A heuristic for solving the irregular strip packing problem with quantum optimization

Paul-Amaury Matt[1]*and Marco Roth[1]

[1]Department of Cyber Cognitive Intelligence (CCI), Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Nobelstrasse 12, 70569 Stuttgart, Germany.

*Corresponding author(s). E-mail(s): paul-amaury.matt@ipa.fraunhofer.de;
Contributing authors: marco.roth@ipa.fraunhofer.de;

Abstract

We introduce a novel quantum computing heuristic for solving the irregular strip packing problem, a significant challenge in optimizing material usage across various industries. This problem involves arranging a set of irregular polygonal pieces within a fixed-height, rectangular container to minimize waste. Traditional methods heavily rely on manual optimization by specialists, highlighting the complexity and computational difficulty of achieving quasi-optimal layouts. The proposed algorithm employs a quantum-inspired heuristic that decomposes the strip packing problem into two sub-problems: ordering pieces via the traveling salesman problem and spatially arranging them in a rectangle packing problem. This strategy facilitates a novel application of quantum computing to industrial optimization, aiming to minimize waste and enhance material efficiency. Experimental evaluations using both classical and quantum computational methods demonstrate the algorithm's efficacy. We evaluate the algorithm's performance using the quantum approximate optimization algorithm and the quantum alternating operator ansatz, through simulations and real quantum computers, and compare it to classical approaches.

Keywords: quantum computing, QAOA, quantum optimization, strip packing problem, Traveling Salesman Problem, irregular packing problem

1 Introduction

The *irregular strip packing problem* is a challenging and economically significant issue that involves fitting a set of polygonal pieces into a fixed-height, rectangular container in a manner that minimizes unused space, or waste. This task has broad implications, affecting industries ranging from fashion to automotive and electronics, where efficient material use is crucial for both cost reduction and environmental sustainability. Traditional approaches often rely on the expertise of specialists using CAD systems to achieve quasi-optimal layouts, highlighting the problem's complexity from both combinatorial and geometric perspectives. The quality of the placements produced by these specialist workers is high and according to [1], automatic solutions can only barely match this level of quality.

In this work, we introduce the *Opus Incertum* algorithm, a quantum computing (QC) heuristic designed to efficiently tackle the irregular strip packing problem. While quantum optimization is a highly active field with applications across a wide range of problems [2], purely geometric problems such as the strip packing problem have received comparatively less attention in this domain. To make the problem tractable for QC, we decompose it into two subproblems, a Traveling Salesman Problem (TSP) which shares similarities with the strip packing problem in terms of computational difficulty and a rectangular packing problem. The TSP can then be solved using QC algorithms such as the Quantum Approximate Optimization Algorithm [3] and a variation thereof called Quantum Alternating Operator Ansatz [4]. Our method works towards applying QC to industrial optimization, offering a promising solution to the irregular strip packing problem by minimizing waste and optimizing material usage.

The remainder of the article is organised as follows. Section 2 is dedicated to related work. Section 3 formally introduces the mathematical formulation of the problem. In Section 4, we introduce a QC-based heuristic. We then present in Section 5 the experimental results of our heuristic when executed on such computers with different methods available and compare the performance with existing classical methods. We finally draw our conclusions in Section 6.

2 Related Work

A recent survey of mathematical models proposed in the last decades for nesting problems can be found in [5]. The integer linear programming models of [6, 7] use a grid for the discrete positioning of pieces. The mixed-integer linear programming models of [8–13] assume continuous positioning of the pieces in the container. [14] introduces a mixed-integer linear programming model with semi-continuous positioning, i.e., continuous positioning on one axis and discrete positioning on the other. [15] is the first paper to tackle the problem of packing irregular shapes with unrestricted rotations. Other mathematical models that have been proposed are non-linear programming and constraint

programming models. All these exact approaches which assume the pieces are polygons allow to find the optimal solution, but only work for small problems.

One of the main difficulties in solving the strip packing problem or other nesting problems is the necessity to represent the problem and perform geometrical computations such as checking if pieces overlap or fit entirely inside the container. [16] provides a survey of the geometric tools available. These include pixel/ raster methods [17–19], direct trigonometry and the D-function [20–23], the no-fit polygon [16, 22, 24–26] and phi-functions [27, 28].

Heuristic methods have been designed in order to solve large instances of the irregular strip packing problem. An often used heuristic is the First Fit Decreasing Algorithm [29], which consists of packing the pieces into bins in the order of decreasing size. The bottom-left heuristic proposed by [30] consists of sequentially placing the pieces in as far as possible in the bottom-left corner without overlapping them with those previously positioned. An initial solution is usually obtained with the bottom-left heuristic and then improved via heuristics, meta-heuristics and compact and separation models. According to [5], some of the best results are obtained with the methods of [31–36].

[37] was the first to propose a quantum inspired algorithm for solving the one-dimensional bin packing problem, where a set of items must be packed into a minimum number of bins. [38, 39] proposed a quantum-classical hybrid approach to solve the one-dimensional bin packing problem using quantum annealing. [40] proposed an Ising model mapping to solve the two-dimensional regular packing problem, in which all pieces are assumed to have a rectangular shape. Using a quantum annealer, they were able to solve problem instances with up to eighteen rectangles.

3 The irregular strip packing problem

The irregular strip packing problem formally involves allocating a set of N pieces $P_0, P_1, \ldots, P_{N-1}$ with polygonal shapes into a rectangular container C with a fixed height H and variable length L. The pieces must be placed completely inside the container in such a way that they do not overlap. The pieces can be placed at any continuous location and can be freely oriented. The objective is to minimize the length L of the container. Since all pieces must be placed inside the container and the height is fixed, minimizing the container length is equivalent to minimizing the unused area of the container, i.e., the surface that is unoccupied by the pieces. The unoccupied surface can be interpreted as *waste* and the amount of waste can be quantified either by the area W of that surface or by the ratio between W and the total area $H \times L$ of the container. An example of a set with eight pieces is shown in Fig. 1(a). From a mathematical perspective, this problem as well as other irregular or regular packing problems, combine the combinatorial hardness of cutting and packing problems with the computational difficulty of enforcing the geometric non-overlap and containment constraints. A quasi-optimal layout for this set

of eight pieces in a container with fixed height which has been designed by hand is provided in Fig. 1(b).

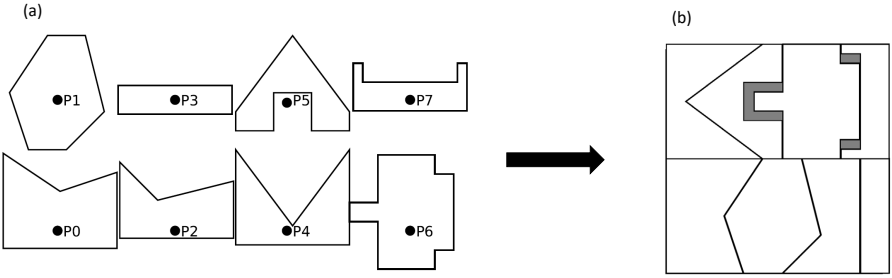


Figure 1: (a) Set of eight pieces to pack. (b) Example of a hand-designed placement of the eight pieces.

4 The Opus Incertum Algorithm

Mathematical models proposed in the literature are solved monolithically, i.e., without use of decomposition methods thereby limiting the size of the problems that can be solved. *Regular packing* (i.e. packing rectangular-shaped pieces) is an exception and constitutes a much simpler problem which can be solved efficiently. To make use of this, we propose a heuristic method where the irregular packing problem is reduced to a regular packing problem. We call the algorithm *Opus Incertum Algorithm*[1]. It consists of the following steps:

1. Compute the geometrical compatibility between pieces
2. Generate groups of geometrically compatible pieces
3. Order the pieces in each group (corresponds to solving a TSP)
4. Spatially arrange the pieces in each group into a compact rectangle
5. Generate candidate partitions of the set of pieces
6. Solve the rectangle packing problem for each partition
7. Local optimization of the layout obtained for each partition
8. Global optimization of the best layout
9. Return the best layout

4.1 Definitions and details

In the following we explain each of these steps in detail and introduce the necessary definitions. The final algorithm is summarized in Algorithm 1.

[1]Opus Incertum, is a reference to an ancient Roman constructing technique of the same name that consists of irregularly shaped and randomly placed uncut stones inserted in a core of concrete. Vitruvius, in De architectura (Ten books on Architecture), favours opus incertum, deriding opus reticulatum (a similar technique using small pyramid-shaped instead of irregular stones) as more expensive and structurally of inferior quality.

4.1.1 Geometrical compatibility between pieces

Let us consider two pieces P_i and P_j. Here, we assume that the position of P_i is fixed and denote (r, θ, ϕ) a placement with polar coordinates (r, θ) of the reference point of P_j with respect to P_i and ϕ the orientation of P_j. We want to find a placement (r, θ, ϕ) of P_j that avoids an overlap with P_i while minimizing the waste. To make this statement mathematically precise, we define the waste as the surface that is inside the *convex hull* of the set of vertices of the pieces. The optimal placement is then the one that avoids overlapping and yields a convex hull with minimum area. We introduce the concept of the *no-fit function*. The no-fit function of two pieces P_i and P_j is the function NFF_{P_i,P_j} that takes as input a polar angle θ and returns the radius r and orientation ϕ to optimally place P_j with respect to P_i given the rotation angle θ. Once computed and stored, the no-fit function allows to find the placement that yields the convex hull with minimum area. The no-fit function is given by

$$\mathrm{NFF}_{P_i,P_j}(\theta) = \underset{r\in\mathcal{R},\phi\in\Phi}{\arg\min} \left\{ \mathrm{area}(\mathrm{CH}[P_i, P_j(r,\theta,\phi)]) \,|\, P_i \cap P_j(r,\theta,\phi) = \emptyset \right\} . \quad (1)$$

Here, $\mathrm{CH}(P_i, P_j)$ denotes the convex hull where the optimal convex hull $\mathrm{CH}^*(P_i, P_j)$ is given by

$$\mathrm{CH}^*(P_i, P_j) := \underset{\theta\in\Theta}{\arg\min} \left\{ \mathrm{area}(\mathrm{CH}[P_i, P_j(r,\theta,\phi)]) \,|\, (r,\phi) = \mathrm{NFF}_{P_i,P_j}(\theta) \right\} . \quad (2)$$

Note that the absolute position of P_i is irrelevant and the relative position of P_j is implicitly given by the no-fit function. In the definitions above, Φ, Θ are discrete sets of angles and $\mathcal{R}$ is a set of radii. For more information see Sec. 4.1.4.

After optimally placing P_j relative to P_i, the waste can be quantified by the difference between the area of the convex hull and the area of the two pieces. We will denote the waste as $d_{i,j} = \mathrm{d}(P_i, P_j)$, since we will later relate this quantity to the distance between two cities i and j in a TSP. With the definitions above, the waste is given by

$$d_{i,j} = \mathrm{d}(P_i, P_j) = \mathrm{area}[\mathrm{CH}^*(P_i, P_j)] - \mathrm{area}(P_i) - \mathrm{area}(P_j) .$$

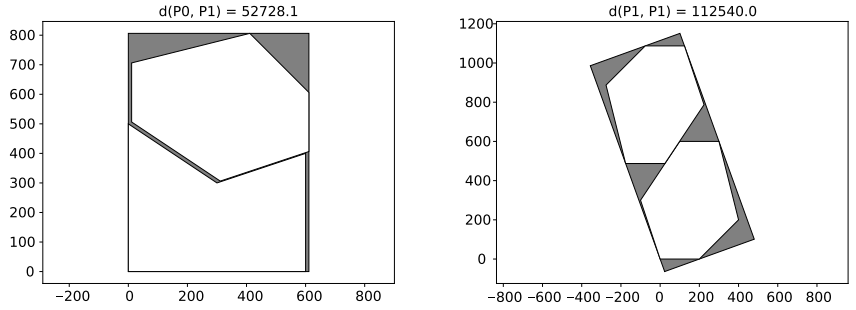
Furthermore, we define the following measure, which we will refer to as *geometrical incompatibility* between P_i and P_j.

Definition 1 (Distance and geometrical incompatibility/compatibility)

$$\mathrm{gi}(P_i, P_j) = \frac{\mathrm{d}(P_i, P_j)}{\mathrm{area}(\mathrm{CH}[P_i, P_j(r,\theta,\phi)])} \quad (3)$$

$$\mathrm{gc}(P_i, P_j) = 1 - \mathrm{gi}(P_i, P_j) \quad (4)$$

Examples of distances are given in Fig. 2. The geometrical incompatibility is always bounded by 0 and 1. An incompatibility of 0 means that the two pieces can be placed without waste.



(a) (b)

Figure 2: The distance between two pieces and wasted area (grey). The axes and distances are given in arbitrary units.

Our packing algorithm seeks to exploit the geometrical compatibility between pairs of pieces. When compatible pieces are placed together, the space between the pieces is small and this allows to minimize the wasted area in the container. The first step of the algorithm consists of computing the geometrical compatibility between all pairs of pieces, or equivalently, to compute the distance matrix, which we define as

Definition 2 (Distance matrix) The $N \times N$ square matrix D with components $D_{i,j} = d_{i,j}$, where $d_{i,j} = \mathrm{d}(P_i, P_j)$ is the distance between pieces P_i and P_j.

4.1.2 Generate groups of geometrically compatible pieces

In the second step of the Opus Incertum algorithm, we generate groups of pieces in which the pieces are pairwise geometrically compatible. We then spatially arrange together the pieces that are geometrically compatible. For this, we use the *single-linkage clustering* algorithm which is a hierarchical clustering method [41]. In the beginning, each element is in a cluster of its own. The agglomerative process consists of grouping the two clusters that contain the closest pair of elements in each step. The clusters are then sequentially combined into larger clusters, until all elements are in the same cluster. We use the geometrical incompatibility measure in Eq. (3) to cluster the set of all pieces. The single-linkage clustering algorithm then produces clusters in which each piece has low geometrical incompatibility with at least one other pieces. The resulting set of clusters forms a partition of the set of all pieces.

To this end, it is necessary to abort the agglomerative process before all the pieces belong to a single cluster. We achieve this in two ways. First, we introduce a threshold for the maximum linkage distance allowed for merging two clusters. To account for the arbitrariness of the threshold, we define several threshold values and obtain different partitions of the set of pieces. Second, we impose a maximum number of pieces per cluster, so that whenever this size is about to be exceeded, the agglomerative process terminates.

4.1.3 Ordering pieces in each cluster

Consider a cluster C_i and denote its pieces $P_{i_1}, \ldots, P_{i_n}$. The goal is to place these pieces without overlap and as compactly as possible by exploiting their geometrical compatibility. Since the compatibility of pieces is only a binary measure, for $n > 2$ we do not know in general how large the area of the minimum bounding polygon for the n cluster pieces is. Computing all possible placements that avoid overlap for n pieces is not reasonable since it has an exponential scaling in terms of the number of pieces which becomes already problematic for small values of n.

We therefore place the pieces one by one in the order of some sequence $P_{i_{\sigma(1)}}, \ldots, P_{i_{\sigma(n)}}$. The number of possible permutations σ for a set of n elements is $n!$. To limit the computation, we choose the sequence with minimal total distance:

$$\min_{\sigma} \sum_{i=1}^{n-1} \mathrm{d}(P_{i_{\sigma(i)}}, P_{i_{\sigma(i+1)}})$$

This sequence corresponds to the *shortest Hamiltonian path*, defined as follows.

Definition 3 (shortest Hamiltonian path) Consider the fully connected, undirected and weighted graph with $n = |V|$ nodes $\{1, \ldots, n\}$ and distances $d_{i,j}$ from node i to j as weights. A Hamiltonian path is a sequence $\sigma(1), \ldots, \sigma(n)$ visiting all nodes of the graph exactly once, i.e., σ is a permutation of the set $\{1, \ldots, n\}$. The shortest Hamiltonian path is the Hamiltonian path with minimum total distance

$$D(\sigma) = \sum_{i=1}^{n-1} d_{\sigma(i),\sigma(i+1)}$$

Finding the shortest Hamiltonian path of a weighted graph is known as *Traveling Salesman Problem* (TSP). When n is small enough (roughly $n \leq 10$), the TSP is an NP-hard combinatorial optimization problem can be solved exactly and fast using brute force search. Solving larger instances quickly becomes computationally challenging. Quantum computing provides methods to solve this problem heuristically. In this work, we will focus on the Quantum Approximate Optimization Algorithm [3] and the a variation thereof called Quantum Alternating Operator Ansatz [4]. The former requries the reformulation of the TSP into a quadratic unconstrained binary optimization (QUBO) problem. This is described in Appendix A.

4.1.4 Packing the pieces of each cluster

Once an order is obtained, the pieces in each cluster are packed as compactly as possible. For this we use a greedy algorithm that follows in the order of the chosen sequence $P_{i_{\sigma(1)}}, \ldots, P_{i_{\sigma(n)}}$. We start by positioning the first piece $P_{i_{\sigma(1)}}$ with the default position $(0, 0)$. For every subsequent step $k = 2, \ldots, n$, we then place the k-th piece $P_{i_{\sigma(k)}}$ by orbiting around the previous piece $P_{i_{\sigma(k-1)}}$ and searching for the position (r, θ) and orientation ϕ that minimizes the area of the bounding box of $P_{i_{\sigma(1)}}, \ldots, P_{i_{\sigma(k)}}$ while avoiding overlap with the previously positioned pieces.

The term orbiting here means that we vary θ from a set of predefined angles Θ. For each θ, the no-fit function Eq. (1) returns a radius r and an angle ϕ. If the k-th piece of the sequence can be placed relatively to the previous piece with position (r, θ, ϕ) without overlap with the previously placed pieces, then we place the piece and proceed with the next piece. In the general case, however, the position returned by the no-fit function can lead to overlapping. We then increase the value of $r \in \mathcal{R}$ while keeping the orientation ϕ fixed, until the piece is far enough to avoid overlapping. We refer to this procedure as *no-fit-function-based greedy packing*. Due to the no-fit function and the constraint of keeping the orientation fixed, the complexity of greedy packing is only $O([|\mathcal{R}| \cdot |\Theta|] \cdot [n-1])$. Once the pieces of a cluster are placed, we compute the corresponding bounding box, see Fig. 3.

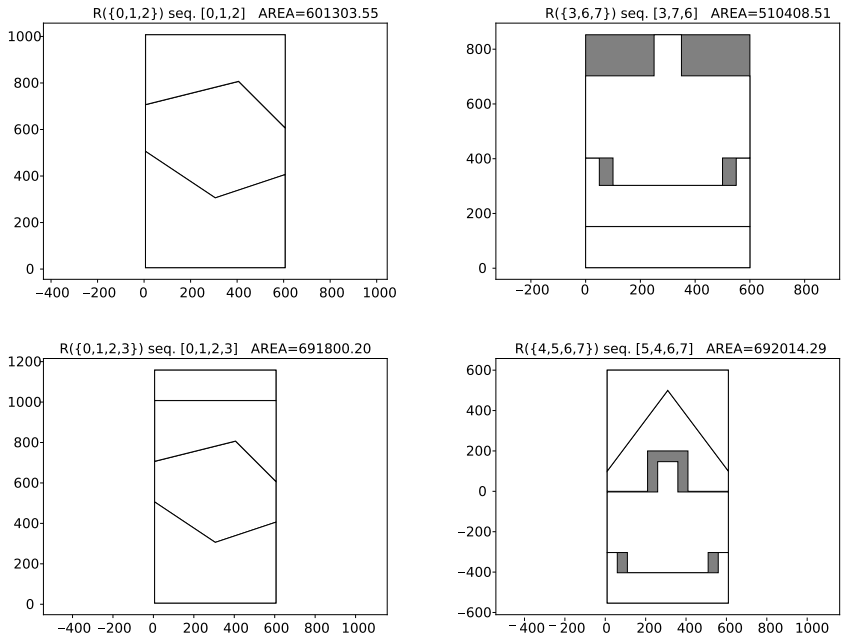


Figure 3: Packing of pieces in clusters. Axes are given in arbitrary units.

4.1.5 Partition filtering

The number of partitions obtained may be large and lead to long computation times. We reduce the number of partitions used for packing the pieces by selecting the most promising ones. This can be done by computing a penalty for each partition and keeping only the best $n_{\text{partitions}}$ ones. A simple penalty score for some partition consists of the sum of areas of the rectangles.

4.1.6 Rectangle packing

Each cluster is a group of pieces that has been placed together tightly into a bounding box of rectangular shape. In the rectangle packing step, these rectangles are allocated to the rectangular container with the help of an efficient rectangle packer. This is a well known problem [42] which formally consists of packing a set of K rectangles $\mathcal{R} = \{R_1, R_2, \ldots, R_K\}$, where each rectangle R_i has fixed length l_i and height h_i in the rectangular container C of variable length L and fixed height H. When all pieces are rectangular, the solution space for the irregular packing problem becomes finite [16]. If we denote (x_i, y_i) the coordinates of the bottom left corner of rectangle R_i, the problem can be described as follows [42]:

$$
\begin{aligned}
\text{minimize} \quad & L \\
\text{subject to} \quad & 0 \leq x_i \leq L - l_i, \quad 1 \leq i \leq K \\
& 0 \leq y_i \leq H - h_i, \quad 1 \leq i \leq K \\
& \text{At least one of the next four inequalities} \\
& \text{holds for every pair } R_i \text{ and } R_j \text{ of rectangles :} \\
& x_i + l_i \leq x_j \\
& x_j + l_j \leq x_i \\
& y_i + h_i \leq y_j \\
& y_j + h_j \leq y_i
\end{aligned}
$$

The first two constraints ensure that every rectangle is contained in the container. The remaining constraints express that rectanlges do not overlap. An illustration for the example in Fig. 1 is given in Fig. 4(a).

4.1.7 Local optimization

The initial placement is obtained from positioning the rectangles in the container and substituting the rectangles for each cluster with the placement of the cluster pieces generated by greedy packing. Each such initial placement can be improved by local optimization. The objective of the local optimization procedure is to reduce the length needed for the container. In this procedure, the position of the pieces is variable, but their orientation is kept constant. The procedure consists of iterating through all the pieces from the bottom-left most to the top right most one and for each pieces, to apply translations of

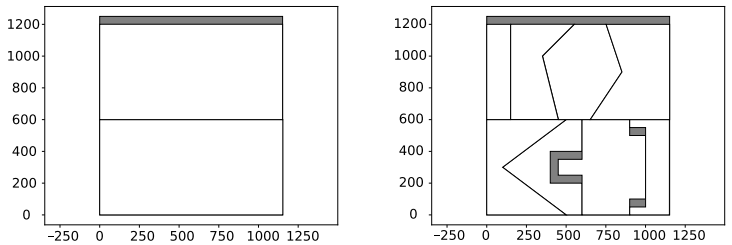


Figure 4: Results for the example shown in Fig. 1. (a) Rectangle packing for the . The rectangles packed are from left to right, bottom to top $R(\{0,1,2,3\})$, $R(\{4,5,6,7\})$. (b) Layout obtained with Opus Incertum. The container has a length of $L = 1151.89$. The percentage of waste is 6.59%. The axes are given in arbitrary units.

small amplitude (the granularity Δr of $\mathcal{R}$) leftwards and/or downwards. Concretely, we apply translations with amplitude Δr and direction given by one of the following four unit vectors:

$$u_1 = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \quad u_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 \\ -1 \end{bmatrix} \quad u_3 = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 \\ +1 \end{bmatrix} \quad u_4 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

Translation are applied as long as the translated pieces does not overlap with the other pieces and remain completely inside the container. We cycle several times through all the pieces until no piece can be translated anymore. The result can be seen in Fig. 4(b).

4.1.8 Global optimization

Local optimization reduces the gaps between neighbouring pieces, but it usually does not completely remove them. It is sometimes possible to make use of these gaps to fit the rightmost piece. By relocating the rightmost piece into one of these gaps, i.e., the piece which occupies a position the maximum value on the horizontal x-axis, we have a good chance to reduce the length needed for the container. This may not always be the case, as there may be several rightmost pieces. The relocation thus needs to be repeated until the currently selected rightmost piece cannot be relocated. To check if a piece can be relocated, we scan from left to right and bottom to top for a position and orientation of the piece to relocate and check if the piece can be inserted without exiting the boundaries of the container and without overlapping with one of the other pieces. Significant improvements of the container length may be obtained as a result of global optimization.

Algorithm 1 Opus Incertum

Inputs: set of N pieces $\mathcal{P}$, fixed container height H, maximum cluster size $n_{\max}$, number of partitions $n_{\text{partitions}}$
Outputs: optimal placement of the pieces and container length
compute the no-fit function NFF_{P_i,P_j} for every pair of pieces $P_i, P_j \in \mathcal{P}$
compute the distance and geometrical incompatibility matrices D and GI
define the set of distance thresholds $\mathcal{T}(GI) \leftarrow \text{values}(GI)$
compute a set of partitions of $\mathcal{P}$ by single-linkage clustering using the distance thresholds of $\mathcal{T}(GI)$ and maximum cluster sizes ranging from 1 to N pieces per cluster
initialize container length $L^* \leftarrow \infty$
for each partition $\mathcal{C}$ **do**
    for for each cluster $C \in \mathcal{C}$ **do**
        solve the TSP for the distance sub-matrix $D(C)$
        let s be the shortest Hamiltonian path obtained
        pack greedily the pieces of C in the order of s
        pack greedily the pieces of C in the reverse order of s
        store the placement whose bounding box has the smallest area
    end for
end for
keep from $\mathcal{C}$ the $n_{\text{partitions}}$ partitions with lowest penalty
for each partition $\mathcal{C}$ **do**
    for bins as relaxation of the rectangular container $H \times L^*$ **do**
        try to pack the selected bounding boxes in the bin
        if a solution is found to the rectangle packing problem **then**
            optimize the placement by local optimization
            let X be the optimized placement
            let l and h be the length and height needed by X
            if $h \leq H$ and $l < L^*$ **then**
                update optimal placement $X^* \leftarrow X$ and length $L^* \leftarrow l$
            end if
        end if
    end for
end for
optimize the placement X^* and length L^* by shifting the right-most piece to the bottom-left-most free area and repeat this step until no further improvement is possible
return optimal placement X^* and length L^*

4.2 The algorithm

With the previous definitions, we can now state the formal algorithm which is shown in Algorithm 1. Given the set of N pieces $\mathcal{P} = \{P_0, \ldots, P_{N-1}\}$, we first compute and store the values of the no-fit functions Eq. (1) for every

pair of pieces P_i and P_j. Note that the no-fit function does not need to be re-calculated if the pair of pieces is identical to a pair of pieces that has already been considered. This happens when pieces have identical shapes. Besides, after $\mathrm{NFF}P_i, P_j$ has been computed, we can derive directly values of the no-fit function NFF_{P_j,P_i} via

$$\mathrm{NFF}_{P_j,P_i}(\theta) = (r, \phi) \text{ where } \mathrm{NFF}_{P_i,P_j}(\theta + 180 - \phi) = (r, -\phi)\,. \tag{5}$$

Once all no-fit functions are determined, the $N \times N$ distance matrix D can be quickly determined. Each distance $d_{i,j}$ is calculated as follows:

$$\begin{aligned}
\theta^* &\leftarrow \arg\min_{\theta \in \Theta} \{\mathrm{area}(\mathrm{CH}(P_i, P_j(r, \theta, \phi))) \,|\, (r, \phi) = \mathrm{NFF}_{P_i,P_j}(\theta)\} \\
(r^*, \phi^*) &\leftarrow \mathrm{NFF}_{P_i,P_j}(\theta^*) \\
d_{i,j} &\leftarrow \mathrm{area}(\mathrm{CH}(P_i, P_j(r^*, \theta^*, \phi^*))) - \mathrm{area}(P_i) - \mathrm{area}(P_j(r^*, \theta^*, \phi^*))
\end{aligned}$$

Let $\mathcal{T}(GI)$ denote the set of threshold geometrical incompatibilities used for the single-linkage clustering algorithm. This set may be any subset of the set of coefficients of the distance matrix GI:

$$\mathcal{T}(GI) \subseteq \mathrm{values}(GI) = \{\mathrm{gi}_{i,j} | i \in \{0, \ldots, N-1\}, j \in \{0, \ldots, N-1\}\}\,.$$

In our algorithm definition, we choose $\mathcal{T}(GI) = \mathrm{values}(GI)$.

For every distance threshold $d_{\max}$ in $\mathcal{T}(GI)$ and maximum cluster size $n_{\max}$ from 1 to N (or some lower upper limit), partition the set of pieces using the single-linkage clustering algorithm and distance matrix GI. We obtain in this way a set of partitions of $\mathcal{P}$. We then initialize the variable L^* which stores the container length for the best placement found. The initial value given is infinity.

For each partition $\mathcal{C}$, we consider every cluster C in $\mathcal{C}$ and the distance of the pieces in C, noted $D(C)$, obtained directly as a sub-matrix of D. We then solve the TSP for $D(C)$ and get the shortest Hamiltonian path $s = i_1, \ldots, i_n$ visiting once exactly all pieces of the cluster. Since the distance matrix D and $D(C)$ are symmetric, the reverse path $s' = i_n, \ldots, i_1$ has the same length and is another equally optimal solution of the TSP. We thus greedily pack the pieces in each cluster C either in the order of the sequence s or in the order of s'. The placements and resulting bounding boxes differ in the general case. We select and store the placement that results in the bounding box with the smallest area of the two in memory.

For each partition $\mathcal{C}$, we try to pack the bounding boxes inside the container using a rectangle packer. More precisely, the packer gets the task of finding a feasible placement of all bounding boxes within a rectangular container called *bin* whose dimensions coincide with the current best container found $H \times L^*$, where L^* is the minimum length found so far. If the rectangle packer is

Table 1: Performance of the Quantum Approximate Optimization Algorithm. The optimality is defined in Eq. (6).

Optimizer	p	Optimality	Execution time
COBYLA	1	78.2%	2.5s
	2	76.6%	5.7s
	3	79.3%	9.0s
	4	77.1%	12.9s
	5	85.9%	17.3s
BFGS	1	68.7%	6.4s
	2	71.0%	16.8s
	3	73.6%	26.5s
	4	70.3%	44.5s
	5	73.1%	50.8s
L-BFGS-B	1	68.7%	6.3s
	2	70.4%	17.5s
	3	73.6%	29.8s
	4	67.9%	38.2s
	5	73.1%	53.3s
SLSQP	1	70.9%	14.1s
	2	83.1%	46.5s
	3	72.1%	87.5s
	4	82.8%	142.2s
	5	84.3%	202.6s
SPSA	1	85.4%	23.4s
	2	81.6%	28.5s
	3	79.1%	36.5s
	4	74.0%	42.9s
	5	77.7%	51.4s

successful, the bounding boxes are replaced by the pieces of the cluster they pack, resulting in a feasible placement of the set of all pieces $\mathcal{P}$ in a container of dimension $H \times L^*$. The placement can be further optimized using local optimization, resulting in a container length $L \leq L^*$. If L is an improvement over L^*, then L^* is updated with L and we store the placement as the optimal placement X^*. If the length is not reduced, then no update is necessary. If the rectangle packer does not find a solution for this dimension, we relax the dimensions of the bin until a solution is found, by enlarging the length L^* and/or the height H. Any solution found is optimized by local optimization and feasible solutions ($h \leq H$) are then compared to the optimal length L^*. Whenever the optimal length is improved, we update the variables L^* and X^*.

Let X^* be the best placement found after consideration of all the partitions. The length can be sometimes further optimized by global optimization, i.e., by iteratively relocating the right-most pieces to the left-most free gap in the container.

5 Experimental Results

To evaluate the performance of our approach, we create two small problem instances which belong to the class of packing problems where we expect the

Table 2: Performance of the Quantum Alternating Operator Ansatz. The optimality is defined in Eq. (6).

Optimizer	p	Optimality	Execution time
COBYLA	1	66.9%	6.8s
	2	70.9%	17.7s
	3	72.6%	33.0s
	4	84.6%	54.5s
	5	74.5%	79.0s
BFGS	1	61.6%	3.0s
	2	54.1%	6.4s
	3	61.3%	10.2s
	4	63.1%	14.4s
	5	63.5%	19.1s
L-BFGS-B	1	62.9%	3.2s
	2	61.7%	6.6s
	3	61.3%	10.4s
	4	60.8%	14.4s
	5	62.9%	19.1s
SLSQP	1	64.2%	3.0s
	2	60.4%	6.4s
	3	61.8%	10.2s
	4	60.9%	14.4s
	5	66.1%	19.1s
SPSA	1	67.2%	38.2s
	2	76.1%	66.6s
	3	80.0%	96.4s
	4	72.9%	126.3s
	5	72.9%	157.3s

Opus Incertum Algorithm to yield optimal results. The sets contain six and seven pieces, respectively. The pieces can be arranged in pairs or triples and form almost perfect rectangles when aligned. These can then be optimally packed inside the container by the rectangle packer. For reproducible, we provide the definition of the sets of pieces in Appendix B. We also evaluate our approach for five problem instances commonly used for benchmarking packing algorithms. The instances are referred to as SHAPES1, SHAPES2, SHIRTS, TROUSERS and SWIM and their definition can be found in [43]. Note however that these sets, which contain between 28 and 99 pieces, do not fulfil our requirements very well.

For each problem instance, we report the performance obtained with `SVGnest`[2] compared to the results of Opus Incertum. Our Python implementation, which is based on the Python modules Shapely[3] and RectPack[4].

To solve the TSPs, we implement one classical brute force search and two quantum algorithms: the Quantum Approximate Optimization Algorithm and the Quantum Alternating Operator Ansatz. The hyper-parameters for the quantum algorithms are the number of repetitions and the classical optimizer,

[2] https://github.com/Jack000/SVGnest
[3] https://pypi.org/project/shapely/
[4] https://pypi.org/project/rectpack/

Algorithm 2 Post-processing of results from quantum circuits.

1: **Inputs**: a binary string $x_{0,0} \dots x_{i,p} \dots x_{n-1,n-1}$ of length n^2
2: **Outputs**: a hamiltonian path for the nodes $\{0, \dots, n-1\}$
3: for each node i, compute $\sum_{p=0}^{n-1} x_{i,p}$ and if the sum is strictly greater than 1, select at random a step p^* which $x_{i,p^*} = 1$ and set all the $x_{i,p}$ for $p \neq p^*$ to 0
4: for each step p, compute $\sum_{i=0}^{n-1} x_{i,p}$ and if the sum is strictly greater than 1, select at random a node i^* which $x_{i,p^*} = 1$ and set all the $x_{i,p}$ for $i \neq i^*$ to 0
5: for each node i, compute $\sum_{p=0}^{n-1} x_{i,p}$ and if the sum is null, select at random a step p^* for which $\sum_j x_{j,p^*} = 0$ and set x_{i,p^*} to 1
6: for each step p, compute $\sum_{i=0}^{n-1} x_{i,p}$ and if the sum is null, select at random a node i^* for which $\sum_{p'} x_{i^*,p'} = 0$ and set $x_{i^*,p}$ to 1
7: let $\sigma = i_0, \dots, i_{n-1}$ be the path defined for any step p by $i_p = i$ if $x_{i,p} = 1$
8: **return** path σ

which are set as follows. We randomly generate 30 symmetric distance matrices of dimension 4×4, using coefficients uniformly distributed between 0 and 1. We run each algorithm on these problems using repetitions ranging from 1 to 5 and optimize the circuits using one of the following classical optimizers: COBYLA, BFGS, L-BFGS-B, SLSQP and SPSA. The quantum algorithms are then executed using Qiskit [44] on a quantum simulator without noise (qasm simulator). Each circuit is executed 1,000 times and the result(s) with highest count is/are post-processed. Post-processing is necessary to ensure the validity of each solution. We call any Hamiltonian path, i.e., a sequence visiting each node exactly once a *valid solution*. The post-processing is shown in Algorithm 2. In case multiple solutions are obtained, we compute the corresponding total distances and keep the Hamiltonian path σ with smallest total distance $D(\sigma)$ as a unique solution. The performance for a path is measured by the optimality of its total distance, which we define as

$$1 - \frac{D(\sigma) - D_{\min}}{D_{\max} - D_{\min}} \tag{6}$$

The optimality is averaged over the different TSPs.

The results are given in the Tables 1 and 2. It can be observed that in our experiments, the vanilla version of the Quantum Approximate Optmiziation Algorithm performs overall better while simulataneously requiring less execution time. As expected, the quality of the solution generally increases with an increasing number of iterations, although we observe some exceptions such as the SPSA variant of the Quantum Approximate Optmiziation Algorithm. For the following application of the Opus Incertum Algorithm, we proceed with a choice of the COBYLA optimizer and $p = 5$ repetitions for the Quantum Approximate Optimization Algorithm, and the COBYLA optimizer and $p = 4$ repetitions for the Quantum Alternating Operator Ansatz.

PROBLEM INSTANCE vs. METHOD	PUZZLE1 6 shapes, 6 pieces, H = 750	PUZZLE2 7 shapes, 7 pieces, H = 420	PUZZLE3 12 shapes, 12 pieces, H = 1200
Opus Incertum (Brute Force Search)	W=14.77%, T=659.02s	W=6.55%, T=927.17s	W=14.47%, T=8784.24s
Opus Incertum (Quantum Approximate Optimization Algorithm, simulator)	W=14.77%, T=645.50s	W=6.55%, T=830.17s	W=18.76%, T=8557.82s
Opus Incertum (Quantum Approximate Optimization Algorithm, quantum computer)	W=14.77%, T=697.29s	W=6.70%, T=899.36s	W=20.10%, T=9155.27s
Opus Incertum (Quantum Alternating Operator Ansatz, simulator)	W=14.77%, T=656.60s	W=6.55%, T=902.98s	W=17.36%, 9348.29s
Opus Incertum (Quantum Alternating Operator Ansatz, quantum computer)	W=14.77%, T=705.43s	W=6.86%, T=1069.09s	W=21.35%, T=22877.37s

Figure 5: Performance results for PUZZLE1, PUZZLE2 and PUZZLE3. Results from real quantum computer are obtained from IBM Ehningen.

PROBLEM INSTANCE vs. METHOD	SHAPES1 4 shapes, 43 pieces, H = 400	SHAPES2 7 shapes, 28 pieces, H = 150	SHIRTS 8 shapes, 99 pieces, H = 400	TROUSERS 17 shapes, 64 pieces, H = 790	SWIM 10 shapes, 48 pieces, H = 5752
Opus Incertum (Brute Force Search)	W=42.97%, T=2137.24s	W=27.12%, T=4981.99s	W=24.20%, T=13279.78s	W=16.77%, T=50639.89s	W=44.42%, T=69377.16s
Opus Incertum (Quantum Approximate Optimization Algorithm, simulator)	W=42.97%, T=3066.53s	W=27.12%, T=5370.31s	W=22.33%, T=15495.49s	W=16.77%, T=39215.46s	W=49.78%, T=32074.27s
Opus Incertum (Quantum Approximate Optimization Algorithm, quantum computer)	W=42.97%, T=3424.93s	W=27.12%, T=13373.23s	W=26.44%, T=83871.25s	W=16.77%, T=47787.09s	W=41.31%, T=34753.92s
Opus Incertum (Quantum Alternating Operator Ansatz, simulator)	W=37.22%, T=3356.84s	W=27.12%, T=6611.17s	W=24.20%, T=8294.80s	W=16.77%, T=39171.81s	W=40.26%, T=35808.40s
Opus Incertum (Quantum Alternating Operator Ansatz, quantum computer)	W=37.22%, T=4462.86s	W=27.12%, T=11551.44s	W=22.33%, T=8930.77s	W=16.77%, T=47459.27s	W=41.31%, T=41310.82s

Figure 6: Performance results for SHAPES1, SHAPES2, SHIRTS, TROUSERS and SWIM. Results from real quantum computer are obtained from IBM Ehningen.

Once the hyper-parameters are set, we build variational circuits with p repetitions and train the parameters of the QAOA algorithms with the chosen optimizer using a noiseless quantum simulator up to $n^2 = 16$ qubits. We compare these results to computations on the IBM quantum computer in Ehningen [45]. As for the hyper-parameter optimization setting, we use 1,000 shots per circuit, select the results with the highest count and post-process the bit-strings to obtain valid solutions.

The performance results for PUZZLE1, PUZZLE2 and PUZZLE3 are shown in Fig. 5 and for SHAPES1, SHAPES2, SHIRTS, TROUSERS and SWIM in Fig. 6. The results obtained with the Opus Incertum Algorithm depend on several factors, such as the number of rotation angles in Θ and orientation angles in Φ, spatial granularity Δr, the spatial granularity of the grid used for relocating pieces, the maximum cluster size $n_{\max}$ allowed and the algorithm and hyper-parameter combination used to solve the TSP instances and the number of partitions. In our experiments, we allow rotations in multiples

of 5 degrees which amounts to 72 rotations in total. For PUZZLE1, PUZZLE2 and PUZZLE3, we restrict the rotations to multiples of 90 degrees. For SHAPES1, SHAPES2, SHIRTS, SWIM and TROUSERS, we consider multiple of 45 degrees, amounting to 8 distinct rotations. The spatial granularity is set to $\Delta r = 5$. The grid is a uniform grid from the container of dimension 100×100. In our experimental setting, we limit the maximum cluster size to 4 pieces as a higher number would then exceed the maximum number of 16 qubits that a quantum simulator can handle. We set the number of partitions to 20 for PUZZLE 1, 40 for PUZZLE2 and 50 for PUZZLE3. These parameters play a critical role in determining the algorithm's ability to find an optimal layout, with the choice of rotation angles directly affecting the potential for piece alignment and the granularity impacting the resolution of placement. The decision to use different parameters for different problem sets illustrates a tailored approach to optimization, trading off between computational demand and the quality of the solution.

Note, however, that better results than those displayed can be achieved by increasing for instance the number of rotations, increasing the maximum cluster size (up to 10 pieces when solving the TSPs by brute force search) and the number of partitions, at the expense of longer computation times. The final placements for each problem instance can be found in Appendix C.

It can be observed that solving the TSP with one of the quantum algorithms is often comparable to brute force solutions with no general tendency of a significant performance supremacy in either direction. Notably, the results obtained by solving the TSP on a real quantum computer are competitive with those obtained from a noiseless simulation, indicating a certain robustness in the Opus Incertum Algorithm. For example, in PUZZLE2, the waste percentage achieved by the Quantum Approximate Optimization Algorithm on a quantum computer ($W = 6.70\%$, $T = 899.36s$) closely mirrors that of the brute force search ($W = 6.55\%$, $T = 927.17s$), illustrating the quantum method's capacity to match classical performance levels. This parity is visible in other problem instances as well and suggests that while quantum computing offers a novel approach to problem-solving, its current stage of development shows comparable efficiency to classical methods for this specific application.

6 Conclusion

We have decomposed the NP-hard strip packing problem into two core problems, the TSP and the regular packing problem. In this work, we have solved the TSP classically and with quantum computing, using two different variants of the QAOA algorithm. Interestingly, the regular packing problem can be formulated in QUBO form and also be solved using quantum computing, as demonstrated recently in [40]. The practicality of the proposed algorithm as a quantum-classical hybrid or a purely classical, quantum-inspired method, is contingent on the advancements in quantum computing and the chosen approach for solving the underlying TSP. It is agnostic to the specific method

employed for the TSP, meaning the performance significantly varies with the choice between novel quantum algorithms or classical heuristics. Furthermore, the selection of hyperparameters, particularly the granularity of angles and grids, introduces a crucial trade-off between the quality and performance of the solution. In comparing our algorithm's effectiveness, it is important to benchmark not only against brute force solutions but also against classical heuristics, which can offer a more efficient yet effective alternative. Another promising avenue for research is exploring scalability, particularly with regards to significantly increasing the number of pieces within each cluster. This is particularly interesting because, fundamentally, the Opus Incertum algorithm performs local optimization that becomes more global as the number of pieces within each cluster increases relative to the total number of clusters.

The proposed approach may be improved as follows. Other measures of geometrical compatibility may be developed and lead to more dense clusters of pieces. The algorithm complexity may be improved. Indeed, the number of partitions of the set of pieces used for generating candidate placements may be reduced, by considering only the most promising partitions. For instance, after computing all partitions, one could assign to each partition a loss simply defined as the total area of the boxes bounding the clustered pieces and choose the partitions with the smallest loss.

7 Acknowledgements

This work was part of the SEQUOIA project funded by the Minister for Economic Affairs, Labour and Tourism Baden-Württemberg. We also acknowledge use of the IBM Quantum Experience for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Quantum team.

References

[1] Gomes, A.M., Oliveira, J.F.: Solving irregular strip packing problems by hybridising simulated annealing and linear programming. European Journal of Operational Research **171**(3), 811–829 (2006)

[2] Abbas, A., Ambainis, A., Augustino, B., Bärtschi, A., Buhrman, H., Coffrin, C., Cortiana, G., Dunjko, V., Egger, D.J., Elmegreen, B.G., et al.: Quantum optimization: Potential, challenges, and the path forward. arXiv preprint arXiv:2312.02279 (2023)

[3] Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: Quantum computation by adiabatic evolution. arXiv preprint quant-ph/0001106 (2000)

[4] Hadfield, S., Wang, Z., O'gorman, B., Rieffel, E.G., Venturelli, D., Biswas, R.: From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. Algorithms **12**(2), 34 (2019)

[5] Leao, A.A., Toledo, F.M., Oliveira, J.F., Carravilla, M.A., Alvarez-Valdés, R.: Irregular packing problems: A review of mathematical models. European Journal of Operational Research **282**(3), 803–822 (2020)

[6] Toledo, F.M., Carravilla, M.A., Ribeiro, C., Oliveira, J.F., Gomes, A.M.: The dotted-board model: a new mip model for nesting irregular shapes. International Journal of Production Economics **145**(2), 478–487 (2013)

[7] Rodrigues, M.O., Toledo, F.M.: A clique covering mip model for the irregular strip packing problem. Computers & Operations Research **87**, 221–234 (2017)

[8] Scheithauer, G., Terno, J.: Modeling of packing problems. Optimization **28**(1), 63–84 (1993)

[9] Daniels, K., Li, Z., Milenkovic, V.: Multiple containment methods (1994)

[10] Dean, H.T.: Minimizing waste in the 2-dimensional cutting stock problem (2002)

[11] Fischetti, M., Luzzi, I.: Mixed-integer programming models for nesting problems. Journal of Heuristics **15**(3), 201–226 (2009)

[12] Alvarez-Valdes, R., Martinez, A., Tamarit, J.: A branch & bound algorithm for cutting and packing irregularly shaped pieces. International Journal of Production Economics **145**(2), 463–477 (2013)

[13] Cherri, L.H., Mundim, L.R., Andretta, M., Toledo, F.M., Oliveira, J.F., Carravilla, M.A.: Robust mixed-integer linear programming models for the irregular strip packing problem. European Journal of Operational Research **253**(3), 570–583 (2016)

[14] Leao, A.A., Toledo, F.M., Oliveira, J.F., Carravilla, M.A.: A semi-continuous mip model for the irregular strip packing problem. International Journal of Production Research **54**(3), 712–721 (2016)

[15] Martinez-Sykora, A., Alvarez-Valdés, R., Bennell, J.A., Ruiz, R., Tamarit, J.M.: Matheuristics for the irregular bin packing problem with free rotations. European Journal of Operational Research **258**(2), 440–455 (2017)

[16] Bennell, J.A., Oliveira, J.F.: The geometry of nesting problems: A tutorial. European journal of operational research **184**(2), 397–415 (2008)

[17] Oliveira, J.F.C., Ferreira, J.A.S.: In: Vidal, R.V.V. (ed.) Algorithms for Nesting Problems, pp. 255–273. Springer, Berlin, Heidelberg (1993)

[18] Segenreich, S.A., Braga, L.M.P.F.: Optimal nesting of general plane

figures: a monte carlo heuristical approach. Computers & Graphics **10**(3), 229–237 (1986)

[19] Babu, A.R., Babu, N.R.: A generic approach for nesting of 2-d parts in 2-d sheets using genetic and heuristic algorithms. Computer-Aided Design **33**(12), 879–891 (2001)

[20] Preparata, F.P., Shamos, M.I.: Computational geometry: an introduction. Springer (2012)

[21] Konopasek, M.: Mathematical treatments of some apparel marking and cutting problems. US Department of Commerce Report **99**(26), 90857–10 (1981)

[22] Mahadevan, A.: Optimization in computer-aided pattern packing (marking, envelopes). PhD thesis (1984). AAI8507009

[23] Ferreira, J., Alves, J., Albuquerque, C., Oliveira, J., Ferreira, J., Matos, J.: A flexible custom computing machine for nesting problems. Proceedings of XIII DCIS, Madrid, Spain, 348–354 (1998)

[24] Milenkovic, V., Daniels, K., Li, Z.: Automatic marker making. In: Proceedings of the Third Canadian Conference on Computational Geometry, pp. 243–246 (1991). Simon Fraser University

[25] Ghosh, P.K.: An algebra of polygons through the notion of negative shapes. CVGIP: Image Understanding **54**(1), 119–144 (1991)

[26] Bennell, J.A., Dowsland, K.A.: Hybridising tabu search with optimisation techniques for irregular stock cutting. Management Science **47**(8), 1160–1172 (2001)

[27] Stoyan, Y., Scheithauer, G., Gil, N., Romanova, T.: Phi-functions for complex 2d-objects. Quarterly Journal of the Belgian, French and Italian Operations Research Societies **2**(1), 69–84 (2004)

[28] Chernov, N., Stoyan, Y., Romanova, T.: Mathematical model and efficient algorithms for object packing problem. Computational Geometry **43**(5), 535–553 (2010)

[29] Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R., Graham, R.L.: Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM Journal on computing **3**(4), 299–325 (1974)

[30] Jakobs, S.: On genetic algorithms for the packing of polygons. European journal of operational research **88**(1), 165–181 (1996)

[31] Sato, A.K., Martins, T.C., Tsuzuki, M.S.G.: An algorithm for the strip

packing problem using collision free region and exact fitting placement. Computer-Aided Design **44**(8), 766–777 (2012)

[32] Sato, A.K., Martins, T.C., Gomes, A.M., Tsuzuki, M.S.G.: Raster penetration map applied to the irregular packing problem. European Journal of Operational Research **279**(2), 657–671 (2019)

[33] Elkeran, A.: A new approach for sheet nesting problem using guided cuckoo search and pairwise clustering. European Journal of Operational Research **231**(3), 757–769 (2013)

[34] Leung, S.C., Lin, Y., Zhang, D.: Extended local search algorithm based on nonlinear programming for two-dimensional irregular strip packing problem. Computers & Operations Research **39**(3), 678–686 (2012)

[35] Imamichi, T., Yagiura, M., Nagamochi, H.: An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. Discrete Optimization **6**(4), 345–361 (2009)

[36] Egeblad, J., Nielsen, B.K., Odgaard, A.: Fast neighborhood search for two- and three-dimensional nesting problems. European Journal of Operational Research **183**(3), 1249–1266 (2007)

[37] Layeb, A., Boussalia, S.R.: A novel quantum inspired cuckoo search algorithm for bin packing problem. International Journal of Information Technology and Computer Science **4**(5), 58–67 (2012)

[38] de Andoin, M.G., Osaba, E., Oregi, I., Villar-Rodriguez, E., Sanz, M.: Hybrid quantum-classical heuristic for the bin packing problem. arXiv preprint arXiv:2204.05637 (2022)

[39] Garcia-de-Andoin, M., Oregi, I., Villar-Rodriguez, E., Osaba, E., Sanz, M.: Comparative benchmark of a quantum algorithm for the bin packing problem. arXiv preprint arXiv:2207.07460 (2022)

[40] Terada, K., Oku, D., Kanamaru, S., Tanaka, S., Hayashi, M., Yamaoka, M., Yanagisawa, M., Togawa, N.: An ising model mapping to solve rectangle packing problem. In: 2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), pp. 1–4 (2018). IEEE

[41] Everitt, B.S., Landau, S., Leese, M., Stahl, D.: Cluster Analysis, pp. 73–75. John Wiley & Sons, Ltd, Hoboken (2011)

[42] Ibaraki, T., Imahori, S., Yagiura, M.: In: Blum, C., Aguilera, M.J.B., Roli, A., Sampels, M. (eds.) Hybrid Metaheuristics for Packing Problems, pp. 185–219. Springer, Berlin, Heidelberg (2008)

[43] Oliveira, J.F., Gomes, A.M., Ferreira, J.S.: Topos–a new constructive algorithm for nesting problems. OR-Spektrum **22**(2), 263–284 (2000)

[44] Qiskit Community: Qiskit: An Open-Source Framework for Quantum Computing (2017). https://doi.org/10.5281/zenodo.2562110. https://github.com/Qiskit/qiskit

[45] IBM Quantum https://quantum-computing.ibm.com (2023)

[46] Lucas, A.: Ising formulations of many np problems. Frontiers in physics **2**, 5 (2014)

Appendix A Formulating the TSP as a QUBO

In this section, we describe here how the TSP is encoded as a QUBO [46]. A path $\sigma(1), \ldots, \sigma(n)$ through the nodes $1, \ldots, n$ of the graph is encoded by a set x of n^2 binary variables $x_{i,p} \in \{0, 1\}$, where i and p are integers that range from 1 to n. Let $x_{i,p} \in \{0, 1\}$ be 1 if the path goes through node i at step p. The path visits n nodes if and only if $\forall p \in \{1, \ldots, n\} : \sum_{i=1}^{n} x_{i,p} = 1$. The path visits each node once if and only if $\forall i \in \{1, \ldots, n\} : \sum_{p=1}^{n} x_{i,p} = 1$. The total distance to be minimized is $D(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{i,j} \sum_{p=1}^{n-1} x_{i,p} x_{j,p+1}$. The TSP is then equivalent to minimizing

$$C(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{i,j} \sum_{p=1}^{n-1} x_{i,p} x_{j,p+1} + A \sum_{p=1}^{n} \left(1 - \sum_{i=1}^{n} x_{i,p}\right)^2 + A \sum_{i=1}^{n} \left(1 - \sum_{p=1}^{n} x_{i,p}\right)^2$$

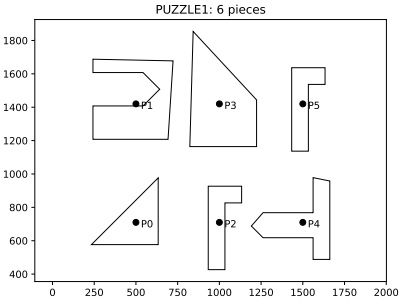
as long as the penalty A is large enough ($A > \max\{|d_{i,j}|\}$).

Appendix B Definition of puzzles

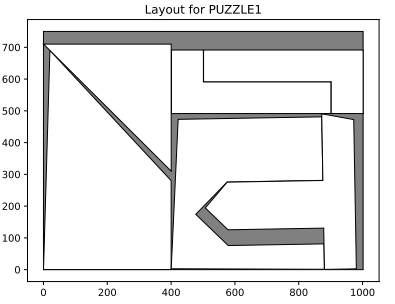
- PUZZLE1 (6 pieces):
  - P_0 = [(0, 0), (400, 0), (400, 400)]
  - P_1 = [(0, 0), (450, 0), (480, 470), (0, 480), (0, 400), (300, 400), (400, 300), (300, 200), (0, 200)]
  - P_2 = [(0, 0), (100, 0), (100, 400), (200, 400), (200, 500), (0, 500)]
  - P_3 = [(0, 0), (400, 0), (400, 280), (20, 690)]
  - P_4 = [(0, 0), (100, 0), (100, 470), (0, 490), (0, 280), (-300, 280), (-370, 200), (-300, 130), (0, 130)]
  - P_5 = [(0, 0), (100, 0), (100, 400), (200, 400), (200, 500), (0, 500)]
- PUZZLE2 (7 pieces):
  - P_0 = [(0, 0), (200, 0), (200, 300), (-100, 300), (-100, 100), (0, 100)]
  - P_1 = [(0, 0), (200, 0), (190, 150), (100, 100), (0, 150)]
  - P_2 = [(0, 0), (300, 0), (300, 100), (200, 100)]
  - P_3 = [(0, 0), (300, 0), (300, 190), (200, 190), (200, 100), (0, 100)]
  - P_4 = [(0, 0), (150, 0), (200, 100), (150, 200), (0, 200), (-50, 150), (0, 100), (-50, 50)]
  - P_5 = [(0, 0), (200, 0), (200, 90), (150, 40), (100, 90), (50, 40), (0, 90)]
  - P_6 = [(0, 0), (300, 0), (300, 100), (200, 100)]
- PUZZLE3 (12 pieces):
  - P_0 = [(0, 0), (400, 0), (400, 120), (480, 120), (480, 480), (280, 480), (280, 720), (480, 720), (480, 780), (0, 780)]
  - P_1 = [(0, 0), (700, 0), (700, 250), (600, 150), (500, 250), (300, 250), (200, 350), (100, 250), (0, 250)]
  - P_2 = [(0, 0), (680, 0), (700, 380), (500, 180), (300, 380), (0, 170)]

- P_3 = [(0, 0), (270, 0), (170, 100), (270, 200), (170, 300), (170, 400), (0, 400), (0, 270), (-70, 270), (-70, 120), (0, 120)]
- P_4 = [(0, 0), (300, 0), (300, 500), (200, 370), (100, 650), (0, 470)]
- P_5 = [(0, 0), (280, 0), (180, 320), (280, 320), (280, 480), (180, 480), (280, 780), (0, 780)]
- P_6 = [(0, 0), (300, 0), (300, 580), (200, 280), (100, 700), (0, 500)]
- P_7 = [(0, 0), (100, 0), (100, 100), (0, 200), (100, 300), (100, 500), (200, 600), (100, 700), (-200, 700), (0, 400), (-200, 200)]
- P_8 = [(0, 0), (100, 0), (100, 80), (300, 80), (300, 0), (400, 0), (400, 200), (0, 200)]
- P_9 = [(0, 0), (400, 0), (500, 300), (400, 300), (400, 500), (500, 500), (400, 800), (100, 800), (100, 700), (-100, 700), (-100, 500), (100, 500), (100, 100), (0, 100)]
- P_{10} = [(0, 0), (100, 0), (100, 100), (200, 100), (200, 0), (300, 0), (300, 100), (400, 100), (400, 300), (300, 300), (200, 200), (100, 300), (0, 200)]
- P_{11} = [(0, 0), (180, 0), (180, 80), (70, 80), (70, 220), (180, 220), (180, 280), (80, 280), (80, 400), (0, 400)]

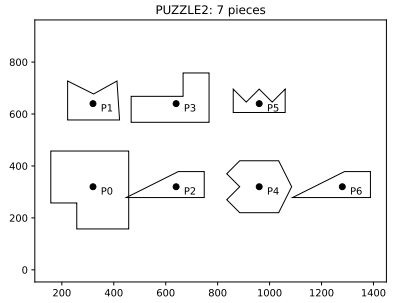
Appendix C Final placements



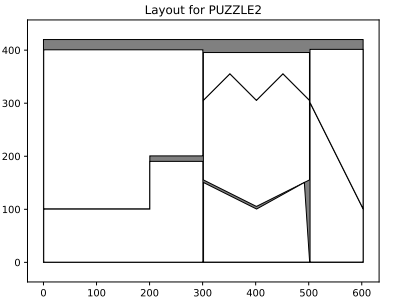
(a) Set of 6 pieces for PUZZLE1



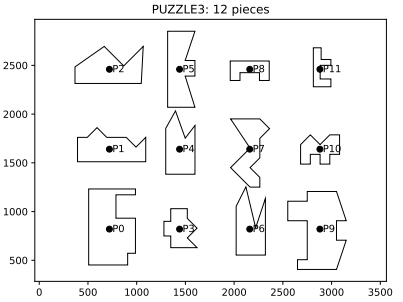
(b) Opus Incertum (Brute Force Search)



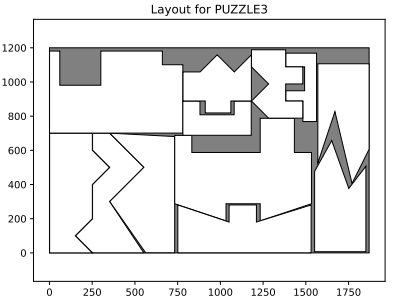
(c) Set of 7 pieces for PUZZLE2



(d) Opus Incertum (Brute Force Search)

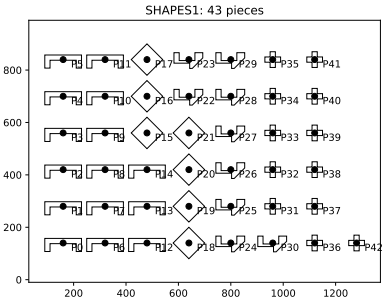


(e) Set of 12 pieces for PUZZLE3

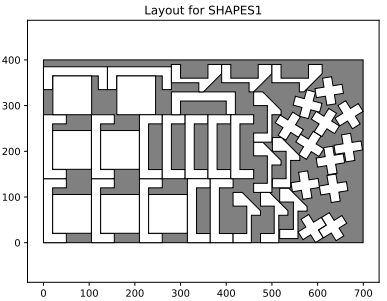


(f) Opus Incertum (Brute Force Search)

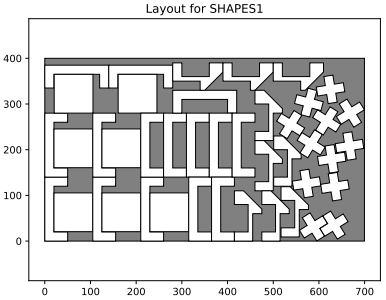
Figure C1: Results obtained for PUZZLE1, PUZZLE2 and PUZZLE3 using Opus Incertum and Brute Force Search.



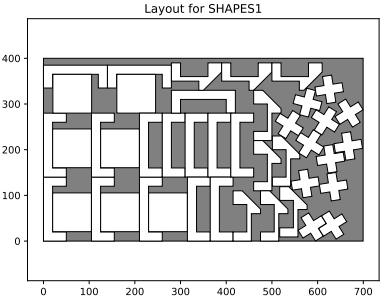
(a) Set of 43 pieces for SHAPES1



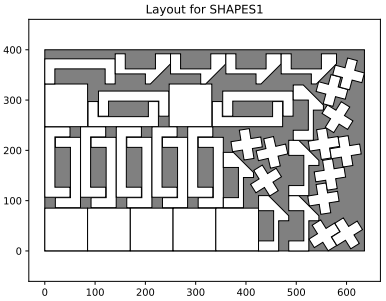
(b) Opus Incertum (Brute Force Search)



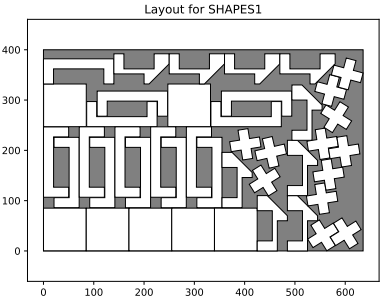
(c) Opus Incertum (Quantum Approximate Optimization Algorithm, simulator)



(d) Opus Incertum (Quantum Approximate Optimization Algorithm, quantum computer)

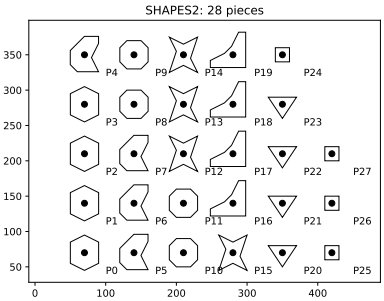


(e) Opus Incertum (Quantum Alternating Operator Ansatz, simulator)

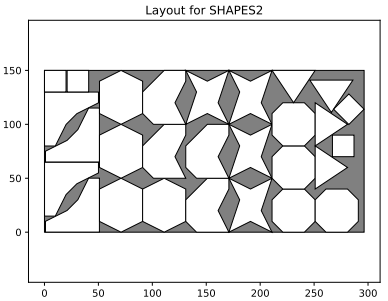


(f) Opus Incertum (Quantum Alternating Operator Ansatz, quantum computer)

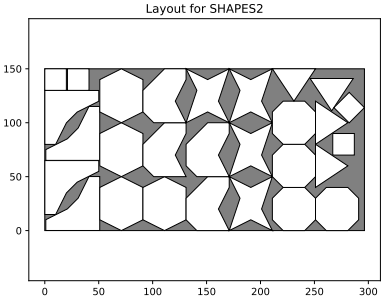
Figure C2: Results obtained for SHAPES1.



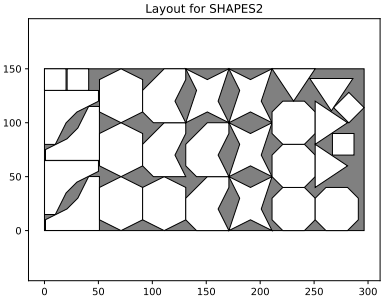
(a) Set of 28 pieces for SHAPES2



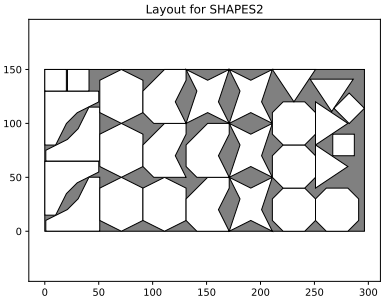
(b) Opus Incertum (Brute Force Search)



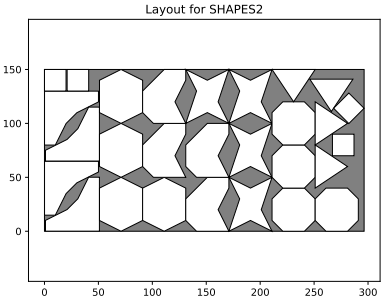
(c) Opus Incertum (Quantum Approximate Optimization Algorithm, simulator)



(d) Opus Incertum (Quantum Approximate Optimization Algorithm, quantum computer)

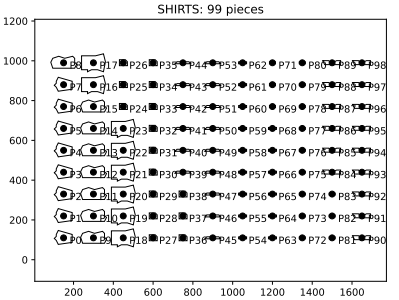


(e) Opus Incertum (Quantum Alternating Operator Ansatz, simulator)

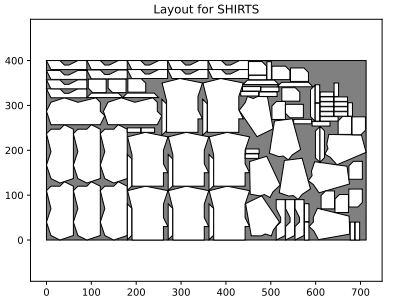


(f) Opus Incertum (Quantum Alternating Operator Ansatz, quantum computer)

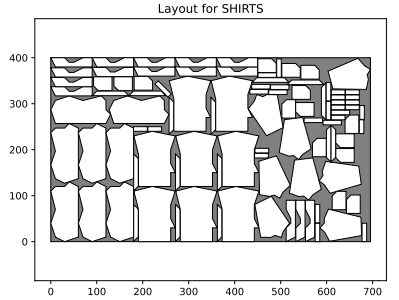
Figure C3: Results obtained for SHAPES2.



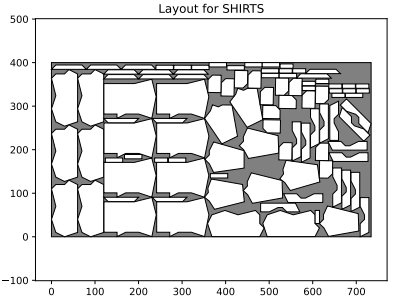
(a) Set of 99 pieces for SHIRTS



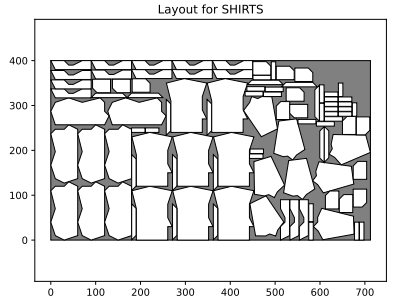
(b) Opus Incertum (Brute Force Search)



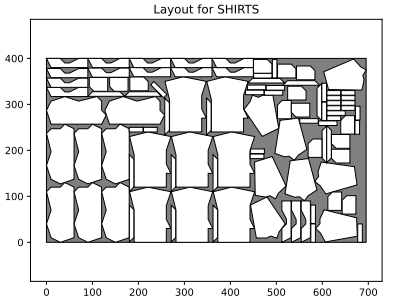
(c) Opus Incertum (Quantum Approximate Optimization Algorithm, simulator)



(d) Opus Incertum (Quantum Approximate Optimization Algorithm, quantum computer)

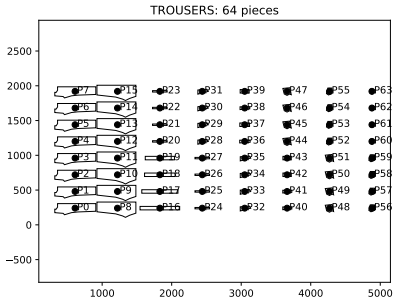


(e) Opus Incertum (Quantum Alternating Operator Ansatz, simulator)

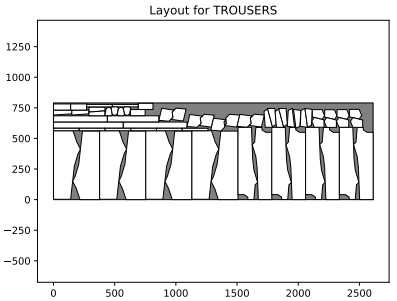


(f) Opus Incertum (Quantum Alternating Operator Ansatz, quantum computer)

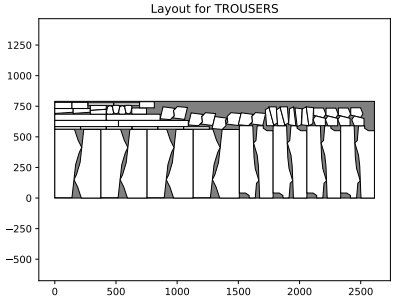
Figure C4: Results obtained for SHIRTS.



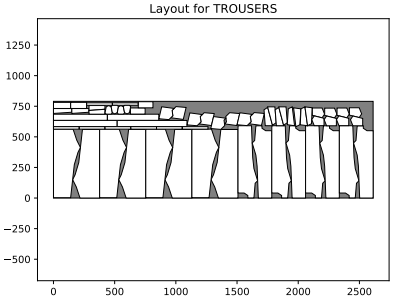
(a) Set of 64 pieces for SHIRTS



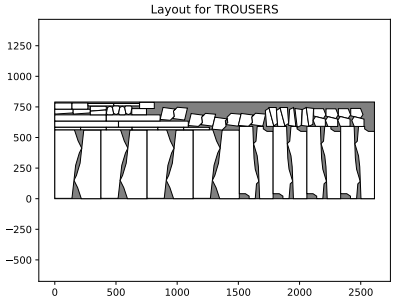
(b) Opus Incertum (Brute Force Search)



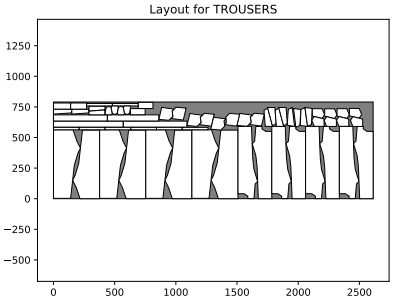
(c) Opus Incertum (Quantum Approximate Optimization Algorithm, simulator)



(d) Opus Incertum (Quantum Approximate Optimization Algorithm, quantum computer)

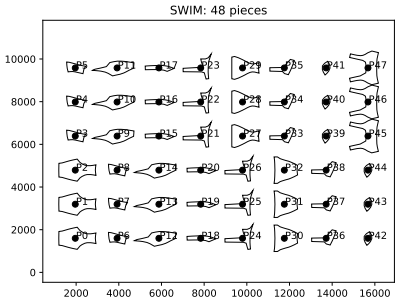


(e) Opus Incertum (Quantum Alternating Operator Ansatz, simulator)

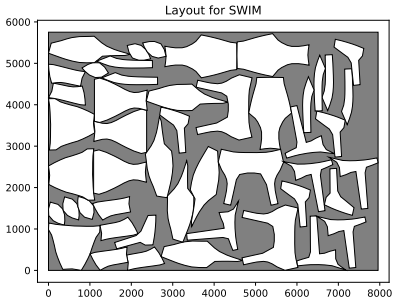


(f) Opus Incertum (Quantum Alternating Operator Ansatz, quantum computer)

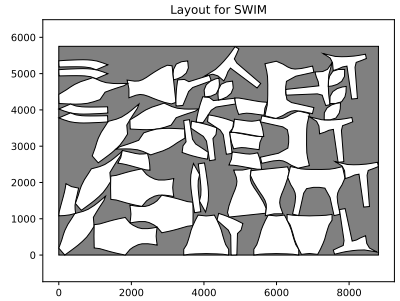
Figure C5: Results obtained for TROUSERS.



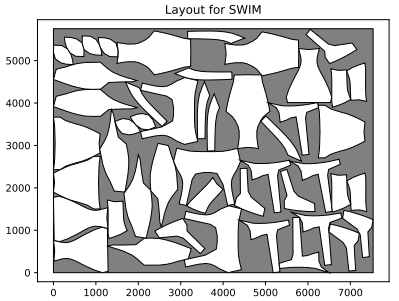
(a) Set of 48 pieces for SWIM



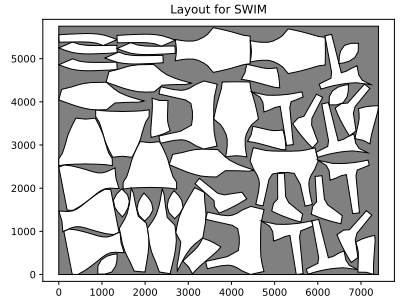
(b) Opus Incertum (Brute Force Search)



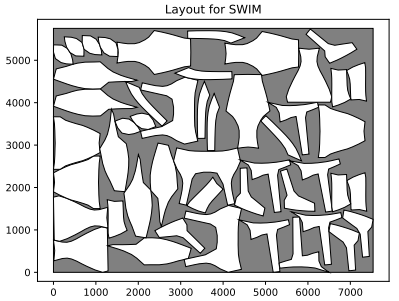
(c) Opus Incertum (Quantum Approximate Optimization Algorithm, simulator)



(d) Opus Incertum (Quantum Approximate Optimization Algorithm, quantum computer)



(e) Opus Incertum (Quantum Alternating Operator Ansatz, simulator)



(f) Opus Incertum (Quantum Alternating Operator Ansatz, quantum computer)

Figure C6: Results obtained for SWIM.