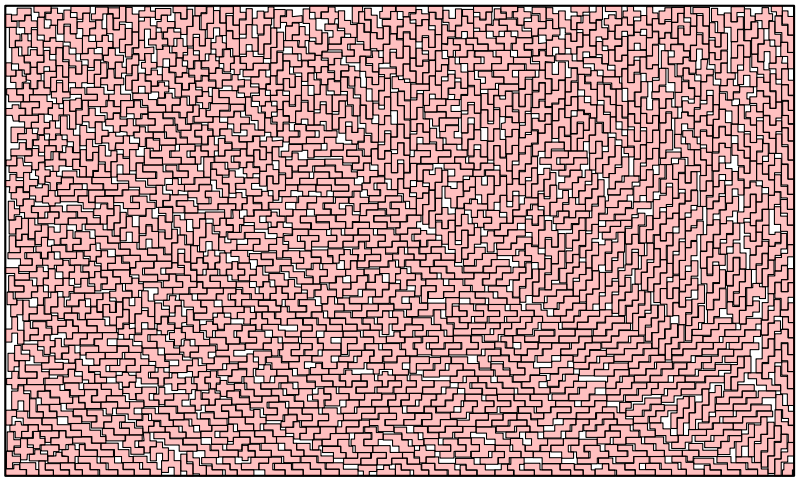


Figure 7 Example solution for `atris2812` (rotated 90 degrees).

Figure 6, shows running time and scores of our approach for rectilinear polygons. This algorithm performs especially well, as demonstrated by the fact that this algorithm found the best solution for 21 out of 29 rectilinear instances in the contest, including the example from Figure 7.

The algorithm considered next is the genetic algorithm as described in Section 2.2. Figure 8 shows how the maximum score evolves over the number of generations, the graphs indicate that after roughly 100 generations we only obtain diminishing returns. However, due to slower running times, we applied this algorithm only to small and medium-sized instances.

The final algorithm presented in this paper (Section 2.3) is the ILP model. Due to long running times, this model can only be executed on small instances. Even then, exact solutions were not found and the best solution was returned after a computation time of 4 hours, in which for most instances between 10^6 and $5 \cdot 10^6$ branch-and-cut nodes were explored by the ILP solver. Furthermore, as can be seen in Figure 9, there seems to be no significant difference between the results for the different ILP variants.

Finally, we provide a brief comparison between the various methods in Figure 10, where we see that the ILP outperforms the genetic algorithm, which outperforms the best-fit algorithm. An example showing a significant difference between the best-fit and the genetic

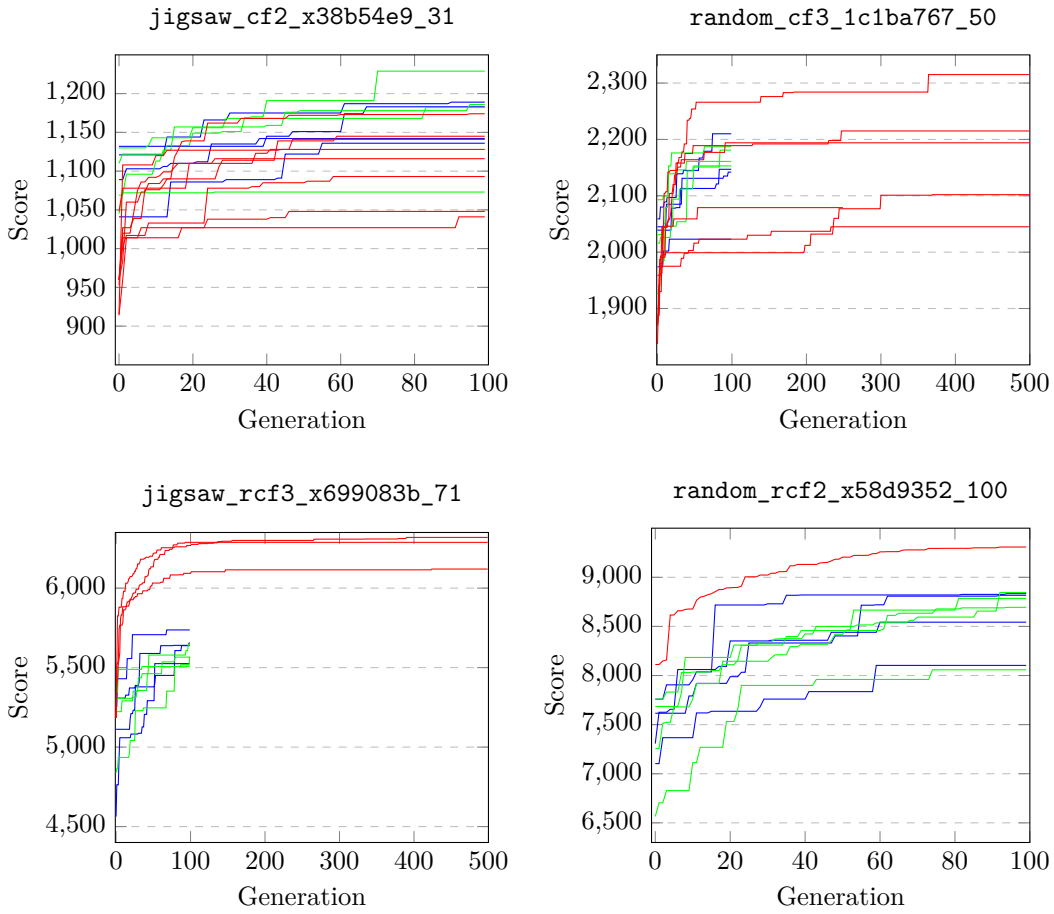


Figure 8 Evolution of maximum score using the genetic algorithms. Red lines indicate a 60% crossover value with swapping items in the mutation phase, blue a 100% crossover value with changing placement strategies, and green a 100% crossover value without mutation (see Section 2.2).

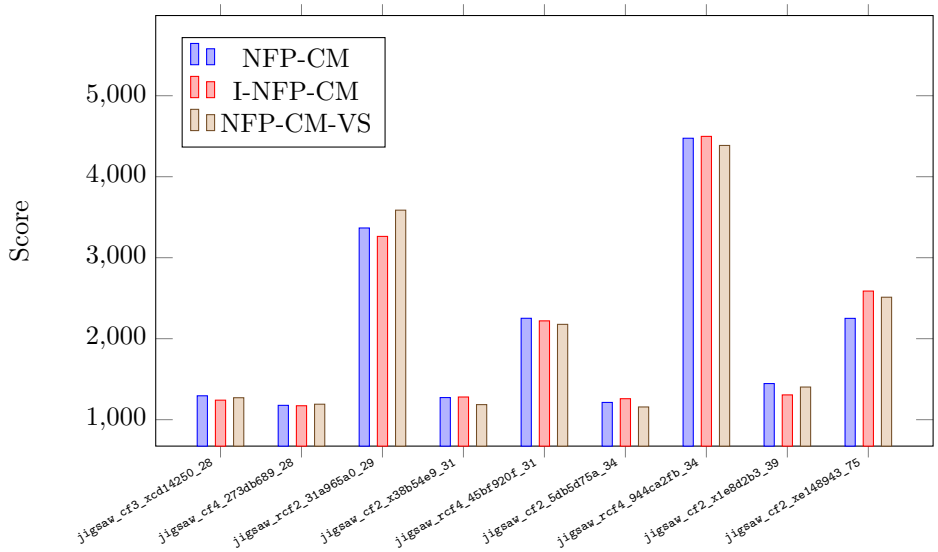


Figure 9 Comparison of results from different ILP models.

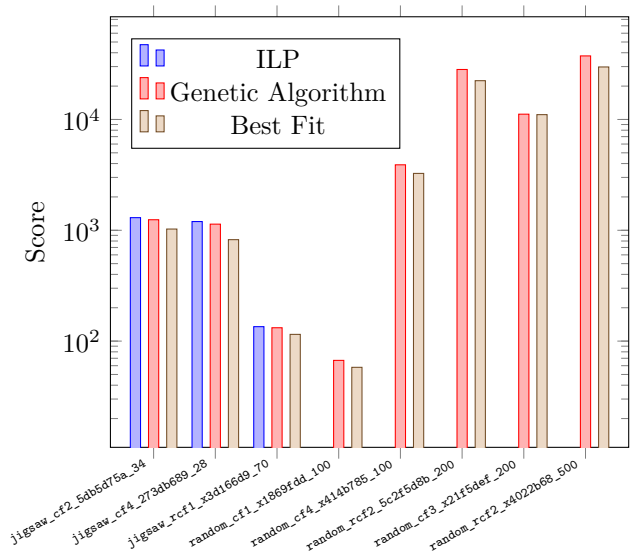
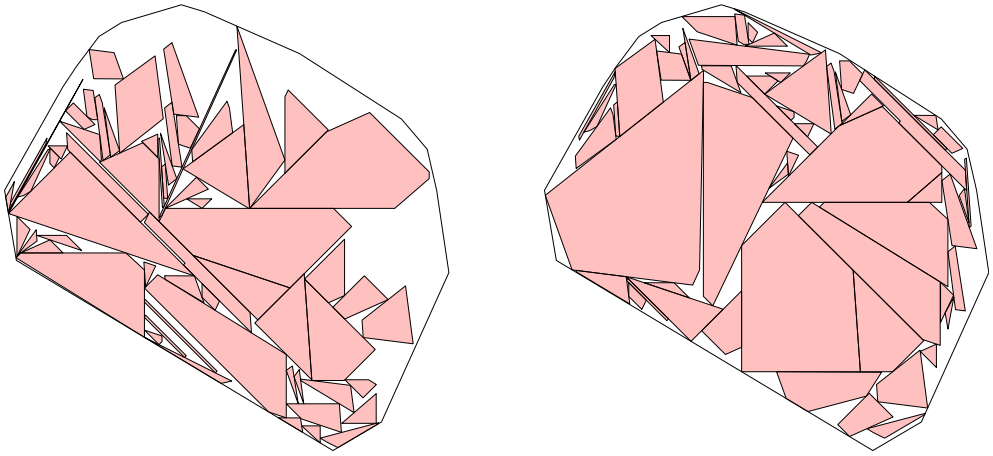


Figure 10 Comparison of results of the genetic algorithm, best-fit algorithm, and ILP.

algorithm is shown in Figure 11.



(a) best fit (minimum distance) **(b)** genetic algorithm

Figure 11 Example for best-fit and genetic algorithm solutions.

4 Conclusion

We present three different approaches to solving the Maximum Polygon Packing Problem (MPPP): an ILP solver, a genetic algorithm, and a best-fit algorithm. We evaluate our approaches on the datasets of the CG:SHOP challenge 2024. Solving the ILP results in the highest scores for small instances but is the least scalable. However, even the small instances were not solved optimally. Therefore, finding more scalable ILP formulations for MPPP is a challenging open problem.

For instances too large for the ILP, we used a genetic algorithm. This approach yields good results for medium-sized instances. However, since evaluating an individual's fitness

requires performing the packing of this individual, this approach becomes slow for large instances. To address this, a different fitness function or a faster way to estimate the score would be needed.

Finally, for the remaining instances, we used the best-fit algorithm, as it is the most scalable and can find a solution for all instances. In particular, for instances with rectilinear polygons, our best-fit algorithm outperformed other approaches in the contest. An interesting direction for further research is to combine the various approaches. For example, an ILP could be used within the best-fit algorithm to solve smaller subproblems optimally.

References

1 Brenda Baker, Ed Coffman, and Ronald Rivest. Orthogonal packings in two dimensions. *SIAM J. Comput.*, 9:846–855, 11 1980. doi:10.1137/0209064.

2 Edmund K. Burke, Graham Kendall, and Glenn Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52(4):655–671, 2004. doi:10.1287/opre.1040.0109.

3 Luiz H. Cherri, Leandro R. Mundim, Marina Andretta, Franklina M.B. Toledo, José F. Oliveira, and Maria Antónia Carravilla. Robust mixed-integer linear programming models for the irregular strip packing problem. *European Journal of Operational Research*, 253(3):570–583, 2016. doi:10.1016/j.ejor.2016.03.009.

4 Richard Cottle, Mukund N. Thapa, et al. *Linear and nonlinear optimization*, volume 253. Springer, 2017. doi:10.1007/978-1-4939-7055-1.

5 Guilherme Dias da Fonseca and Yan Gerard. Shadoks approach to knapsack polygonal packing. In *Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPIcs*, pages 83:1–83:9, 2024.

6 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Stefan Schirra. Maximum polygon packing: The CG:SHOP challenge 2024, 2024. arXiv:2403.16203.

7 Daniel H. Greene. The decomposition of polygons into convex parts. In Franco P. Preparata, editor, *Computational Geometry*, volume 1 of *Advances in Computing Research*, pages 235–259. JAI Press, Greenwich, Conn., 1983.

8 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: https://www.gurobi.com.

9 Africa Gómez and Daniel de la Fuente. Resolution of strip-packing problems with genetic algorithms. *Journal of the Operational Research Society*, 51(11):1289–1295, November 2000. doi:10.1057/palgrave.jors.2601019.

10 Martin Held. Priority-driven nesting of irregular polygonal shapes within a convex polygonal container based on a hierarchical integer grid. In *Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPIcs*, pages 85:1–85:6, 2024.

11 Elizabeth Hopper and Brian C.H. Turton. A review of the application of meta-heuristic algorithms to 2d strip packing problems. *Artificial Intelligence Review*, 16(4):257–300, December 2001. doi:10.1023/A:1012590107280.

12 Yannan Hu, Hideki Hashimoto, Shinji Imahori, and Mutsunori Yagiura. Efficient implementations of construction heuristics for the rectilinear block packing problem. *Computers & Operations Research*, 53:206–222, 2015. doi:10.1016/j.cor.2014.06.021.

13 Yannan Hu, Hideki Hashimoto, Shinji Imahori, and Mutsunori Yagiura. Practical algorithms for two-dimensional packing of general shapes. In Teofilo F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics, Second Edition, Volume 1: Methologies and Traditional Applications*, pages 585–609. Chapman and Hall/CRC, 2018. doi:10.1201/9781351236423-33.

14 Angus Johnson. Clipper2: Polygon clipping and offsetting library. https://github.com/AngusJohnson/Clipper2, 2023. Accessed: 2026.

15 C. Kenyon and E. Remila. Approximate strip packing. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 31–36, 1996. doi:10.1109/SFCS.1996.548461.

16 Juan J. Lastra-Díaz and María Teresa Ortuño. Mixed-integer programming models for irregular strip packing based on vertical slices and feasibility cuts. *European Journal of Operational Research*, 313(1):69–91, 2024. doi:10.1016/j.ejor.2023.08.009.

17 Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002. URL: https://www.sciencedirect.com/science/article/pii/S0377221702001236, doi:10.1016/S0377-2217(02)00123-6.

18 Canhui Luo, Zhouxing Su, and Zhipeng Lü. A general heuristic approach for maximum polygon packing. In *Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPIcs*, pages 84:1–84:9, 2024.

19 José Oliveira, A. Miguel Gomes, and José Soeiro Ferreira. Topos – a new constructive algorithm for nesting problems. *OR Spektrum*, 22:263–284, 01 2000. doi:10.1007/s002910050105.

20 Rodrigues, Marcos O., Cherri, Luiz H., and Mundim, Leandro R. MIP models for the irregular strip packing problem: new symmetry breaking constraints. *ITM Web Conf.*, 14:05, 2017. doi:10.1051/itmconf/20171400005.

21 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.6 edition, 2023. URL: https://doc.cgal.org/5.6/Manual/packages.html.

22 Ron Wein, Alon Baram, Eyal Flato, Efi Fogel, Michael Hemmer, and Sebastian Morr. 2D Minkowski sums. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.6 edition, 2023. URL: https://doc.cgal.org/5.6/Manual/packages.html#PkgMinkowskiSum2.