





A General Heuristic Approach for Maximum Polygon Packing

Canhui Luo  

Huazhong University of Science and Technology, Wuhan, China

Zhouxing Su¹  

Huazhong University of Science and Technology, Wuhan, China

Zhipeng Lü  

Huazhong University of Science and Technology, Wuhan, China

Abstract

This work proposes a general heuristic packing approach to address the Maximum Polygon Packing Problem introduced by the CG:SHOP 2024 Challenge. Our solver primarily consists of two steps: (1) Partitioning the container and polygons to form a series of small-scale subproblems; (2) For each subproblem, sequentially placing polygons into the container and attempting to eliminate overlaps.

2012 ACM Subject Classification Theory of computation → Computational geometry; Computing methodologies → Search methodologies

Keywords and phrases packing, polygon, heuristic, differential evolution, local search, tabu search

Digital Object Identifier 10.4230/LIPIcs.SoCG.2024.86

Category CG Challenge

Funding This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 72101094 and the Special Project for Knowledge Innovation of Hubei Province under Grant 2022013301015175.

Acknowledgements We want to thank the organizers of CG:SHOP 2024 and all other participants for creating such an engaging challenge. We also want to thank Dominik Krupke for providing a helpful official validator for solutions.

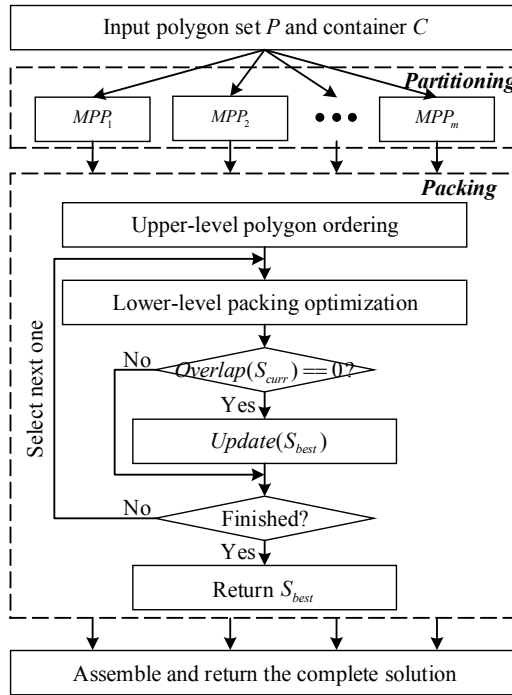
1 Introduction

The recent CG:SHOP 2024 Challenge introduced a variant of irregular packing problems known as the Maximum Polygon Packing (MPP) problem. The MPP problem involves a convex polygonal container C and a polygon set $P = \{p_1, p_2, \dots, p_N\}$, where polygon p_i is associated with a value v_i . It seeks for a non-overlapping packing with the maximum total value. The challenge presents a total of 180 instances whose number of polygons ranges from 28 to 50,000. The official document [4] gives a detailed description of the challenge.

Our proposed algorithm employs a general process to solve these instances indiscriminately, and the overall framework is presented in Figure 1. We first partition a large-scale problem into multiple small-scale subproblems (Section 2) and then solve each subproblem using upper-level polygon ordering (Section 3.1) and lower-level packing optimization techniques (Section 3.2). Section 4 presents our experimental results, followed by conclusions.

¹ Corresponding author: Zhouxing Su





■ **Figure 1** The framework of our proposed algorithm.

2 Partitioning

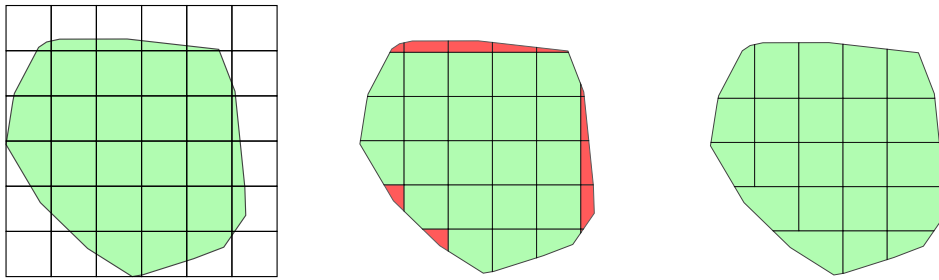
In this section, we present the decomposition of the original large-scale problem into a series of smaller MPP subproblems. It involves two components: partitioning the container C into multiple regions and assigning polygons to each region.

2.1 Container Partitioning

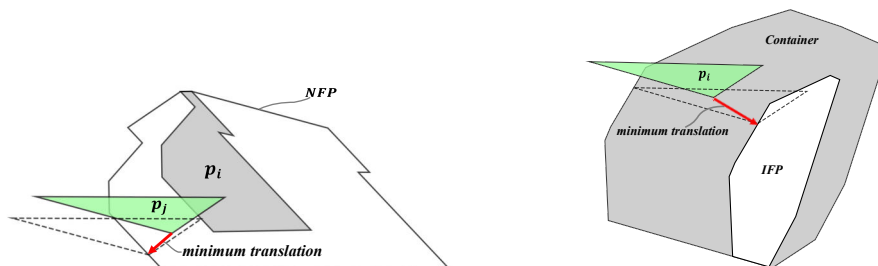
The container partitioning process consists of two steps, as shown in Figure 2. Initially, we arrange two-dimensional square grids starting from the bottom-left corner of the bounding box until the entire container is covered. The subregions formed by the intersection of the container with all the grids constitute its partition $C = C_1 \cup C_2 \cup \dots \cup C_m$. Subsequently, we merge the small subregions with adjacent grids, which are difficult to be used effectively. The grid is dimensioned to keep the scale of each subproblem at approximately 300 polygons, making a trade-off between effectiveness and efficiency of lower-level packing optimization.

2.2 Polygon Assignment

We adopt a simple approach of randomly assigning polygons to each subregion. Specifically, for each subregion C_i , we randomly select a polygon p_j from P until $\frac{\sum_j area(p_j)}{area(C_i)} \geq \frac{\sum_{i=0}^N area(p_i)}{area(C)}$. The advantage of random assignment lies in ensuring that the overall characteristics of each subproblem align with the original problem.



■ **Figure 2** The partitioning process for the instance `jigsaw_cf1_4fd4c46e`. Step 1 (left): Cover the container with squares; Step 2: Intersect and merge small regions (from the middle to the right).



■ **Figure 3** Examples of NFP between two polygons and IFP between container and polygon.

3 Packing

3.1 Upper-Level Polygon Ordering

We define a priority for each polygon. We repeatedly select one remaining polygon with the highest priority (ties are broken by value) and try to insert it into the current solution. If the insertion with lower-level packing optimization fails, we skip the current polygon and turn to the next one. For the majority of instances, the priority is defined as the value-to-area ratio of a polygon (we also call it unit value). Polygons with higher unit values are prioritized for putting in the container, which is called the Unit Value First (UVF) strategy. For small-scale instances ($N < 100$), we employ the $\alpha\beta$ -random strategy. It randomly selects $\alpha\%$ and $\beta\%$ of the polygons and reassigns their UVF-based priority to the highest and the lowest, respectively. These instances are run for multiple times to ensure comprehensive optimization, with α and β set to 10 in our implementation.

3.2 Lower-Level Packing Optimization

The position of a polygon can be represented by the coordinates $l = (x, y)$ of a reference point, such as the bottom-left corner of the boundary. Then, the translation of a polygon can be represented by a vector pointing from its original position to its new position. Given a feasible packing S and a polygon p to be placed, it is impossible to find a non-overlapping position for p without moving other polygons in most cases. This section introduces the algorithm for eliminating overlaps for an invalid packing, which involves solving an unconstrained nonlinear problem and heuristic polygon movement.

3.2.1 Overlap Minimization

To determine the appropriate translation for the polygons, we utilized the no-fit polygon (NFP) and inner-fit polygon (IFP), which are fundamental in algorithmic approaches to geometric design and optimization challenges. For a fixed polygon p_i and a movable polygon p_j , $\text{NFP}(p_i, p_j)$ describes their non-overlapping positions with boundaries in contact precisely, which can be utilized to determine the minimum translation for p_j to avoid overlap. Similarly, $\text{IFP}(p_i, p_j)$ is employed to determine the minimum translation to place p_j inside p_i . Figure 3 illustrates the polygon translations determined using NFP (left) and IFP (right). The readers may refer to Burke et al. [2] for a more detailed description.

For a packing S , based on NFP and IFP, we define the overlap between polygons p_i and p_j as $f_{ij}(S)$, representing the minimum translation to separate them, and $f_{0i}(S)$ as the minimum translation for moving p_i to fit into the container. Subsequently, we employ the separation algorithm proposed by Imamichi et al. [7] to minimize the overlap, which involves solving an unconstrained nonlinear programming problem as follows:

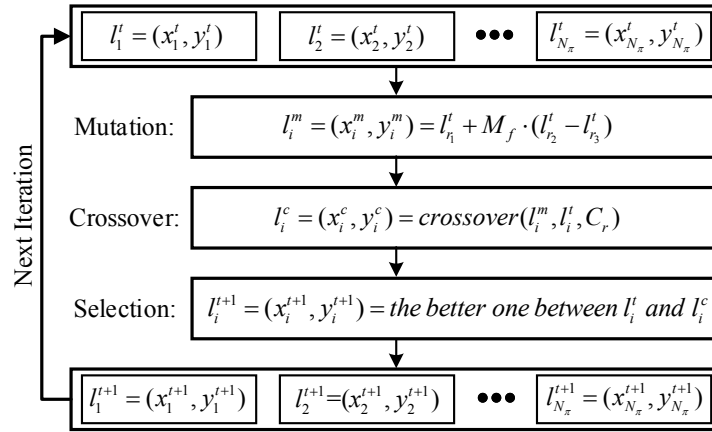
$$\min_S F(S) = \sum_{0 \leq i < j \leq N} f_{ij}^2(S) \quad (1)$$

The model relaxes the non-overlapping constraint but introduces repulsion forces between any two overlapped polygons. We use the classic L-BFGS (limited memory BFGS) method to solve this problem. It makes the packing S converge to a local optimum but strongly depends on the initial layout. The final floating-point computation results will be rounded to integer grids, as the challenge only allows integer coordinates for polygons.

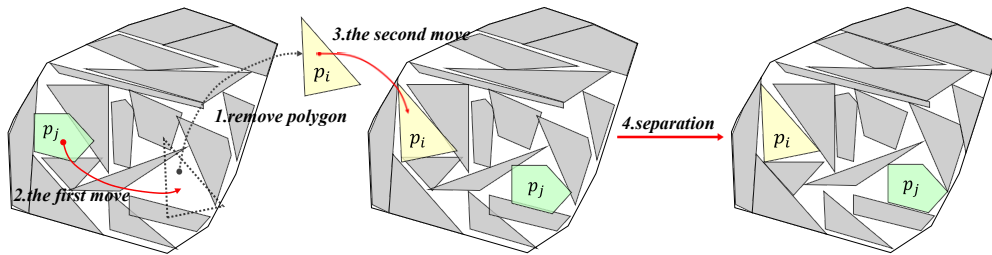
3.2.2 Polygon Move and Swap

The movement of polygons plays an important role in the heuristic strategy of jumping out of local optimum traps, which involves two aspects: inserting a new polygon into the current solution and moving an existing polygon to the position with minimal overlap. We employ Differential Evolution (DE) to optimize the placement of a single polygon. This population-based optimization algorithm iteratively evolves a population of candidate solutions by combining individual differences. It employs a systematic mutation and crossover strategy to explore the solution space efficiently, and the solving framework is depicted in Figure 4.

For a polygon p , we first generate a population $\pi^t = \{l_1^t, l_2^t, \dots, l_{N_\pi}^t\}_{t=0}$ of size N_π , where each individual $l_i^t = (x_i^t, y_i^t)$ denotes a random position within $\text{IFP}(C, p)$. At each iteration, we adopt the “rand/1” [9] mutation operation: randomly select three distinct individuals, $l_{r_1}^t$, $l_{r_2}^t$, and $l_{r_3}^t$, and use the difference between $l_{r_2}^t$ and $l_{r_3}^t$ to perturb $l_{r_1}^t$, obtaining $l_i^m, \forall i \in [1, \dots, N_\pi]$. The differential variation is scaled by a parameter M_f (see the mutation operation in Figure 4). Subsequently, the mutated population undergoes a one-to-one crossover operation with the parent population. The $\text{crossover}(l_i^m, l_i^t, C_r)$ operation randomly preserves the variable value of one dimension from the mutated individual l_i^m (ensuring the offspring to be different from l_i^t), while the other dimension is chosen with a probability of C_r , and gets the crossover result $l_i^c, \forall i \in [1, \dots, N_\pi]$. Finally, the selection operation retains the better one from each pair of l_i^c and l_i^t to form the next generation. The criterion for “better” here is the sum of the squares of overlaps between p and all other polygons (smaller means better). Note that the overlap between p and its original position will also be considered to prevent p from being placed back when trapped into a local optimum. When the maximum iteration number $iter_{de}$ is reached, or a non-overlapping position is found, the algorithm returns the best individual (i.e., the position of polygon p) from the population. The key parameters N_π , M_f , C_r , and $iter_{de}$ in the DE process are set to 20, 1.0, 0.9, and 50, respectively. Additional operation variants and parameter configuration analysis can be found in reference [10].



■ **Figure 4** The iterative framework of Differential Evolution: evolution from generation t to generation $t + 1$ through mutation, crossover, and selection operations.



■ **Figure 5** Polygon *swap* operation, followed by a separation process. Meanwhile, the right subfigure shows the best results for instance `jigsaw_cf1_7b534d0f`, with 18 polygons placed.

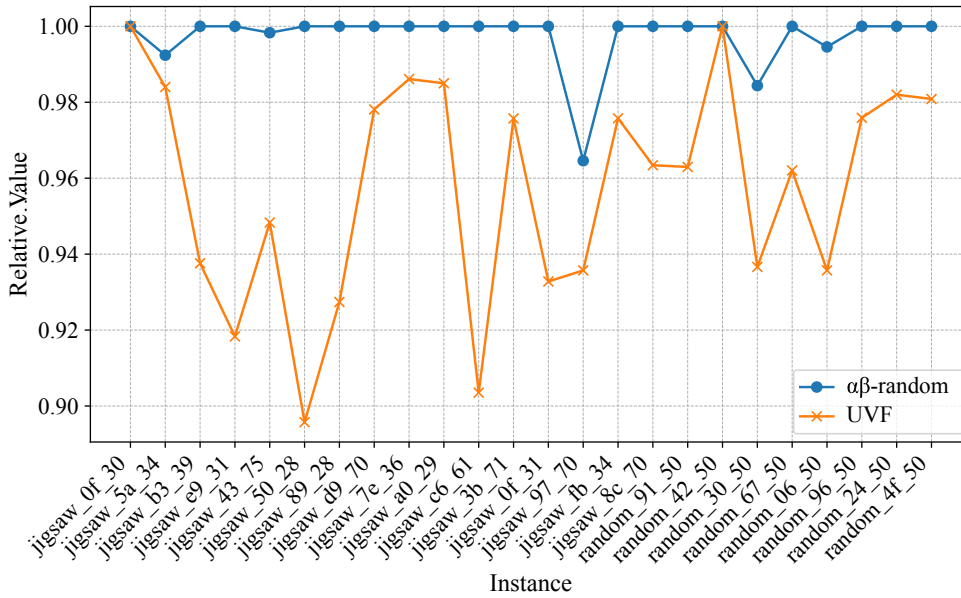
Let $move(S, p_i, S')$ be a movement/insertion of polygon p_i , resulting in S' . We further define $swap(S, p_i, p_j, S')$, the swap of polygons p_i and p_j , as a single move usually results in poor packing. It consists of two moves: (1) $move(S \setminus \{p_i\}, p_j, S_{temp})$, and (2) $move(S_{temp}, p_i, S_{swap})$. The first move releases space by ignoring p_i so that p_j will be moved near p_i with a high probability. Similarly, p_i will be placed near the original position of p_j in the second move. Following the *move/swap*, the separation algorithm can be applied to the new layout, converging it to another local optimum, as illustrated in Figure 5.

3.2.3 Swap-based Local Search

Evaluating all polygon pairs to perform the best swap is time-consuming. It entails $\frac{N \cdot (N-1)}{2}$ evaluations, each involving two moves with a time complexity of $O(N_\pi \cdot iter_{de} \cdot E_{lap})$, where $O(E_{lap})$ for computing overlap between a single polygon and others, along with one separation with a time complexity quadratic in N . Hence, we adopt a random *swap*-based local search. Two polygons are randomly chosen at each iteration, and the *swap* operation, followed by separation, is performed. It is accepted and substitutes the current packing upon obtaining a better solution. The maximum number of iterations is controlled by parameter $iter_{ls}$.

3.2.4 Move-based Tabu Search

In contrast to the *swap* operation, evaluating the best *move* of polygons has lower complexity. After the local search reaches a local optimum, we employ a *move*-based tabu search to explore the solution space comprehensively. Tabu search is a famous metaheuristic algorithm



■ **Figure 6** The comparison between $\alpha\beta$ -Random strategy and UVF strategy on small instances.

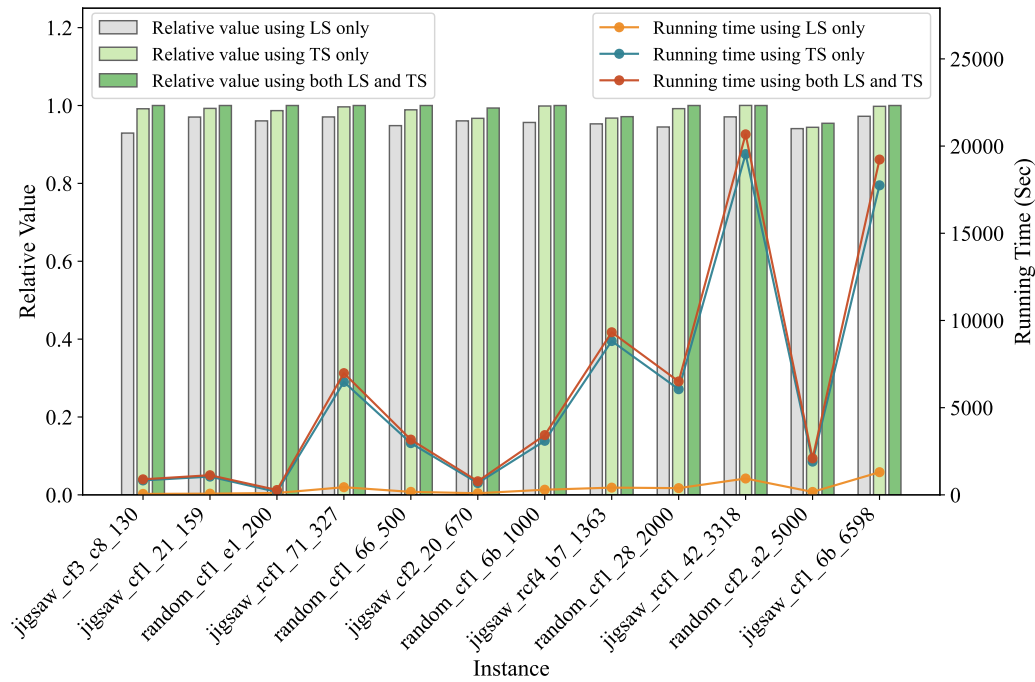
introduced by Glover [5] and can be used for solving combinatorial optimization problems. We adopt a tabu strategy incorporating a recency-based tabu list to prohibit reverting recent operations to escape from the local optimum. In detail, if a polygon is moved at iteration i , it is forbidden to be moved again until iteration $i + T$, where T is the tabu tenure, a uniformly distributed random integer between $[0.2 \cdot N]$ and $[0.5 \cdot N]$. At each iteration, all the non-tabu overlapped polygons are evaluated by *move* operation and separation, and the best move that minimizes $F(S)$ in Equation (1) is performed. Note that the polygons in tabu status can still be moved when applying the separation algorithm. This procedure is repeated until the maximum number of iterations $iter_{ts}$ is reached.

4 Results

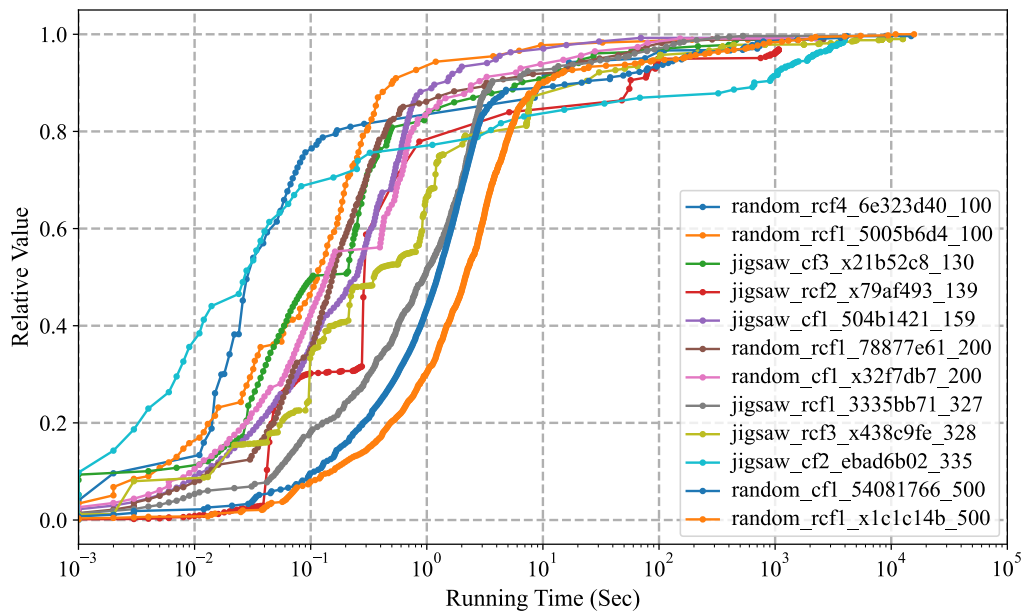
We implement our algorithm in C++ programming language and compile it with Visual Studio 2022. During the CG:SHOP 2024 competition, all instances are sequentially tested on a 2.50GHz Intel Xeon Gold 6133 CPU and 128GB of RAM, and up to 16 threads are employed for parallel optimization of subproblems for a single instance. We use the 2D Minkowski sums package [11] in CGAL for the generation of the NFP and the IFP between polygons, and the L-BFGS library [8] for the implementation of the separation algorithm.

Overall, our proposed algorithm demonstrates advantages on instances with significant variations in polygonal unit values, which aligns with our upper-level UVF strategy. Moreover, we matched the best solution on most small-scale instances ($N < 100$), as shown in Figure 6. It is attributed to the $\alpha\beta$ -random sequential strategy and multiple runs, which confirms the deficiency of the UVF strategy that does not consider the impact of the shape of a polygon.

Figure 7 illustrates that in the lower-level packing optimization, move-based tabu search (TS) plays a major role, and swap-based local search (LS) can provide better initial solutions for TS. Nonetheless, LS also yields satisfactory results in a short running time. The primary time consumption in the optimization arises from the attempts to insert each polygon into the current solution in the late stage, while the improvement to the objective value is mainly obtained in the first tens of seconds, as shown in Figure 8.

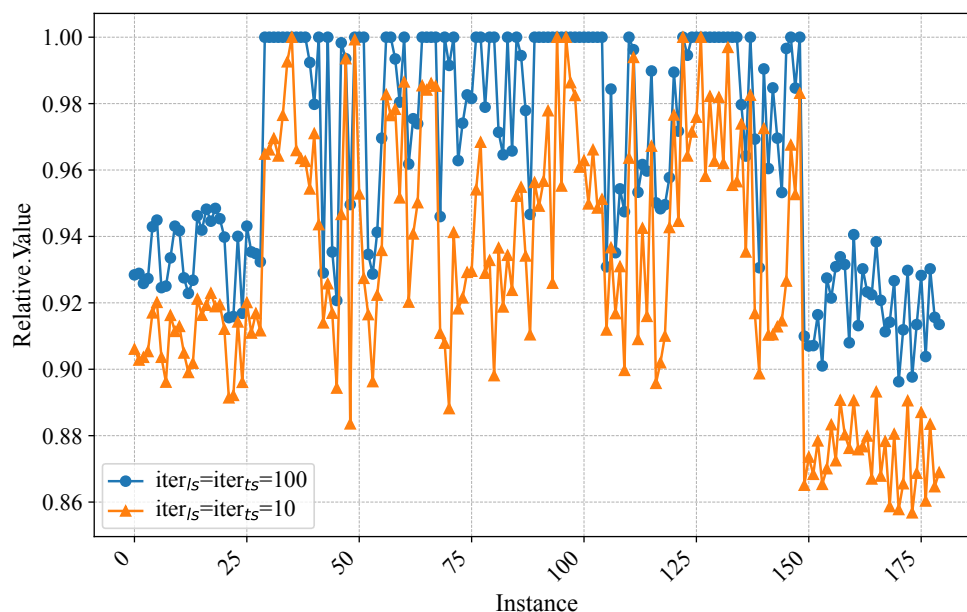


■ **Figure 7** The comparison of relative value and running time of instances under different search strategies, where the relative value refers to the ratio of the value of our solution to the best one.



■ **Figure 8** The relative value over time for the instances with hundreds of polygons, while large-scale instances will be partitioned into multiple parallelizable sub-problems with similar scales.

The maximum number of iterations of local search and tabu search, $iter_{ls}$ and $iter_{ts}$, are critical parameters affecting the solution quality, with a higher number of iterations potentially resulting in higher-quality solutions but also requiring a longer time. In our implementation, both parameters are set to 100. We also conducted experiments under the configuration of $iter_{ls} = iter_{ts} = 10$. Figure 9 compares the results of the two configurations



■ **Figure 9** The performance of the proposed algorithm under different parameters of $iter_{ls}$ and $iter_{ts}$. Each instance is identified by the lexicographic order of its name among all instances.

on all the tested instances. The overall advantage of $iter_{ls} = iter_{ts} = 100$ is about 3.49%.

5 Conclusion

This work introduces a heuristic algorithm with local search and tabu search for solving the maximum polygon packing problem and got the second place in the CG:SHOP 2024 Challenge. The proposed algorithm is versatile in different instance categories without consideration or utilization of the specific characteristics of a polygon. Other challenge participants also proposed various solving frameworks and strategies [3, 6, 1], such as slate preprocessing, integer linear programming, and diverse priority strategies. Integrating these inspiring techniques may improve our algorithm further.

References

- 1 Alkan Atak, Kevin Buchin, Mart Hagedoorn, Jona Heinrichs, Karsten Hogleve, Guangping Li, and Patrick Pawelczyk. Computing maximum polygonal packings in convex polygons using best-fit, genetic algorithms and ilps. In *Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPICs*, pages 83:1–83:9, 2024. doi:10.4230/LIPICs.SoCG.2024.83.
- 2 Edmund K Burke, Robert SR Hellier, Graham Kendall, and Glenn Whitwell. Complete and robust no-fit polygon generation for the irregular stock cutting problem. *European Journal of Operational Research*, 179(1):27–49, 2007.
- 3 Guilherme Dias da Fonseca and Yan Gerard. Shadoks approach to knapsack polygonal packing. In *Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPICs*, pages 84:1–84:9, 2024. doi:10.4230/LIPICs.SoCG.2024.84.
- 4 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Stefan Schirra. Maximum polygon packing: The CG:SHOP Challenge 2024, 2024. arXiv:2403.16203.
- 5 Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

- 6 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. doi:10.4230/LIPIcs.SoCG.2024.85.
- 7 Takashi Imamichi, Mutsunori Yagiura, and Hiroshi Nagamochi. An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. *Discrete Optimization*, 6(4):345–361, 2009.
- 8 Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989. URL: <http://users.iems.northwestern.edu/~nocedal/lbfgs.html>.
- 9 Rainer Storn. On the usage of differential evolution for function optimization. In *Proceedings of north american fuzzy information processing*, pages 519–523. Ieee, 1996.
- 10 Yong Wang, Zixing Cai, and Qingfu Zhang. Differential evolution with composite trial vector generation strategies and control parameters. *IEEE transactions on evolutionary computation*, 15(1):55–66, 2011.
- 11 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>.