Heuristics for Evolutionary Optimization for the Centered Bin Packing Problem

de Jeu, Luke; Yaman, Anil

published in
Applications of Evolutionary Computation
2024

DOI (link to publisher)
10.1007/978-3-031-56852-7_11

document version
Publisher's PDF, also known as Version of record

Heuristics for Evolutionary Optimization for the Centered Bin Packing Problem

Luke de Jeu and Anil Yaman(✉)

Vrije Universiteit Amsterdam, De Boelelaan 1111,
1081 HV Amsterdam, The Netherlands
a.yaman@vu.nl

Abstract. The Bin Packing Problem (BPP) is an optimization problem where a number of objects are placed within a finite space. This problem has a wide range of applications, from improving the efficiency of transportation to reducing waste in manufacturing. In this paper, we are considering a variant of the BPP where irregular shaped polygons are required to be placed as close to the center as possible. This variant is motivated by its application in 3D printing, where central placement of the objects improves the printing reliability. To find (near) optimum solutions to this problem, we employ Evolutionary Algorithms, and propose several heuristics. We show how these heuristics interact with each other, and their most effective configurations in providing the best solutions.

Keywords: Bin Packing Problem · Heuristics · Evolutionary Algorithms

1 Introduction

The Bin Packing Problem (BPP) is an optimization problem in which a number of objects are placed within a space to maximize a certain objective function. The BPP has a wide range of applications in many fields. Some examples include: reducing material cost in both additive and subtractive manufacturing (for arranging object placement in 3D printing or cutting sheet material) [17,18], reducing costs in transportation (for arranging packages in shipping containers) [14], and warehouse optimization (for optimal placement and storage of goods) [21].

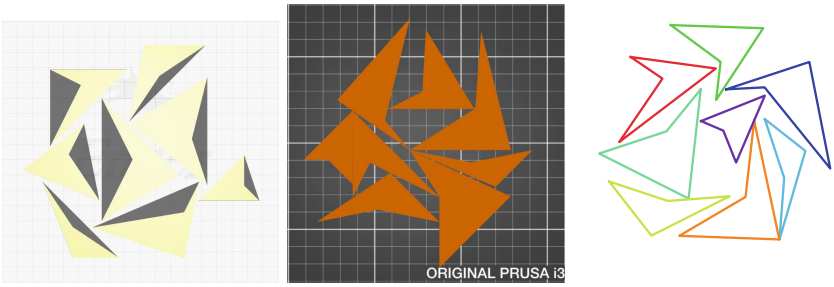
The dimensionality of the BBP problem, in terms of space and objects, can differ depending on the nature of the application domain. In this paper, we consider the field of Fused Filament Fabrication (FFF) 3D printing for additive manufacturing where three-dimensional irregular shaped objects are placed on a printing build plate. The objective in this case is to place the objects as close to the center of the placement area as possible. Due to mechanical limitations of non-industrial and non-professional FFF 3D printers, the heated build plate is

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-56852-7_11.

S. Smith et al. (Eds.): EvoApplications 2024, LNCS 14634, pp. 162–177, 2024.
https://doi.org/10.1007/978-3-031-56852-7_11

sensitive to convection and thermal expansion. This causes heat dissipation and warping of the printing build plate. This effect is observed at the edge of the build plate the most. This is shown to be one of the major causes of printing failure [23,27]. As the popularity of FFF 3D printing has been increasing in recent years, there is a need for more convenient and effective methods for more reliable printing [20]. In this paper, we focus on a possible solution to this problem, where a variant of the BPP aims to place the objects as close to the center as possible. Other possible applications of this problem could also be found in areas such as electronic manufacturing, design of circular circuit boards and urban planning.

Our use case is concerned with the placement of irregular polygons at the center of a printing build plate. Therefore, we formalize this problem as the 2D Irregular Centered BPP (2DICBBP). An example illustration of results from two commonly used software, in comparison with the results of our approach, are shown in Fig. 1. A clear distinction between these results is that, the algorithm in Fig. 1a can handle only convex shapes, whereas, the algorithm in Fig. 1b can also handle concave shapes. In Fig. 1c, we show the results of our proposed approach that can yield placement that appears visually denser than the others.



(a) Ulitmaker Cura (b) PrusaSlicer (c) Proposed approach

Fig. 1. Visual comparison of the auto-arrange functionality between two currently available slicer software, and a solution found by the methods proposed in this research. Each example is over the same concave shapes.

The BBP is difficult to solve computationally. Finding an optimal arrangement is NP-hard [10]. Since computing the optimal solution is challenging, heuristic search algorithms, such as Evolutionary Algorithms (EA) [4], become feasible alternatives for reducing computational time and finding approximate solutions [26].

Many 2DBPP solving methods only focus on packing rectangles of varying dimensions [15]. However, some solving methods are designed for irregular shapes, including concave, convex and non-symmetric polygons of varying number of vertices [24]. The most common application for the 2DBPP is found in industrial environments such as production (which requires the efficient cutting of materials), warehouse management, and transport [8,22]. Due to the nature

of these applications, the vast majority of the 2DBPP solving methods attempt to leave as much unused space as possible on one of the sides of the "bin", while packing as densely as possible from the opposite side. For these reasons, the heuristics such as the "bottom-left" heuristic where the placement of objects starts from the bottom-left and proceeds to top-right, have been used widely [5,6,16].

Guo et al. (2022) presents an extensive review of the decades of research and literature on the 2D Irregular Bin Packing Problem (2DIBPP) [13]. The wide range of environments, methods, and parameters across the literature demonstrate the variability within the 2DIBPP, as well as the wide range of applications. The designed application of the BPP variation is in essence a 3D packing problem simplified to a 2D environment. Because of this, it shares similarities with both 2D and 3D related work. The applications of the 2DIBPP generally lay in removing/cutting material (such as CNC machining or cutting wood, paper, sheet metal, etc.), with the intended purpose being to minimize waste material [19]. The applications of the 3DBPP and 3DIBPP tend to arrange objects (such as filling containers), with the intended purpose being to maximize the transport efficiency [11,12]. While the methods in this paper present a solution in a 2D environment, the underlying purpose is similar to that of a 3D BPP, which is to place items. Additionally, the goal is to place finite number objects within a space, as close to the center as possible. However, we are only concerned with a single bin, thus, placement of the objects in multiple bins is beyond the scope of this paper.

Currently, slicer software for 3D printers attempts to tackle this problem through auto-arrange features. However, this process is not optimized. UltiMaker Cura and PrusaSlicer (Fig. 1) are considered to be two of the most popular slicer software. Cura has millions of users, according to UltiMaker [3]. Both of these slicers use the same fundamental process for this auto-arrange feature, libnest2d [1,2]. While relatively simple and low computational cost, this implementation is non-optimal and inherently flawed when applied to irregular polygons. For instance, this method is unable to utilize the space within a concave shape, and will always use a convex bounding box of each shape [19]. During the course of this research, a new version of PrusaSlicer was released which tackles this exact problem (version 2.6.1). Through the use of numerous optimizers, the newly improved auto-arrange feature is much more effective and can utilize concave shapes, but still shows room for improvement. The recency of this feature demonstrates how improvements on the BPP around a center point are currently in development, in demand, and being utilized.

The 2DBBP with the object of center placement remains largely unexplored. One of the variations of the BPP that shares similar characteristics is referred to as the Circle Packing Problem (CPP). The CPP packs circles within a bin, and in some cases, this bin is also circular. In these cases, the objective is to minimize the size of the circular bin, similar to our objective. Evolutionary methods such as genetic algorithms and differential evolution have shown to be effective for this problem [9]. Additionally, the load-balanced BPP attempts to pack items in such

a way that the center of mass of all the items in the bin is as close to the center of the bin as possible, for which a hill-climbing search method is used [25]. This is one of the few BPP variations for which the item's relation to the center of the bin plays a role in the objective. Thus, extending this literature, in this work, we investigate several heuristics combined with the Evolutionary Algorithms to find effective solutions for the 2-Dimensional Bin Packing Problem for irregular shapes that are required to be placed as close to the center as possible. We focus on four-sided irregular polygons to strike a balance between complexity and feasibility.

2 Method

The problem we tackle in this paper is a version of the BPP where we aim to place polygons within a 2D space such that their distances to the center is minimized. This 2D space will be referred to as the bin. In addition, the polygons should not overlap. Thus, we can state this problem formally as:

Let n be the number of polygons. Each polygon is represented as P_i for i in $[1, n]$. Each polygon P_i has vertices represented as V_{ij} for j in $[1, k]$, where k is the number of vertices in polygon P_i. Let (x_{ij}, y_{ij}) represent the coordinates of the j-th vertex of i-th polygon. Let (X_c, Y_c) represent the coordinates of the center of the bin. The objective function can be formulated as:

Minimize $D = \max(\sqrt{(x_{ij} - X_c)^2 + (y_{ij} - Y_c)^2})$ for all i in $[1, n]$ and all j in $[1, k]$.

Subject to:

- The polygons do not overlap.
- The polygons are within the bin.

To find a (near-)optimum solution to this problem, we use EAs in combination with a set of heuristics we proposed. In the context of the EAs, we refer to a candidate solution as PolyGroup, which encodes an arrangement of the polygons. We can generate new PolyGroups from existing ones using evolutionary operators that are informed by our proposed heuristics.

To calculate the fitness of a PolyGroup, an important evaluation metric is the distance from the center to the furthest coordinate of the polygons within the PolyGroup. This distance can also be described as the radius of the circle that circumscribes the PolyGroup (see Fig. 3 for an example illustration, where circumscribed circles of PolyGroups are shown in red). As a fitness value, it is more meaningful to have an indication of the density of the polygons, as opposed to an absolute value such as this distance/radius. This can help with comparison of performance of two different evolutionary processes that involve different shape initializations. Therefore, to reduce differences in fitness across the use of different polygons, and to gain more insight on the packing density instead of absolute distance, another method is used. The fitness is calculated

as the ratio between the surface area of the polygons, and the surface area of the circumscribed circle.

$$fitness = \frac{\text{surface area of polygons}}{\text{surface area of circumscribed circle}} \tag{1}$$

Within a run, the surface area of the polygons will remain constant. The surface area of the circumscribed circle should decrease, resulting in a higher fitness value. Compared to using the absolute value of the distance to the furthest away coordinate, this ratio method performs the crucial part of improving when a desired change is made (a change which brings the furthest coordinate(s) closer), while additionally providing insight into the density and allowing for more accurate comparisons between runs with different shapes. For comparing the results of different heuristics, while testing over the same shapes, simply using an absolute value would still show the same performance difference. However, this fitness method has as added value that it could somewhat be compared with tests using completely different scales or sizes.

The genotype representation of a PolyGroup encodes the four coordinates of each polygon on 2-dimensions. We also add a rotation parameter to specify the rotation angle of polygons. We fix the number of polygons to 5 and thus the representation of a PolyGroup consists of 45 real-valued numbers.

2.1 Baseline EA

Baseline EA provides a bare minimum algorithm for building our heuristics. This also provides a point of comparison for the success of the proposed heuristics. The algorithm consists of the following steps that are standard in EAs:

Initialization. An initial population of PolyGroups is generated by loading in a pre-generated random set of polygons for each PolyGroup. Not only does this random polygon selection contain the data of the polygon shapes, but also the positional and orientation data which is used as the initial placement, in order to make the experiment repeatable. For a different run, a different randomly generated set of shapes (and starting positions) is initialized, in order to examine the EA behavior over altered starting conditions.

Evolutionary Operators. The Baseline EA involves only the mutation operator to keep it as simple as possible. Other operators, such as recombination, are implemented and discussed in the Heuristics section. Through the mutation operator, a new PolyGroup is generated by duplicating a random PolyGroup from the initialization, or from the survivors of the previous generation. This newly created PolyGroup will then attempt to undergo mutations in the form of changing the position and orientation of the polygons within this duplicate. These changes in position and orientation are made randomly within pre-defined bounds. Only the mutations which do not cause any overlap between the polygons are accepted. A limited number of mutation attempts are made per polygon (i.e., 10 attempts), in order to prevent exceedingly long runtimes when one or more polygons have a small chance of mutating to an unoccupied space, as a result of being surrounded by other polygons. In the instance that none of the

mutations are accepted, the duplicate PolyGroup will have identical polygon placement and orientation as the PolyGroup it was originally duplicated from.

Selection. We use a $(\mu+\lambda)$ selection strategy where λ number of offspring are generated from μ number of parents, the new generation is formed by selecting top μ individuals (ranked based on fitness) from the concatenated set of these two populations [7].

2.2 Heuristics

We propose four heuristics. Each of them is expected to have a positive effect on the overall performance. However, the exact effect of the heuristics may change when applied in combination with one or multiple other heuristics.

Recombination. The Recombination heuristic creates a new PolyGroup from two randomly selected parents. The resulting child PolyGroup is a mix between its two parents. Each polygon in the child receives a center coordinate and a rotation value in between the center and rotational values of the corresponding polygon of the parents. The shape of the corresponding polygon is then built at the assigned center point with the assigned rotation. This allows the Recombination heuristic to place the child polygon at a location and orientation depending on the location and orientation of the corresponding polygon of the parents, while preserving the shapes encoded into the genotypes.

Formally, the Recombination heuristic can be stated as follows: let G be a PolyGroup, containing an arranged set of polygons. Let n be the number of polygons in the PolyGroup. Each polygon P_i for $i \in [1, n]$ consists of a set of four tuples and a rotation value. Each tuple contains the x- and y-coordinate of a vertex of the polygon. These four vertices can be used to calculate a center coordinate, which combined with the rotation value forms (x_i, y_i, t_i). A new PolyGroup G', consisting of polygons P'_i each with a center coordinate and a rotation in degrees (x'_i, y'_i, t'_i), is generated from two parents $G^{(1)}$ and $G^{(2)}$ as follows:

$$x'_i = x_i^{(2)} + (x_i^{(1)} - x_i^{(2)})/2 + r(x_i^{(1)} - x_i^{(2)})/2,$$
$$y'_i = y_i^{(2)} + (y_i^{(1)} - y_i^{(2)})/2 + r(y_i^{(1)} - y_i^{(2)})/2$$

where r is a random value sampled from a triangular distribution between -1 and 1. For t'_i in P'_i, a similar calculation is used, using the same value for r, but utilizing additional steps to ensure the polygon rotates the direction that is shortest (clockwise or counter-clockwise), depending on the shortest rotation between $t_i^{(1)}$ and $t_i^{(2)}$. After the generation of the center coordinate and the rotation value for each P'_i in G', the shape of the corresponding P_i of either parent $G^{(1)}$ or $G^{(2)}$ is imposed on the center coordinate, to regenerate the coordinates of the four vertices of P'_i.

As the creation of polygons P'_i does not check if the newly created polygons are overlapping with any other polygons in G', additional steps are required to correct any overlapping polygons, and to ensure none of the polygons in G' are overlapping. These additional steps are similar to the random mutation operator,

except for a distinct difference. The random mutation would only attempt a number of times to move and rotate a polygon randomly within the pre-defined range, the additional steps to separate any overlapping polygons will continue until no polygons are overlapping, and can move polygons beyond the pre-defined range. Instead of reverting the polygon back to its original position after a failed mutation attempts, such as done by the random mutation, these additional steps continue to move the polygon from its new still overlapping position, after every failed mutation. Through this method, an overlapping polygon will randomly "wander" its vicinity until it no longer overlaps with any other polygons.

Mutation Direction. The mutation operator implemented in Baseline EA moves and rotates each polygon randomly within given parameters. However, this mutation method can be altered to improve the fitness. The Mutation Direction heuristic performs a local search on a polygon based on its local fitness, and accepts solutions only if the local fitness is improved. The local fitness is the fitness of an individual polygon, which is defined as the maximum distance from a vertex of the polygon to the center of the bin. Identical to the mutation operator, the Mutation Direction heuristic has a fixed number of attempts. However, with the Mutation Direction heuristic, only the mutations which improve the local fitness of the polygon are accepted.

The local fitness of a polygon is calculated as: each polygon P has vertices $V_i = (x_k, y_k)$ for k in $[1, m]$ where m is the number of vertices of each polygon. Let (X_c, Y_c) represent the coordinates of the center of the bin. For all i in $[i, n]$

$$fitness_{local} = \max(\sqrt{(x_k - X_c)^2 + (y_k - Y_c)^2}), \text{ for all } k \in [1, m].$$

Mutation Order. For most operations, the order of the polygons and PolyGroups is changed to prevent any unintentional biasing. However, the Mutation Order heuristic offers a different approach. The heuristic sorts polygons by their local fitness (the distance from the center to the furthest coordinate) in ascending order before mutation. This is thought to enhance performance as central polygons can move inward first, freeing up space for outer polygons. For instance, if polygon A is blocked by polygon B from moving closer to the center, polygon A cannot move inward. But if polygon B had moved inward first, polygon A could have followed suit. This heuristic aims to improve such situations where a movable polygon blocks another polygon's mutation. By mutating polygons in ascending local fitness order, polygons closer to the center mutate first, potentially creating space for outer polygons to move inward.

Variable Step Size. The translation and rotation amount for each mutation is random, within bounds. This random mutation strength is then multiplied by the step size. The step size can be adjusted to strengthen or weaken the maximum amount by which a polygon can be mutated. For the Baseline EA, this step size remains constant throughout all generations. The Variable Step Size heuristic introduced a step size which changes depending on the current generation number. The variable step size is determined according to the following

formula:

$$\text{variable step size} = \frac{\text{absolute initial step size}}{\sqrt{\text{current generation number} + 1}} \tag{2}$$

The result of this variable step size formula is that the step size will decrease over the course of the generations. The step size will be the largest for the first generation, and will be most drastically reduced over the first few generations, after which the reduction in step size will become less drastic.

3 Experimental Setup

Each of the mentioned heuristics can be active or inactive, regardless of the activation state of the other heuristics. Therefore, to examine the effect of each heuristic on the fitness of the EA, as well as examining the effect in combination with other heuristics, a binary decision table can be constructed with 16 configurations for the four heuristics. This can be seen in Table 1. We run 30 independent evolutionary processes for each configuration for 500 generations with a population size of 50. Using our implementation, a single evolutionary process, of 500 generations, over one configuration, resulted in a runtime between 30 and 120 s depending on the number of heuristics used. Our implementation is on Python 3.10, and ran on an AMD Ryzen 5 3600X processor, without parallelization. The algorithm could further be optimized to increase the speed. In comparison, other available software packages (demonstrated in Fig. 1a and 1b) were implemented using C++ and can provide results in seconds. However, they do not employ an evolutionary optimization approach to provide better placement.

During the initialization process of each run, all configurations are initialized with the same set of randomly generated shapes and starting positions. Therefore, all configurations within a run share the exact same starting parameters. For each different run, a different set of random shapes and starting positions is used. All starting conditions are equal within a run, but are different across runs.

The final fitness value of the 30 runs of a configuration, can be compared to the 30 final fitness values of another configuration using the Wilcoxon signed-rank test. This test can be used to compare each configuration with every other configuration, and examine its statistical significance.

4 Results

4.1 Configuration Performance Results

After running each configuration, fitness progression plots have been created for each individual configuration. The fitness plot of each individual configuration, along with their average final fitness (AFF - average of final fitness value at the end of the evolutionary processes), and the standard deviation range of the 30 runs, can be found in Supplementary Material. Additionally, the complete

Table 1. The Configurations Setup table shows the properties of each configuration. The name of each configuration is shortened to *config* and a number. When a heuristic is stated as *True*, it is applied to that configuration. When a heuristic is set to *False*, it is not.

Configurations Setup				
Configurations	Recombination	Mutation Direction	Mutation Order	Variable Step Size
config 1	False	False	False	False
config 2	False	False	False	True
config 3	False	False	True	False
config 4	False	False	True	True
config 5	False	True	False	False
config 6	False	True	False	True
config 7	False	True	True	False
config 8	False	True	True	True
config 9	True	False	False	False
config 10	True	False	False	True
config 11	True	False	True	False
config 12	True	False	True	True
config 13	True	True	False	False
config 14	True	True	False	True
config 15	True	True	True	False
config 16	True	True	True	True

evolutionary process of a single run over all configurations can be seen in this video: https://rb.gy/7i9diu.

The numerical results can be found in Table 2, sorted by (AFF) in descending order. The results show emerging groups of four configurations, which each have similar average final fitness values, and share heuristic properties. Configurations 10, 12, 2, and 4 form Group A, and have the highest AFF. All four configurations of Group A have Mutation Direction set to False, and Variable Step Size set to True. Configurations 14, 16, 15 and 13 form Group B. All configurations in Group B have Recombination and Mutation Direction set to True. Configurations 5, 6, 7 and 8 form Group C. All configurations in Group C have Recombination set to False and Mutation Direction set to True. Finally, configurations 11, 1, 9 and 3 form Group D. All configurations in Group D have Mutation Direction and Variable Step Size set to False.

Figure 2 shows the average fitness trends during the evolutionary processes. It is clear that each configuration within a group shows similar behavior/progression as the other configurations in its group. Additionally, each group shows different behavior/progression compared to the other groups. Group A and D show similar behavior, as they both show relatively slow progression over

Table 2. The numerical results of the runs from Fig. 2, sorted by AFF in descending order. The column labeled as "Statistical Test" shows the statistical significance of the results relative to the first row ("=" and "+" representing statistical similarity or difference respectively) based on the Wilcoxon signed-rank test, and a p-value of 0,05. The groups are defined based on the ranking on the AFF of the configurations.

Configurations Results by AFF								
Configurations	Heuristics				AFF	Std Range	Statistical Test	Group
	Recombination	Mutation Direction	Mutation Order	Variable Step Size				
config 10	True	False	False	True	**0,633**	0,108		A
config 12	True	False	True	True	0,624	0,084	=	
config 2	False	False	False	True	0,611	0,1	=	
config 4	False	False	True	True	0,609	0,077	+	
config 14	True	True	False	True	0,562	0,121	+	B
config 16	True	True	True	True	0,546	0,117	+	
config 15	True	True	True	False	0,529	0,144	+	
config 13	True	True	False	False	0,518	0,122	+	
config 5	False	True	False	False	0,483	0,123	+	C
config 6	False	True	False	True	0,481	0,135	+	
config 7	False	True	True	False	0,48	0,117	+	
config 8	False	True	True	True	0,476	0,128	+	
config 11	True	False	True	False	0,453	0,089	+	D
config 1	False	False	False	False	0,441	0,076	+	
config 9	True	False	False	False	0,439	0,074	+	
config 3	False	False	True	False	0,427	0,065	+	

the evaluations, but continue to increase throughout all 500 generations. The main performance difference between Group A and D is that Group A has a much stronger fitness increase in the first few generations. The only difference in heuristics between these two groups, is that Group A has Variable Step Size set to True, while Group D has Variable Step Size set to False. Both overlap in sharing Mutation Direction to be set as False. Group B and C also show comparable differences in their behavior. Both show a very strong increase in fitness in the first few generations. The configurations in these groups increase their performance to such an extent that their fitness temporarily surpasses the fitness of the configurations in Group A (which will in later generations overtake Group B and C in fitness). The only difference in heuristics between Group B and C is that Group B has Recombination set to True, while Group C has Recombination set to False. Both overlap in sharing Mutation Direction as True.

To have a visual understanding of the difference in performance between each group, the polygon positioning of the first and last generation can be observed for a configuration of each group in Fig. 3. The example configurations are config

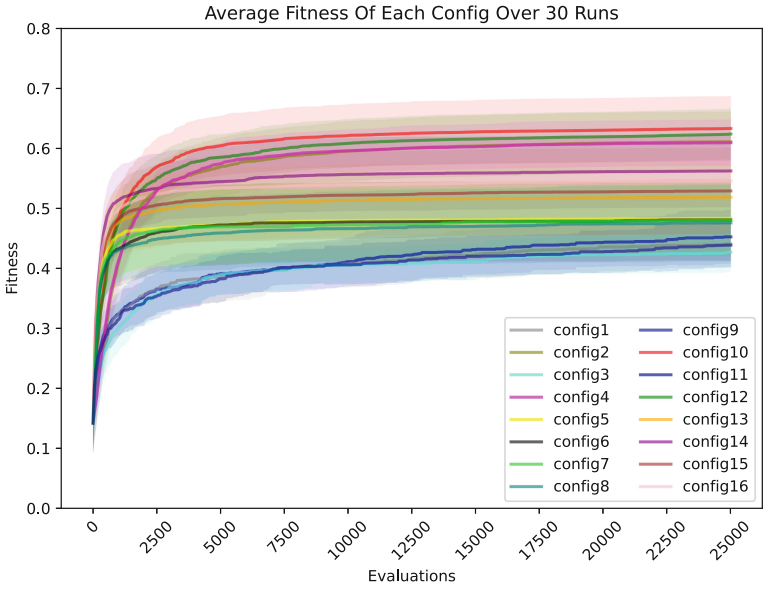


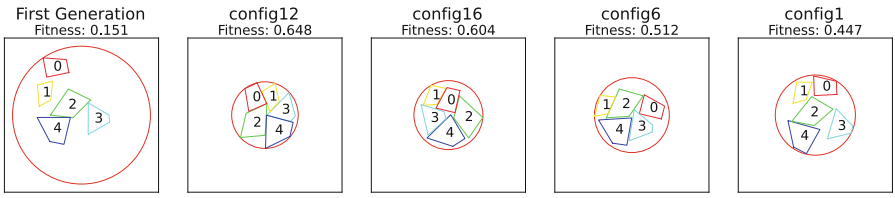
Fig. 2. The average fitness of each configuration, using 30 runs per configuration. Each run lasted 500 generations (25000 evaluations in total for population size of 50). Using evaluations on the x-axis allows for more accurate plot comparison when other parent/children sizes are used.

12 for Group A, config 16 for Group B, config 6 for Group C and config 1 for Group D. These configurations are the second best in each group, to represent the average performance of the group. The visualizations of the polygon positioning of all configurations, in addition to the group examples of two other runs, can be found in Supplementary Material.

The Std range of each configuration, as seen in Table 2, gives an indication as to the consistency between runs. This shows how sensitive the configurations are to changes in the shapes/initial positions, and how much a configuration relies on randomness and its vulnerability towards local maxima. The Std range between the configurations within each group tends to be similar. The average Std range for groups A, B, C and D are 0.092, 0.126, 0.126 and 0.076 respectively.

4.2 Individual Heuristic Effects

While the results in Fig. 2 and Table 2 show which combinations of heuristics result in the highest fitness value, the effect of the individual heuristics must also be examined. For example, when examining the effect of Recombination, one of the pairs to examine is the pair of config 1 and config 9. As seen in Table 1, these configurations share the same heuristics, but config 9 is with the Recombination heuristics whilst config 1 is without. Pairs such as these can be examined to gain better insight into what the effect is of each individual heuristic on different



(a) First generation (b) Group A (c) Group B (d) Group C (e) Group D

Fig. 3. The polygon positioning of the first and last generation of one configuration per group. Each of the visualizations is over the same run (run 1), therefore having the same shapes and starting position.

heuristic configurations. These pairs, and the effect of each individual heuristic, can be seen in Table 3.

Table 3a examines the effect of the Recombination heuristic. For each use of the Recombination heuristic, it shows a positive or insignificant effect. This indicates that the Recombination heuristic as implemented in this research did not have a measurable negative effect, and only had a positive or insignificant effect. Only one of the configuration pairs between 1–4 has a significant positive effect, while all configuration pairs between 5–8 have a significant positive effect. This would seem to indicate that the effect of the Recombination heuristic is influenced by the presence of the Mutation Direction heuristic. Table 3b examines the effect of the Mutation Direction heuristic. The effect of this heuristic is significant for each configuration pair, indicating it has a strong effect on the performance, regardless of the other heuristics. However, whether the Mutation Direction heuristic has a significant positive or negative effect seems to be reliant on the presence of other heuristics. For configuration pairs 1, 3, 5 and 7, the heuristic has a significant positive effect, while for pairs 2, 4, 6, and 8, the heuristic has a significant negative effect. This clearly shows that the Mutation Direction heuristic as implemented in this research is heavily reliant on the Variable Step Size heuristic. When the Variable Step Size heuristic is False, then the presence of the Mutation Direction heuristic shows a significant positive effect on the final fitness. However, when the Variable Step Size heuristic is True, then the presence of the Mutation Direction heuristic shows a significant negative effect on the final fitness. Table 3c examines the effect of the Mutation Order heuristic. The Mutation Order Heuristic has no significant effect for configuration pairs 1–4, while showing more significant effect in configurations 5–8. The only two configuration pairs with a significant positive effect, pair 5 and 7, seem to indicate that the Mutation Order heuristic only has a somewhat significant effect when there is Recombination and no Variable Step Size present.

Finally, Table 3d examines the effect of the Variable Step Size heuristic. The results clearly demonstrate that the Variable Step Size heuristic only has a positive effect on the final fitness when the Mutation Direction heuristic is False. If

Table 3. Each sub-table shows the numbers of configuration pairs which are identical, besides being with or without a single focused heuristic. The presence of this focused heuristic either has a positive (P), negative (N), or statistically insignificant (I) effect on the AFF. The non-focused heuristics are kept constant between the two configurations of each pair. The heuristics are shortened to a letter for compactness. Recombination (R), Mutation Direction (D), Mutation Order (O), Variable Step Size (V). The use of these heuristics is denoted as either True (T) or False (F).

Pair nr.	Without	With	D	O	V	Effect on AFF
1	config 1	config 9	F	F	F	I
2	config 2	config 10	F	F	T	I
3	config 3	config 11	F	T	F	P
4	config 4	config 12	F	T	T	I
5	config 5	config 13	T	F	F	P
6	config 6	config 14	T	F	T	P
7	config 7	config 15	T	T	F	P
8	config 8	config 16	T	T	T	P

(a) Focused heuristic: Recombination

Pair nr.	Without	With	R	O	V	Effect on AFF
1	config 1	config 5	F	F	F	P
2	config 2	config 6	F	F	T	N
3	config 3	config 7	F	T	F	P
4	config 4	config 8	F	T	T	N
5	config 9	config 13	T	F	F	P
6	config 10	config 14	T	F	T	N
7	config 11	config 15	T	T	F	P
8	config 12	config 16	T	T	T	N

(b) Focused heuristic: Mutation Direction

Pair nr.	Without	With	R	D	V	Effect on AFF
1	config 1	config 3	F	F	F	I
2	config 2	config 4	F	F	T	I
3	config 5	config 7	F	T	F	I
4	config 6	config 8	F	T	T	I
5	config 9	config 11	T	F	F	P
6	config 10	config 12	T	F	T	I
7	config 13	config 15	T	T	F	P
8	config 14	config 16	T	T	T	N

(c) Focused heuristic: Mutation Order

Pair nr.	Without	With	R	D	O	Effect on AFF
1	config 1	config 2	F	F	F	P
2	config 3	config 4	F	F	T	P
3	config 5	config 6	F	T	F	I
4	config 7	config 8	F	T	T	I
5	config 9	config 10	T	F	F	P
6	config 11	config 12	T	F	T	P
7	config 13	config 14	T	T	F	I
8	config 15	config 16	T	T	T	I

(d) Focused heuristic: Variable Step Size

the Mutation Direction heuristic is True, then the Variable Step Size heuristic has no significant effect.

5 Discussion

In conclusion, the 2DICBPP, as implemented in this paper, demonstrates consistent positive and negative changes when certain heuristic combinations are applied. However, these effects are not always the same, and can vary greatly depending on the presence of other heuristics. Each heuristic impacts the performance and behavior of the EA differently. Additionally, the effect of the heuristics themselves are impacted by the behavior of the EA.

The Recombination heuristic has only shown positive or insignificant effects. Its positive effects are most notable when the EA without the Recombination heuristic is prone to getting stuck at a local optimum. This indicates that the main positive effect of the Recombination heuristic comes from introducing randomness and new variations. The effect of the Mutation Direction and Variable Step Size heuristics showed high dependence on the presence of one another. The presence of the Mutation Direction will make the effect of the Variable Step Size insignificant, which otherwise would be positive. The presence of the Variable Step Size will make the Mutation Direction have a negative effect, which would otherwise have a positive effect. The Mutation Order heuristic generally had a very insignificant and mild effect, but seems more influential in EAs with more variety among the population.

These findings demonstrate how the effect of individual heuristics on the performance of the EA can be highly dependent on the presence of other heuristics, within the confines of the 2DICBPP.

6 Conclusion

The objective of this paper is to formalize a variant of a 2-dimensional bin packing problem that can be applied to 3D printing for object placement to improve efficiency and reliability. We propose several heuristics within the EAs to improve the placement of polygon shaped objects as close to the center as possible, in order to reduce the effect of mechanical limitations.

We demonstrate the effects of the proposed heuristics, when they are applied individually or in combination. Each heuristic contains some variables or design decisions, which may or may not form an accurate representation of similar heuristics. Additional research that can examine varieties of other types of heuristics may lead to new insights or allow for heuristic combinations that can have a positive impact. Although our visual inspection of the results of the existing software packages indicated that they could be improved, quantitative assessment of their performance and comparing to our results can provide further insights.

The main comparison between the configurations was using the fitness values after 500 generations. While it is likely that many applications which require some solution for the 2DICBPP have the time to run an extensive EA with hundreds of generations, some applications may be more time sensitive and require a much shorter run. Researching the result of the heuristics on multiple generation intervals may give additional insights into effective heuristics for different use cases.

Finally, an optimization problem such as the 2DICBPP may benefit from stepping away from traditional EA implementations and applying more versatile methods. For example, the Mutation Direction heuristic may be used mutualistically with the Variable Step Size heuristic, or a type of shuffling/randomization heuristic, when being alternated at crucial moments. EAs which apply different heuristics throughout the same run, depending on generation intervals or if certain triggers are met, may result in more efficient solving methods, but will need further research.

References

1. GitHub - tamasmeszaros/libnest2d: 2D irregular bin packaging and nesting library written in modern C++ – github.com. https://github.com/tamasmeszaros/libnest2d. Accessed 13 Oct 2023
2. GitHub - Ultimaker/pynest2d: Python bindings for libnest2d – github.com. https://github.com/Ultimaker/pynest2d. Accessed 12 Nov 2023
3. UltiMaker Cura – ultimaker.com. https://ultimaker.com/software/ultimaker-cura/. Accessed 30 Oct 2023
4. Bäck, T., Schwefel, H.P.: An overview of evolutionary algorithms for parameter optimization. Evol. Comput.**1**(1), 1–23 (1993). https://doi.org/10.1162/evco.1993.1.1.1. https://doi.org/10.1162/evco.1993.1.1.1
5. Berkey, J.O., Wang, P.Y.: Two-dimensional finite bin-packing algorithms. J. Operational Res. Soc. **38**(5), 423–429 (1987). https://doi.org/10.1057/jors.1987.70
6. Burke, E., Hellier, R., Kendall, G., Whitwell, G.: A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem. Oper. Res. **54**(3), 587–601 (2006). https://doi.org/10.1287/opre.1060.0293
7. Costa, L., Oliveira, P.: An evolution strategy for multiobjective optimization. In: Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600). IEEE (2002). https://doi.org/10.1109/cec.2002.1006216
8. Côté, J.F., Haouari, M., Iori, M.: Combinatorial benders decomposition for the two-dimensional bin packing problem. INFORMS J. Comput.**33**(3), 963–978 (2021). https://doi.org/10.1287/ijoc.2020.1014
9. Flores, J.J., Martínez, J., Calderón, F.: Evolutionary computation solutions to the circle packing problem. Soft. Comput. **20**, 1521–1535 (2016)
10. Garey, M.R., Johnson, D.S.: Approximation algorithms for bin packing problems: a survey. Analysis and Design of Algorithms in Combinatorial Optimization, pp. 147–172 (1981). https://doi.org/10.1007/978-3-7091-2748-3_8
11. Gonçalves, J.F., Resende, M.G.: A biased random key genetic algorithm for 2d and 3d bin packing problems. Int. J. Prod. Econ. **145**(2), 500–510 (2013). https://doi.org/10.1016/j.ijpe.2013.04.019
12. Griffiths, V., Scanlan, J.P., Eres, M.H., Martinez-Sykora, A., Chinchapatnam, P.: Cost-driven build orientation and bin packing of parts in selective laser melting (SLM). Eur. J. Oper. Res. **273**(1), 334–352 (2019). https://doi.org/10.1016/j.ejor.2018.07.053
13. Guo, B., Zhang, Y., Hu, J., Li, J., Wu, F., Peng, Q., Zhang, Q.: Two-dimensional irregular packing problems: a review. Front. Mech. Eng. 8, August 2022. https://doi.org/10.3389/fmech.2022.966691
14. Kang, K., Moon, I., Wang, H.: A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. Appl. Math. Comput.**219**(3), 1287–1299 (2012). https://doi.org/10.1016/j.amc.2012.07.036
15. Kao, C.Y., Horng, J.T.: On solving rectangle bin packing problems using genetic algorithms. In: Proceedings of IEEE International Conference on Systems, Man and Cybernetics (1994). https://doi.org/10.1109/icsmc.1994.400073
16. Laabadi, S., Naimi, M., Amri, H.E., Achchab, B.: A binary crow search algorithm for solving two-dimensional bin packing problem with fixed orientation. Procedia Comput. Sci. **167**, 809–818 (2020). https://doi.org/10.1016/j.procs.2020.03.420
17. Lamas-Fernandez, C., Bennell, J.A., Martinez-Sykora, A.: Voxel-based solution approaches to the three-dimensional irregular packing problem. Oper. Res. **71**(4), 1298–1317 (2023). https://doi.org/10.1287/opre.2022.2260

18. Lodi, A., Martello, S., Vigo, D.: Approximation algorithms for the oriented two-dimensional bin packing problem. Eur. J. Oper. Res. **112**(1), 158–166 (1999). https://doi.org/10.1016/s0377-2217(97)00388-3 ¡error l="308" c="Invalid ¡error l="306" c="Invalid ¡error l="307" c="Invalid
command: paragraph not started." /¿ command: paragraph not started." /¿ command: paragraph not started." /¿ ¡error l="308" c="Invalid ¡error l="306" c="Invalid
command: paragraph not started." /¿ command: paragraph not started." /¿
19. Lopez, E., Ochoa, G., Terashima-Marín, H., Burke, E.: An effective heuristic for the two-dimensional irregular bin packing problem. Ann. Oper. Res. **206**, 241–264 (2013). https://doi.org/10.1007/s10479-013-1341-4
20. Mpofu, T.P., Mawere, C., Mukosera, M.: The impact and application of 3d printing technology. Int. J. Sci. Res., June 2014
21. Munien, C., Ezugwu, A.E.: Metaheuristic algorithms for one-dimensional bin-packing problems: a survey of recent advances and applications. J. Intell. Syst. **30**(1), 636–663 (2021). https://doi.org/10.1515/jisys-2020-0117
22. Puchinger, J., Raidl, G.R., Koller, G.: Solving a real-world glass cutting problem. In: Evolutionary Computation in Combinatorial Optimization, pp. 165–176. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24652-7_17
23. Tamir, T.S., Xiong, G., Fang, Q., Dong, X., Shen, Z., Wang, F.Y.: A feedback-based print quality improving strategy for FDM 3d printing: an optimal design approach. The International J. Adv. Manuf. Technol. **120**(3-4), 2777–2791 (2022). https://doi.org/10.1007/s00170-021-08332-4
24. Terashima-Marín, H., Ross, P., Farías-Zárate, C., López-Camacho, E., Valenzuela-Rendón, M.: Generalized hyper-heuristics for solving 2d regular and irregular packing problems. Ann. Oper. Res. **179**, 369–392 (2010)
25. Trivella, A., Pisinger, D.: The load-balanced multi-dimensional bin-packing problem. Comput. Oper. Res. **74**, 152–164 (2016). https://doi.org/10.1016/j.cor.2016.04.020
26. Volna, E.: Genetic algorithms for two dimensional bin packing problem. In: AIP Conference Proceedings, vol. 1648, p. 550002 (2015). https://pubs.aip.org/aip/acp/article/1648/1/550002/802815/Genetic-algorithms-for-two-dimensional-bin-packing
27. Zhang, J., Wang, X.Z., Yu, W.W., Deng, Y.H.: Numerical investigation of the influence of process conditions on the temperature variation in fused deposition modeling. Mater. Des. **130**, 59–68 (2017). https://doi.org/10.1016/j.matdes.2017.05.040