
Proceedings Paper:
Webster, A., Jia, X. and Xie, S.Q. (2024) Hyper-Heuristics for Irregular Object Multi-Container Packing. In: Proceedings of 2023 29th International Conference on Mechatronics and Machine Vision in Practice (M2VIP). 2023 29th International Conference on Mechatronics and Machine Vision in Practice (M2VIP), 21-24 Nov 2023, Queenstown, New Zealand. IEEE. ISBN: 9798350325621.

https://doi.org/10.1109/m2vip58386.2023.10413433

Hyper-Heuristics for 2D Irregular Object Multi-Container Packing

Aron Webster*, Xiaodong Jia†, Sheng Quan Xie‡
*†*School of Chemical and Process Engineering* ‡*School of Electronic and Electrical Engineering*
University of Leeds
Leeds, UK
{*pmaow, †X.Jia, ‡S.Q.Xie}@leeds.ac.uk

***Abstract*—This paper presents a preliminary study on the use of hyper-heuristics for solving multi-container irregular object packing problems. The work is part of a larger project which aims to develop a robotic packing system for the application of packing nuclear waste into containers. As part of this project, a novel packing optimisation scheme is proposed to plan how to pack the objects into given containers. We propose a new hyper-heuristic algorithm for optimising both packing order and placement heuristics for each object. We analyse the performance and highlight the strengths and weaknesses of the proposed approach. The work presented in this paper is conducted on 2D datasets, however all the methodology can be adapted to 3D packing. Based on the comparison of our results against a 2D multi-container packing algorithm from literature, our approach shows promise, however it suffers primarily from needing large computation times to find good solutions owing to the large number of combinations to the problem. Future work will focus on ways to cut down the number of combinations (without degrading the quality of solutions) as well as trying to speed up the packing algorithm by limiting the number of rotation angles for the objects.**

***Index Terms*—2D packing, Multi-container packing, Hyper-heuristic, Genetic algorithm, Optimisation**

I. Introduction

For a given set of objects, packing optimisation (PO) describes the task of determining what position/orientation each object will have within a container such that the number of objects in the container is maximised, subject to constraints (e.g. maximum weight limit on a container). PO has attracted numerous studies over the years, due of its relevance in areas such as the transportation of goods (shipping and home delivery by the likes of Amazon), additive manufacturing (more parts printed per tray), engineering design (component layout in engine compartments), garment manufacturing (packing shapes to minimise wasted fabric), etc.

One popular method of PO in literature is sequential packing, where objects are placed one by one into a container according to given placement rules (or placement 'heuristics'). Most existing sequential PO algorithms only consider a single placement heuristic and focus on optimising the structure by optimising the order that objects are packed [1,2]. The problem is that by only using a single placement heuristic, the number of sites where an object can be placed is limited. In contrast, a hyper-heuristic approach employs multiple placement heuristics to expand the number of potential sites for an object [3-6]. Hyper-heuristics have shown promise with packing problems in the past, however the number of studies focusing on irregular object multi-container packing using hyper heuristics is very limited.

This paper presents a multi-container packing algorithm with 4 placement heuristics (outlined in section 3.B), combined in a novel hyper-heuristic algorithm for the packing of 2D irregular shapes, where both the order that objects are packed, and the placement heuristics used for each object are optimised. The efficacy of the proposed approach is tested using benchmark datasets and the results are compared to previous results in literature. Based on the results, several areas for future work are proposed.

II. Problem Definition

In this study, the goal of optimisation is to find both an ordering for the set of objects, and a placement heuristic for *each* object such that, when the objects are packed in this order with each object placed according to its prescribed heuristic, the number of containers N needed to pack all the objects is minimised.

Let $O = \{o_1, o_2, ..., o_n\}$ denote a set of n 2D irregular objects which are to be packed into identical rectangular containers, with width and length denoted as W and L respectively. Additionally, let $H = \{h_1, h_2, ..., h_k\}$ denote a set of placement heuristics available to the packing algorithm, where k is the number of placement heuristics available.

The optimisation variables can thus be defined as a 2-tuple set: $\Omega = \{(o_1, h_i^1), (o_2, h_i^2), ..., (o_n, h_i^n)\}$ where $i \in 1, ..., k$. Each entry in the set Ω contains an object $o \in O$ with an associated placement heuristic $h \in H$ which is used to pack that object.

Assuming the objects are packed in the order they appear in Ω the goal of optimisation can be stated as a desire to minimise N by optimising 1) the order of objects in the set Ω (by changing the permutation of the pair entries in the set), and 2) the choice of placement heuristics used to pack each object (by changing the index i of each h associated with each object in Ω), subject to the following constraints:

1) No overlap between packed objects.
2) All packed objects are within the container boundary.

Using N directly as the optimisation objective will result in tied solutions for cases where there are multiple solutions with the same number of containers. Hence, we instead adopt the same objective as in [7] which aims to maximise the percentage usage of each container. This is expressed as:

$$F = \frac{\sum_{j=1}^{N} U_j^2}{N} \tag{1}$$

Where U_j is the utilisation ratio of each container $j \in 1, ..., N$ and is defined as:

$$U_j = \frac{\sum_{m=1}^{n_j} a_m}{LW} \tag{2}$$

Where a_m is the area of object $m \in O$.

III. Solution Algorithm

To start packing a set of objects, a single container is opened and the objects are placed one by one into the container starting with the first object in the list. Each object is placed according to the placement heuristic assigned to it. If the next object in the list cannot be packed into the current container, the container is closed off and a new container is opened. The algorithm then continues packing the objects into the new container until the next object cannot fit. This process repeats until all the objects have been packed.

The optimisation of object ordering and placement heuristics is performed by a Genetic Algorithm (GA) and the packing of the individual objects into the containers is performed by a packing software called DigiPac™ [8] developed by StructureVision© at the University of Leeds.

A. Optimisation Algorithm

As stated, the optimisation of the packing order and heuristics for each object is performed by a GA. The population of the GA is made up of individual solutions, or 'chromosomes', where each chromosome is equivalent to an instance of Ω and the number of chromosomes is set by the user. Each gene in a chromosome is equivalent to a 2-tuple in the set Ω and contains one object and a heuristic to pack it. The order of the genes determines what order the objects will be packed.

In each generation, the GA employs crossover and mutation operators to create a new population of solutions ('offspring') for the next generation. For selecting parent chromosomes for crossover or mutation, tournament selection is used. When generating the new population, the algorithm will generate a random number between [0,1]. If the number is less than the user defined crossover rate, the algorithm performs crossover (producing two offspring), otherwise the algorithm performs mutation (producing a single offspring). This process iterates until a full population of new solutions has been generated.

For crossover, Davis's order crossover [9] is used to generate two offspring with different gene permutations. Each gene in the each child chromosome will retain the placement heuristic is possessed in the parent chromosomes. For mutation, the operator selects two different genes at random from a parent chromosome and swaps their positions. The operator then randomly changes the value of i for each h in the two selected genes to a different value of i from $1, ..., k$.

The GA also utilises elitism to retain a small portion of the best solutions from each generation to carry over to the next generation. In doing so, the best solution found is retained throughout the GA run.

B. Packing Model

The packing software DigiPac™ is used to pack a set of shapes. DigiPac™ works by digitising objects, to pixels in 2D or voxels in 3D, which enables the algorithm to easily represent arbitrary shapes. The packing space is likewise digitised (to form a lattice grid) with the resolution of the lattice grid (and in turn the objects) being set by the user.

To pack a shape, the algorithm fixes the rotation angle of the object to pack and then systematically translates the object, in discrete steps, across each grid cell in the packing space lattice, from the bottom of the container to the highest point of the existing packing structure. Each time the object is moved, an overlap detection is performed to test if the current object overlaps with any of the previously packed objects or the container boundary. If there is no overlap, the algorithm stores this location as a feasible packing site. Once the lattice grid has been traversed, the object is rotated by a fixed angle increment (which is set by the user), and the shape is once again translated across the lattice grid. This process is repeated until the object has completed a full rotation. The algorithm then searches through all the stored feasible sites and selects a site based on the placement heuristic assigned to that object.

Currently, there are 4 placement heuristics implemented in DigiPac™:

- **Height Minimisation** – the object is placed in a location which minimises the height of the object in the packing structure.
- **Seat Position Minimisation** – the object is placed in the lowest available site in the packing structure.
- **Contact Area Maximisation** – the object is placed in the location which results in maximum contact area between the object and the packed objects in the container.
- **Contact Number Maximisation** – the object is placed in the location which maximises the number of packed objects the new object is in contact with.

IV. Implementation

DigiPac™ is implemented in C++ and the GA is implemented in MATLAB. To pack the shapes, MATLAB writes the shape vertices to a text file, calls DigiPac™ which then reads in the shape data, pixelises the shapes and then packs them. The packed shapes and utilisation ratio are then printed to a text file from DigiPac and read back into MATLAB. The programme is run on a PC with an AMD Ryzen 5 5600x processor and 16Gb of RAM.

The initial population of solutions comprises randomly ordered solutions with the heuristics assigned to each object likewise generated at random. For all the tests in this paper the algorithm was run for 100 generations with a population

size of 50. The Crossover rate was set to 0.5, the number of elite solutions was set to 5 and the number of solutions for tournament selection was set to 10.

A. Data

For testing the algorithm, 5 datasets were selected, all of which are available on the ESICUP website (https://www.euro-online.org/websites/esicup/data-sets/). The datasets are shown in Table 1 along with the number of objects in each set (n) and the permitted rotations as given by the original authors of each set.

The results are benchmarked against the results for the same datasets used in [7] for the 'Nest-MB' (Medium Bins) instances, under the same container size conditions. For the rotation angles of the objects, two separate trials were run on each set of shapes; one where the rotations were the same as the given rotations and another where the shapes were permitted a full rotation in 1-degree increments.

In [7], the authors test packing the shapes using the given angles as well as using free rotation. Within the context of their paper, free rotation refers to the fact that objects can have any angle between 0 and 360 degrees, however this does not mean that objects are continuously rotated when being packed. The authors use a pre-processing step before packing each object to identify promising angles for that object, after which, the object is packed for each identified angle (where the angle remains fixed in each case).

V. Results

Table 2 lists the results of this study along with the results obtained in [7] for the same data sets, both of which are averaged over multiple runs. Fig. 1. shows an example of a packed structure obtained using full rotation. For our data, the algorithm was run 10 times for each dataset in both cases (given rotation and full rotation). The best results for each dataset are highlighted in bold.

Looking at our results in table 2, our algorithm performed better on all cases using full rotation in 1-degree increments when compared to using the given rotation angles. This is unsurprising as using full rotation allows the algorithm to find more potential sites for each object. When comparing our results (using full rotation) with the results from [7], our algorithm outperformed theirs for the 'Albano' and 'Swim' datasets and achieved close results for the 'Fu' dataset.

For the datasets with a higher number of objects ('Poly5b' and 'Shirts'), our algorithm performed noticeably worse. Compared to the algorithm in [7] which uses an allocation algorithm to allocate subsets of objects to different containers (which are then optimised individually), our algorithm aims to solve the problem in a more global fashion by optimising the entire object set, rather than splitting it into subsets. The result is that, in theory, our approach can achieve better results as it is better able to explore the solution space, however the downside is that the solution space quickly becomes very large for an increasing number of objects.

TABLE I: Datasets used with number of objects and given rotation angles.

Dataset	n	Given Rotations
Fu	12	[0,90,180,270]
Albano	24	[0,180]
Swim	48	[0,180]
Poly5b	75	[0,90,180,270]
Shirts	99	[0,180]

The number of combinations for our approach can be calculated as:

$$n_{combinations} = n!(k^n) \tag{3}$$

Where $n!$ is the number of ways to order a list of objects, n, and k^n is the number of heuristic combinations for k heuristics with n objects. As an example, using equation 3, the number of solutions for 'Fu' (n = 12, k = 4) and 'Shirts' (n = 99, k = 4) is approximately $8\text{x}10^{15}$ and $3.7\text{x}10^{215}$ respectively.

Consequently, whether our algorithm achieves good solutions or not will depend heavily on both the quality of the initial solutions in the GA, and on the number of generations the GA is allowed to run for. For example, for the 'Shirts' dataset, the best F score achieved by our algorithm was 0.593, which is better than the average score (0.570) for the same dataset in [7]. However, this was only achieved in one of the runs, with the rest producing much worse results. As such, future work needs to be done to reduce the number of combinations for the problem so the algorithm can consistently achieve good solutions without requiring very long run times or multiple runs.

Regarding computation times, our algorithm took much longer to run than the algorithm in [7]. The authors of [7] provided the run time, averaged over all datasets in 'Nest-MB', as 116 seconds. For the full rotation case, our average run time was 217 minutes. This longer run time can be attributed to the fact that our approach tests many more packing orders and rotation angles for objects when compared to [7]. In [7] the authors keep the order of objects fixed from largest area to smallest and only test a small number of different orientations. Using a fixed order with only heuristic optimisation was tested with our algorithm however it was found to produce poor results. Using a fixed order works well when the object set is pre-allocated and the containers are optimised individually (as in [7]), because for each container, large objects end up at the bottom of the container with the smaller object packed in the gaps. With our approach however, it was found that keeping the order fixed from largest to smallest area resulted in the first few containers all having only large objects (with many gaps between them) and the last container having many small objects (which only occupied a small portion of the container).

VI. Conclusions and Future Work

This paper presented a novel hyper-heuristic GA with 4 placement heuristics for the 2D irregular object multi-container packing problem. Based on the results presented, the algorithm shows promise, being able to outperform results from literature for some test cases, however it suffers from

TABLE II: Comparison of our results to literature results

	Our Results				Results From [7]			
	Given Rotations		Full Rotation (1^o)		Given Rotations		Free rotation	
Dataset	N	F	N	F	N	F	N	F
Fu	4	0.421	4	0.436	4	**0.443**	4	0.440
Albano	3	0.522	3	**0.551**	3	0.480	3	0.510
Swim	5.9	0.334	5.3	**0.404**	5	0.397	5	0.390
Poly5b	8.7	0.371	8	0.439	7	0.478	7	**0.480**
Shirts	9	0.457	8.8	0.520	8	0.518	8	**0.570**

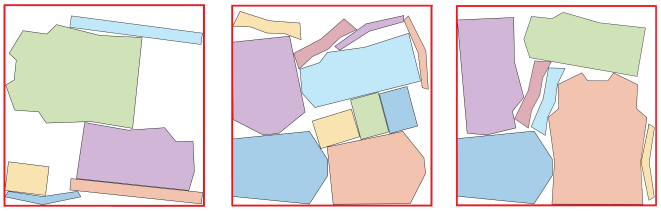

Fig. 1: Example of a packing structure. This was the best packing result for the 'Albano' dataset using full rotation.

long computation times and degradation in quality of results for large object sets. Based on this observation there are two main areas for future work: 1) reduce the number of combinations to the problem so the algorithm is less dependent on the quality of initial solutions/number of generations, and 2) reduce computation times by improving the DigiPac™ algorithm and limiting the number of object rotations.

For Task 1, the algorithm will be adapted to use the same approach as in [7], whereby the object set will be split into subsets which will then be allocated to separate containers before the hyper-heuristic packing algorithm is applied to each container in turn. In doing so, the global search ability of the algorithm would likely be lost, however this is considered an acceptable trade off since the chances of finding the global optimum within such a large solution space is very limited. Additionally, by considering the optimisation of each container individually, the object ordering can also be fixed from largest area to smallest. This would reduce the global search ability of the algorithm even further; however it would also remove the $n!$ term from equation 3, significantly reducing the number of combinations to the problem. Finally, we will also implement additional placement heuristics and then test subsets of these heuristics to see if good results can be obtained with fewer than 4 heuristics. In doing so the k term in equation 3 could be decreased, decreasing the number of combinations further.

For Task 2, the first modification will be to re-write the entire algorithm in C++, utilizing parallel computation to boost the speed further. Additionally, the number of search sites on the lattice grid will also be reduced by only searching for sites near the top of the packing structure. Currently, when searching for a site for each object, DigiPac™ will search over the entire space from the bottom of the container to the highest point of the existing packed structure. This means that small objects can end up being placed in gaps near the bottom of the structure. For 2D applications (such as strip packing for the garment industry), this is desirable. However if the algorithm is to be adapted to 3D for the packing of nuclear waste, this is unsuitable since objects cannot be placed underneath previously packed objects without collision. By limiting the search to only the top part of the packing structure, not only will this make the algorithm more suitable for the planning of packing real-world objects, it will also cut down the number of search sites on the lattice grid, increasing the speed further. The final, and most significant task will be to seek ways to limit the rotation angles for the objects. In [7], the authors considered the edges of the object to pack as well as the edges of the packed objects and container to calculate angles which result in edge alignment between packed objects or the container wall. In [10], the authors used principal component analysis to identify convex features of the object to pack and use this to calculate angles which aim to promote good nesting of the object with the packed pile. Future work will seek to find additional methods like these as well seeing if the methods can be adapted for 3D packing.

REFERENCES

[1] L.J.P. Araújo, A. Panesar, E. Özcan, J. Atkin, M. Baumers, I. Ashcroft, "An experimental analysis of deepest bottom-left-fill packing methods for additive manufacturing", International Journal of Production Research, 58:22, 6917-6933, 2020.

[2] E. Lo Valvo, "Meta-heuristic Algorithms for Nesting Problem of Rectangular Pieces", Procedia Engineering. 183. 291-296 , 2017.

[3] E. López-Camacho, H. Terashima-Marin, P. Ross, G. Ochoa, "A unified hyper-heuristic framework for solving bin packing problems", Expert Systems with Applications, Volume 41, Issue 15, 2014, Pages 6876-6889.

[4] M. Beyaz, T. Dokeroglu, A. Cosar, "Robust hyper-heuristic algorithms for the offline oriented/non-oriented 2D bin packing problems", Applied Soft Computing, Volume 36, 2015, Pages 236-245.

[5] J. Thomas, N.S. Chaudhari, "Design of efficient packing system using genetic algorithm based on hyper heuristic approach, Advances in Engineering Software, Volume 73, 2014, Pages 45-52.

[6] C. Tu, R. Bai, U. Aickelin, Y. Zhang, H. Du, "A deep reinforcement learning hyper-heuristic with feature fusion for online packing problems", Expert Systems with Applications, Volume 230, 2023, 120568.

[7] A. Martinez-Sykora, R. Alvarez-Valdes, J.A. Bennell, R. Ruiz, J.M. Tamarit, "Matheuristics for the irregular bin packing problem with free rotations", European Journal of Operational Research, Volume 258, Issue 2, 2017, Pages 440-455.

[8] DigiPac, structurevision.com, https://www.structurevision.com/digipac.htm (accessed June 14, 2023).

[9] A. Moraglio, R. Poli, "Geometric crossover for the permutation representation". Intelligenza Artificiale. 5. 49-63. 2011.

[10] B. Guo, Z. Liang, Q. Peng, Y. Li, F. Wu, "Irregular Packing Based on Principal Component Analysis Methodology," in IEEE Access, vol. 6, pp. 62675-62686, 2018.