Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Voxel-Based Solution Approaches to the Three-Dimensional Irregular Packing Problem

Carlos Lamas-Fernandez, Julia A. Bennell, Antonio Martinez-Sykora

To cite this article:
Carlos Lamas-Fernandez, Julia A. Bennell, Antonio Martinez-Sykora (2023) Voxel-Based Solution Approaches to the Three-Dimensional Irregular Packing Problem. Operations Research 71(4):1298-1317. https://doi.org/10.1287/opre.2022.2260

https://pubsonline.informs.org/journal/opre

OPERATIONS RESEARCH
Vol. 71, No. 4, July–August 2023, pp. 1298–1317
ISSN 0030-364X (print), ISSN 1526-5463 (online)

Methods

Voxel-Based Solution Approaches to the Three-Dimensional Irregular Packing Problem

Carlos Lamas-Fernandez,[a,*] **Julia A. Bennell,**[b] **Antonio Martinez-Sykora**[a]

[a] Centre for Operational Research, Management Science and Information Systems (CORMSIS), Southampton Business School, University of Southampton, Southampton, SO17 1BJ, United Kingdom; [b] Centre for Decision Research, Leeds University Business School, University of Leeds, Leeds, LS2 9JT, United Kingdom
*Corresponding author

Contact: c.lamas-fernandez@soton.ac.uk, https://orcid.org/0000-0001-5329-7619 (CL-F); j.bennell@leeds.ac.uk, https://orcid.org/0000-0002-5338-2247 (JAB); a.martinez-sykora@soton.ac.uk, https://orcid.org/0000-0002-2435-3113 (AM-S)

Received: December 2, 2019
Revised: May 25, 2021
Accepted: December 14, 2021
Published Online in Articles in Advance: May 4, 2022

Area of Review: Optimization

https://doi.org/10.1287/opre.2022.2260

Abstract. Research on the three-dimensional (3D) packing problem has largely focused on packing boxes for the transportation of goods. As a result, there has been little focus on packing irregular shapes in the operational research literature. New technologies have raised the practical importance of 3D irregular packing problems and the need for efficient solutions. In this work, we address the variant of the problem where the aim is to place a set of 3D irregular items in a container, while minimizing the container height, analogous to the strip packing problem. In order to solve this problem, we need to address two critical components; efficient computation of the geometry and finding high-quality solutions. In this work, we explore the potential of voxels, the 3D equivalent of pixels, as the geometric representation of the irregular items. In this discretised space, we develop a geometric tool that extends the concept of the nofit polygon to the 3D case. This enables us to provide an integer linear programming formulation for this problem that can solve some small instances. For practical size problems, we design metaheuristic optimisation approaches. Because the literature is limited, we introduce new benchmark instances. Some are randomly generated and some represent realistic models from the additive manufacturing area. Our results on the literature benchmark data and on our new instances show that our metaheuristic techniques achieve the best known solutions for a wide variety of problems in practical computation times.

Supplemental Material: The e-companion is available at https://doi.org/10.1287/opre.2022.2260.

Keywords: 3D irregular packing • open dimension problem • voxel • metaheuristics

1. Introduction

The three-dimensional (3D) irregular packing problem consists of the efficient placement of arbitrary 3D items within a designated volume without overlapping. Although 3D packing of regular shapes (boxes) such as the container loading problem and two-dimensional (2D) irregular packing problems have received a lot of attention in the literature, the 3D irregular variants have not. This problem is of great interest to researchers because of the maturity of the field, making this problem the next significant frontier, and also because of the rise in new applications, most notably those motivated by additive layer manufacturing and 3D printing.

Problems in which the space available is scarce so not all the items can be packed are equivalent to the well-known knapsack problem. For example, prioritizing items to be carried for emergency services, such as an ambulance or for humanitarian aid by helicopter.

Alternatively, the space may not be constrained but using more space increases the cost, so the problem is to pack all the items using the least space. One example arises when packaging customer orders for delivery, when the sender needs to decide what is the most suitable box or set of boxes to pack a given set of items. Fueled by e-commerce, the business to customer deliveries are increasing. In this context, efficient packaging naturally plays a financial role, but also a sustainable one (Mangiaracina et al. 2015). In 2019, the worldwide parcel traffic increased to over 21 billion parcels from 12 billion parcels delivered in 2015 (Mazareanu 2020); being able to reduce the waste generated would have an economic benefit and a significant positive environmental impact.

A major industry that benefits from efficient packing algorithms in three dimensions is the additive manufacturing (AM) industry. AM is the process of generating physical objects from computer designs, usually by adding layers of a certain material. Among other techniques, it includes 3D printing. If more than one item is printed in a single print run, a packing software is needed to generate a layout that ensures items do not overlap and that the layout is contained within the print area. AM is a flourishing industry that had an estimated worth of $4 billion in 2014 and is expected to grow to over $21 billion by 2020 (Thompson et al. 2016). The resulting configuration of the printing layouts plays an important role in the estimation of the material cost and production time for additive manufacturing (Baumers et al. 2013). One of the issues being addressed in recent studies is focused on managing the 3D printers efficiently, which involves making decisions about minimizing the extra support required, the height of the build, and other aspects that depend on the materials being used (see Attene (2015) or Griffiths et al. (2019) for examples).

The main reason for not having efficient algorithms available for solving problems that consider 3D irregular items is the lack of geometric tools available in the literature to find good placements and ensure the feasibility of the layouts, that is, the nonoverlapping condition of the items.

In this article, we examine the problem of packing all items, referred to as small items, into a single box-shaped container, referred to as the large object, with a fixed width and length but undetermined height. The small items are of an irregular shape and have a fixed orientation. The objective is to place all the items within the container in nonoverlapping positions, while minimizing the height of the layout. According to the typology in Wäscher et al. (2007), this is an irregular three-dimensional open dimension problem (I3DODP). This problem is also commonly called the strip packing problem.

The earliest papers tackling 3D packing are motivated by additive manufacturing applications and use a polygonal mesh for the item representation. An early example is by Ikonen et al. (1997) who proposes a genetic algorithm to pack arbitrary shaped items. Their algorithm places items according to a certain order, selecting one of a given finite set of orientations and "attachment" points. These points are the points where one item might make contact with another item, and they are provided as part of the input data. This problem is also tackled by Dickinson and Knopf (1998), who propose a constructive algorithm (CA) where each item is packed in a position that is selected with the aim of maximizing the density of the packing.

Egeblad et al. (2007, 2009) also use a polygonal mesh representation in what they call a general purpose algorithm. The heuristic works with an initial layout and explores the solution space by translating items either vertically or horizontally. Items are permitted to lie in overlapping positions during the search. A more specific application is presented in Egeblad et al. (2010), where a combination of preprocessing techniques and bespoke heuristics is proposed to solve a practical knapsack problem appearing in the furniture industry. Also based on polygonal mesh, Liu et al. (2015) develops an efficient constructive algorithm that allows the rotation of items based on a minimal potential energy placement. More recently, Ma et al. (2018) consider a slightly different problem where the container has a fixed size and the objective is to maximize its utilization by selecting and packing a subset of items from a collection. Their approach consists of shrinking items to a small percentage of their original size and placing them in a starting nonoverlapping position; then they define a nonoverlapping space for each item and allow them to grow, move, and rotate within that space until all objects achieve their original size.

A different line of research is the one explored in Stoyan et al. (2005), where they use phi-objects to represent the geometry (Stoyan 1983, Scheithauer et al. 2005, Bennell et al. 2010). Analogous to the two-dimensional case, the phi-objects are mathematical descriptions based on parametric functions. In principle, this approach can accurately represent the irregular item including curved surfaces. The theory is based on a few simple basic shapes called primary phi-objects. These have formal mathematical descriptions, which are combined into parametric functions (called phi-functions) that describe the interaction of the items given a position and orientation. As a result, they can test for overlap efficiently in any orientation. More complex items are constructed through combining primary phi-objects and the associated phi-functions, which form nonlinear constraints in their model. Deriving these models is complex and can be simplified by using quasi-phi-functions, described in Romanova et al. (2018). With this representation, it is possible to formulate most packing problems as nonlinear programs. The strength of this approach relies on their precise analytical description of the objects. However, items may need many primary phi-objects to be represented and the complexity of the resulting models makes them hard to solve. In general, for the instances available in the literature, the model can only find a local optimum and the resulting packing is not very successful compared with other simpler metaheuristic approaches, as we will see in Section 5.1.1.

Phi-objects and polygonal mesh representations both suffer from increasing complexity with the irregularity of the item. For phi-objects, complicated shapes require the combining of many primary objects to represent the final shape, whereas the polygonal mesh approach

needs more faces and vertices with the more features one object has (irregularities, holes, etc.) in order to provide a good representation. Both, the number of primary objects in a phi-object and the number of vertices or faces in a mesh have a direct impact in the computational cost of the packing algorithms using them. To overcome this, some researchers opt for approximating the three-dimensional models by discrete sets of smaller regular shapes. One frequently adopted way of discretizing the shapes is to approximate them by cubes, usually referred to as voxels.

Approximating the small items using voxels is used in Hur et al. (2001), where the authors develop a bottom-left approach for cylinder containers, and use a genetic algorithm to search over the space of packing sequences. Also using voxels, Jia and Williams (2001) propose a simulation-based packing algorithm motivated by particle packing. In their algorithm, particles can randomly move and rotate as long as they do not overlap with each other. In a later work, this algorithm was made more computationally efficient by Byholm et al. (2009). They take advantage of the discretized space to employ some computational tricks with the shape representation. This includes, for example, removing some voxels that are not going to play a role in the final packing result, resulting in reduced computational time. de Korte and Brouwers (2013) also deal with particle packing in their investigation of the packing of spheres, which they approximate by voxels.

One important parameter affecting the decomposition or discretization is the resolution or the size of the basic units used to approximate the item. If it is too large, the fidelity of the approximation will be poor and some geometrical aspects can be lost. For example, holes or concavities smaller than a voxel cannot be represented. On the other hand, if the resolution is small, giving high fidelity, the model will require a large amount of computer memory and the packing algorithms will be very slow because of more operations being performed. Some approaches (Cagan et al. 1998, Edelkamp and Wichern 2015) use tree structures as a way to overcome this problem. The tree structure has very coarse representations at the top but can be made finer later in the tree if the features require it. If the basic volume units used are cubes, these trees are called octrees and each cube is divided in eight identical smaller cubes at each level.

In Edelkamp and Wichern (2015), they approximate shapes by spheres that are organised in a tree structure. This representation is then used in a simulated annealing algorithm that finds 3D printing layouts of irregular objects. The algorithm takes advantage of the use of spheres to allow free rotation of items. Cagan et al. (1998) use rectangular solids (not necessarily cubes) for the decomposition and develops a simulated annealing solution method based on simple movements and rotations in order to find efficient component layout for various industrial designs (e.g., a car engine or a heat pump).

Voxelization and octrees are used in a variety of applications because of their power for dealing with complicated shapes and are an active topic of research in their own right (Schwarz and Seidel 2010, Baert et al. 2014), including closely related variations, such as chain codes (Sánchez-Cruz et al. 2014, Lemus et al. 2015).

Our chosen approach is discussed in Section 2, where we introduce the problem formally, describe our geometric approach, and introduce an integer linear programming (ILP) model. Because this model is too complex for practical use, we also investigate metaheuristic approaches. In Section 3, we provide the building blocks for developing metaheuristic algorithms, including a constructive algorithm and different local search neighborhoods. These neighborhoods explore two different problem representations, one based on an item sequence that is decoded by different placement rules and one based on the item position in the layout. Based on these building blocks, we develop three different packing algorithms that are presented in Section 4.

Along with the lack of previously published work on the problem, there is also a lack of benchmark data. We have gathered together the main data sets found in the literature and added instances found in online 3D printing data sets. Furthermore, we developed an instance generator to provide a wide range of data sets with specific characteristics. The generator and the benchmark instances are described in Section 5 along with a full discussion of the results of our computational experiments.

The contributions of this work are the following. We explore the potential of voxels to represent irregular items for 3D packing problems. We develop effective solution methods for the I3DODP that can solve large problems. We address the lack of data by providing a rich set of benchmark data sets and an instance generator. In adopting a voxel representation of the items to be packed, we develop a new geometric tool analogous to the nofit polygon (NFP) for two-dimensional irregular packing, called the nofit voxel. Finally, we provide a new mathematical programming formulation for the I3DODP and test its capabilities.

2. Voxelised Three-Dimensional Packing

In this section, we introduce formally the optimization problem we are solving, based on a discrete voxel space. We also discuss geometric considerations and introduce tools for handling the nonoverlap constraints.

2.1. Problem Description

We have a set of n small items $\mathcal{I} = \{p_1, p_2, \ldots, p_n\}$ that need to be placed inside a large object $\mathcal{C}$ (the container).

The container has a rectangular base and a variable height (z-coordinate), and our objective is to find a set of n placements $L = \{l_1, l_2, \ldots, l_n\} \subset \mathbb{Z}^3$ for all the items in $\mathcal{I}$ ensuring that no two items overlap and that all items lay within the container, while minimizing the container height. The orientation of the items is fixed.

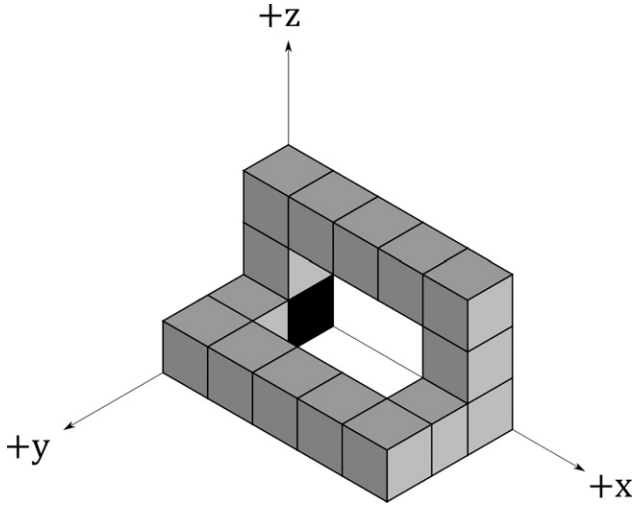
2.2. Voxel Representation

In order to represent the irregular items from $\mathcal{I}$, we divide their bounding box into identical small cubes, which are called voxels, and use a binary code to distinguish which ones are part of the item and which ones are not. Having a rectangular shape (bounding box) as a support enables us to encode this information in a structure analogous to a binary matrix. For each item, the voxel in the bottom-left-back corner of its bounding box is called the reference voxel. To locate the items in the container, it is sufficient to determine the position of the reference voxels. In Figure 1, we show an example of an item and its reference voxel. We also consider the bottom-left-back corner of the container as the origin of the coordinate system, so all the points inside the container have positive coordinates.

2.3. Constraint Handling

There are two main constraints for the strip packing problem; the containment of small items within the boundary of the container (containment constraints) and no overlap between small items (no-overlap constraints). In this case, we also add the constraint of fixed orientation of the items. Because the containers we use are rectangular, the containment constraints are trivial. The reference voxel of each item has a maximum permitted x and y position, which corresponds to the size of the container in the x or y dimension minus the magnitude of the item along that dimension. For convenience, we set a maximum value for z that is large enough to easily contain all the small items. We denote the positions that lay inside the container for the item $p_i \in \mathcal{I}$ as the set $\mathcal{IFV}_{p_i,\mathcal{C}}$, the inner-fit voxel of the item p_i and the container $\mathcal{C}$. Note that this is a similar concept to the inner-fit rectangle in two dimensions, see Gomes and Oliveira (2006) for details.

Figure 1. Example of an Irregular Item Represented by Voxels with Its Reference Voxel Highlighted (Solid Black)



Nonoverlapping constraints between items are, however, more difficult to model, as both of their shapes are arbitrary. If we look at the two-dimensional literature, the nofit polygon (NFP) is a concept that has been consistently used in 2D irregular packing problems since it was first developed in the sixties by Art (1966) who described it as the envelope. Informally, the NFP is derived from two polygons and represents the relative positions of these polygons in terms of whether they overlap, represented by the interior of the NFP, touch, represented by the boundary of the NFP, or are separated, represented by the outside of the NFP. A more in-depth definition and a review of various algorithms for calculating the NFP can be found in Bennell and Oliveira (2008). It is worth noting that NFPs have a close relationship with Minkowski sums. For two arbitrary polygons A and B, it can be shown that $A \oplus -B = \{a - b : a \in A, b \in B\} = NFP_{AB}$ (see, for example, Milenkovic et al. (1992) or Bennell et al. (2000)).

To test the overlap between items, we define the nofit voxel (NFV), which is analogous to the union of the NFP and its interior. In general, dividing items into discrete cubes allows us to perform very quick intersection tests to identify overlap between items. Before formally defining the NFV, we first describe the simple overlap test for items represented by voxels.

Given two items $p, q \in \mathcal{I}$, with their reference voxels located at $l_p = (p_x, p_y, p_z)$ and $l_q = (q_x, q_y, q_z)$, respectively, we say that they are in a nonoverlapping position if either:

1. Their bounding boxes do not intersect.
2. In the intersection of their bounding boxes, no two voxels from the items coincide in the same position. For a binary matrix where an entry is positive if it is a voxel of an item, this is equivalent of saying no two positive values of their binary matrices are in the same position.

The second condition can be simply tested by checking the corresponding elements of the matrices that represent the voxels of the two items and stopping at the first coincidence. Testing overlap in this order leads to an efficient implementation as, unless two items are very close in the layout, the first condition, which is very simple to test, will be sufficient to conclude that there is no overlap between them.

Building on this idea, it is trivial to see that if l_p and l_q lead to feasible positions for p and q, then for any $\alpha = (\alpha_x, \alpha_y, \alpha_z) \in \mathbb{Z}^3$, the points $l_p + \alpha = (p_x + \alpha_x, p_y + \alpha_y, p_z + \alpha_z)$ and $l_q + \alpha = (q_x + \alpha_x, q_y + \alpha_y, q_z + \alpha_z)$ also lead to feasible positions for p and q.

In other words, a nonoverlapping position is maintained as long as the relative position of the items does not change. Following this idea, we define the NFV for two items p and q, which are represented by voxels, as a set of the overlapping relative positions of the items p and q. More formally,

Definition 1. The *Nofit voxel* of p and q is a set $\mathcal{NFV}_{p,q} \in \mathbb{Z}^3$ with the property that, if $l_p = (0,0,0)$ and $l_q \in \mathcal{NFV}_{p,q}$, p and q intersect.

In Figure 2, we illustrate the nofit voxel of two irregular items.

If the NFV of two items is known, it can be used to determine if two items overlap by testing whether the reference point of q is inside the NFV. However, in Definition 1, we require that the location of p is fixed at the origin. Let us now consider the case where p is located at an arbitrary point l'_p and q is located at l'_q. We can still use the NFV to test for overlap. In this case, they intersect only if $l'_p - l'_q \in \mathcal{NFV}_{p,q}$. This test is more efficient than the simple overlap test described above. Because this is a recurrent test, during the remainder of the paper, we introduce the simplified notation $\mathcal{NFV}_{p,q}(p_x, p_y, p_z)$ for it, defined in Equation (1).

$$(q_x, q_y, q_z) \in \mathcal{NFV}_{p,q}(p_x, p_y, p_z) \Longleftrightarrow (q_x - p_x, q_y - p_y, q_z - p_z) \in \mathcal{NFV}_{p,q} \tag{1}$$

To compute the NFV of two items p and q, $\mathcal{NFV}_{p,q}$, we can simply keep p fixed at the origin and perform an overlap test between p and q in all of their possible overlapping positions (i.e., the positions where the bounding boxes of p and q overlap). When a test is positive, the point is added to the $\mathcal{NFV}_{p,q}$ and the process continues until all points are tested.

Note that, if $\mathcal{NFV}_{p,q}$ is known, it is trivial to calculate $\mathcal{NFV}_{q,p}$ using the following property:

$$\mathcal{NFV}_{q,p}(i,j,k) = \{-a : a \in \mathcal{NFV}_{p,q}(i,j,k)\}. \tag{2}$$

Thanks to the property in Equation (2), the calculation of NFVs only needs to be done once for each pair of item types. Once the NFVs are available, the overlap checks are merely a matter of checking if the reference voxel is in a set, provided the orientation of the items remains unchanged.

2.4. ILP Formulation

In this section, we describe an integer linear programming model for the problem stated in Section 2.1. The model uses binary variables to determine the reference voxel of the items and defines constraints based on the NFV information to ensure nonoverlapping.

For each item p, we define a region, $\mathcal{B}(p) \subseteq \mathcal{IFV}_{p,\mathcal{C}}$ where it can be placed. This region is a discrete set of voxels that are identified in the model by the following binary variables:

$$x_{pijk} = \begin{cases} 1, & \text{if } p \text{ is placed in } (i,j,k) \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

We mean placed in the sense that, if x_{pijk} is set to one, the reference voxel of the item p is located at (i, j, k). As mentioned earlier, the reference voxel is chosen to be the corner voxel of the item's bounding box with smallest coordinates.

The full model is as follows:

$$\text{minimize } H, \tag{4}$$

subject to

$$\sum_{(i,j,k)\in\mathcal{B}(p)} x_{pijk}(h_p + k) \le H \qquad \forall p \in \mathcal{I}, \tag{5}$$

$$\sum_{(i,j,k)\in\mathcal{B}(p)} x_{pijk} = 1 \qquad \forall p \in \mathcal{I}, \tag{6}$$

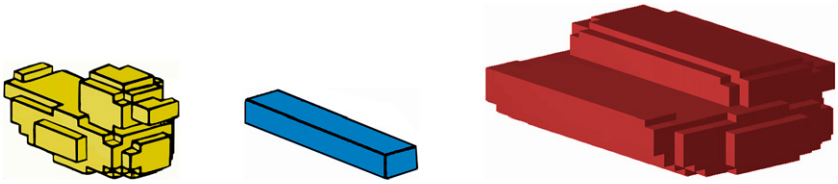
$$x_{pijk} + \sum_{\substack{(l,m,n)\in \\ \mathcal{NFV}_{p,q}(i,j,k)\cap\mathcal{B}(q)}} x_{qlmn} \le 1 \qquad \forall p, q \in \mathcal{I}, \; p \ne q, \; \forall (i,j,k) \in \mathcal{B}(p), \tag{7}$$

$$H \in \mathbb{N}, \tag{8}$$

$$x_{pijk} \in \{0,1\} \qquad \forall p, \in \mathcal{I}, \; \forall (i,j,k) \in \mathcal{B}(p), \tag{9}$$

where h_p is the height of item p. The first Constraint (5) sets the value of the variable H, which is the total height of the container in voxels. It is defined as being higher than the highest point of any of the placed pieces (which is their reference voxel position plus their height). In Equation (6), we make sure that all items are placed. The sum of the placement variables for each item p is restricted to the set $\mathcal{B}(p)$; thus, this

Figure 2. (Color online) Two Arbitrary Items, p and q and Their Nofit Voxel $\mathcal{NFV}_{p,q}$



constraint also guarantees containment. Finally, Equation (7) uses the NFV of two items to ensure that there is no overlap between each pair of items. There is a constraint for each position (i,j,k) where p can be placed. Then for all other items, q, we constrain the sum of the binary variables of the position of p with the binary variables for the position of q that belong to the $\mathcal{NFV}_{p,q}(i,j,k)$. In principle, these would be the points of $\mathcal{NFV}_{p,q}(i,j,k)$; but we concern ourselves only with the ones that are also part of $\mathcal{B}(q)$, which also ensures containment. Constraint (8) restricts the domain of H to natural numbers, and Constraints (9) defines the placement variables as binary.

This model is an extension of the Dotted Board Model from Toledo et al. (2013) for the two-dimensional problem; to the best of our knowledge, it is the first integer linear programming model available for the discretized I3DODP.

If we define the regions $\mathcal{B}(p)$ to be $\mathcal{B}(p) = \mathcal{IFV}_{p,\mathcal{C}}$ for all items, the optimal solution of the model would be the optimal solution of the discretized packing problem. Unfortunately, solving such a model requires a very high computational effort, which makes solving reasonable sized instances impractical, as we will show in Section 5.2, where we provide some computational experiments evaluating the capabilities of the model.

However, we can reduce the number of variables by constraining the placement options for each item. Given a starting layout, bounds on the number of voxels an item can move away from its current position provide an easier-to-solve model. This is a common strategy known as compaction and separation, which has been used successfully in the past in 2D irregular packing literature in conjunction with metaheuristics (see, for example, Bennell and Dowsland 2001 or Gomes and Oliveira 2006). Models that use phi-functions follow a similar approach, see, for example, Stoyan et al. (2005).

In order to use this approach to compact the layout, the model needs to be given an initial solution and a region $\mathcal{B}(p)$ for each item p that bounds the movement, which is substantially smaller than $\mathcal{IFV}_{p,\mathcal{C}}$. Given $\mathcal{B}(p)$, the next step is to solve Model (4)–(9) to optimality. If the height is reduced, the procedure has been successful. At this point, new $\mathcal{B}(p)$ regions can be defined again, with the aim to further reduce the overall height (H).

As a separation procedure, the initial solution must be infeasible (contain overlap). In this case, the height of the container is fixed at H thus removing the constraints in (5). The regions $\mathcal{B}(p)$ are defined to respect the fixed height and the objective function is to remove overlap and find a feasible solution. This separation procedure is used as one of our neighborhoods in the variable neighborhood search (VNS) presented in Section 4.3.

3. Building Blocks of the Three-Dimensional Packing Heuristics

In this section, we examine a number of components that are the building blocks for the solution algorithms we design to solve the I3DODP. We first describe a constructive procedure to generate initial solutions. Because the procedure represents the solution as a sequence of items, and determines an item's position using placement rules, we are able to use sequence-based neighborhoods within a search heuristic. Given an initial solution, an alternative approach is to define neighborhoods that work with a solution representation that uses the position of the items in a layout. This approach usually requires a relaxation of the overlap constraint or containment constraint. For this case, we describe neighborhoods that work on the layout of complete solutions, where a strategic oscillation technique and a new objective function related to overlap guide the search.

The two approaches that we investigate align with the strategies presented in Bennell and Oliveira (2009) for 2D irregular packing, working with sequence representations and with layout representations.

3.1. Constructive Algorithm

We propose a constructive algorithm based on a bottom-left-back strategy. The idea is to place the items one at a time, ensuring their placement minimizes first the z coordinate, followed by the x and then the y coordinates. For a packing problem with a collection of items $\mathcal{I} = \{p_1, p_2, \ldots, p_n\}$ and a container $\mathcal{C}$, let $S = \{s_1, s_2, \ldots, s_n\}$ be an ordered sequence of the items to be placed. This sequence could be a random permutation of $\mathcal{I}$, or a specific sorting of the items, for example by decreasing volume.

The algorithm starts by placing the first item in the sequence, s_1, at location $l_1 = (0,0,0)$. This placement is always feasible because of our definition of the reference points and the coordinate system of the container. Then, the next item in the sequence, s_i, is placed in the bottom left back position $l_i = (x,y,z)$ such that the item lays within the container, that is, $l_i \in \mathcal{IFV}_{s_i,\mathcal{C}}$ and it does not overlap with any other placed item, that is, $l_i \notin \mathcal{NFV}_{s_j,s_i}(l_j),\ \forall j < i$. To find position l_i, the algorithm tests points from $\mathcal{IFV}_{s_i,\mathcal{C}}$ until it reaches a valid placement. If an item of the same type has been placed earlier, the search starts from the last valid placement to avoid retesting the same points of $\mathcal{IFV}_{s_i,\mathcal{C}}$ previously found to be infeasible.

The points are evaluated starting from the lowest z values, followed by the lowest x and the lowest y, so the first valid position found is the one that lays at the bottom-left-back-most possible position. This step can be modified easily if the x or y position are sought in reverse order (i.e., from highest value to lowest); in

that case, the algorithm rule would be called right instead of left or front instead of back. In fact, each item in the sequence can be packed using a different rule. To acknowledge this, we introduce the notation $R = \{r_i\}$ for a set of rules, where r_i represents one of the following: left-back, left-front, right-back, or right-front. A solution of the algorithm is then given by $L = CA(S, R)$, where items have been placed in the order determined for the sequence S, in the bottom-most possible position and according to the corresponding rule from R.

Although the algorithm is deterministic, different sequences S and different placement rules R can result in different packing layouts. This opens the possibility to search over the sequence of items to be placed. We explore this possibility in the next section with the sequence-based neighborhoods.

3.2. Sequence-Based Neighborhoods

Searching over the sequence of items has been a common topic of research in 2D irregular packing problems; see, for example, Gomes and Oliveira (2002), who implement a two-exchange (swap) heuristic to search over an item sequence. Dowsland et al. (1998) and Abeysooriya et al. (2018) use a technique called Jostle that constructs solutions iteratively and extracts the placement sequence from the layout generated on the previous step.

In this work, we identify two types of sequence-based neighborhoods, the sequence swap neighborhood, and the rule change neighborhood. In Section 4.1, we describe a procedure to exploit them in an iterated local search algorithm.

The sequence swap neighborhood consists of all the solutions obtained by swapping two items of a different type in the sequence S. If $L = CA(S, R)$ and $S = \{s_1, s_i, \ldots, s_j, \ldots, s_n\}$, $i < j$

$$N_S(L) = \{\{CA(S', R)\} : S' = \{\ldots, s_j, \ldots, s_i, \ldots\}\ \forall s_i, s_j \in S, s_i \neq s_j, i < j\}. \quad (10)$$

The rule change neighborhood consists in changing of one rule for one item in the packing sequence. More formally, if $L = CA(S, R)$, then the neighborhood is defined by

$$N_R(L) = \{\{CA(S, R')\} : R' = \{\ldots, r_{i-1}, r_i', r_{i+1}, \ldots\} : r_i' \in R \setminus r_i\} \quad (11)$$

Each rule is effectively creating a different construction heuristic, and this space of solutions can be searched algorithmically. This kind of search is sometimes referred to as a hyperheuristic; see, for example, Burke et al. (2010) and Terashima-Marín et al. (2010) for examples in 2D irregular packing using a polygonal representation or Mundim et al. (2018) for a framework using a discretized approach.

3.3. Layout-Based Neighborhoods

A layout-based neighborhood of a solution L is a set of solutions $N(L)$ that are reachable from L by applying a perturbation or move. This movement is no longer in the sequence or in the set of rules but a change of position of one or more items in the packing layout. As opposed to the sequence-based neighborhoods, the solutions in these neighborhoods might not always be feasible, as some moves might introduce overlap between items.

We distinguish two types of layout-based neighborhoods. The first type involves moving an item to a nearby position and is called *single item neighborhoods*. The second type involves more dramatic changes, such as swapping the positions of two items in the layout and is called *full solution neighborhoods*. The reason for this distinction is that the single items neighborhoods are usually quicker to evaluate and provide solutions of similar quality to the current one because only one item is moved. However, the full solution neighborhoods are necessary to create kicks in the solution that help to escape local optima.

3.3.1. Axis Aligned Direction Neighborhood. The axis aligned neighborhood of an item p, $N_A(L, p, \delta)$ includes all the solutions of the form $L = \{l_1, l_2, \ldots l_n\}$ where the reference point of the item p, l_p, is substituted by another point from $\mathcal{IFV}_{p,\mathcal{C}}$ that is reachable by translating l_p by a fixed amount of voxels between zero and δ, in an axis aligned direction. More formally,

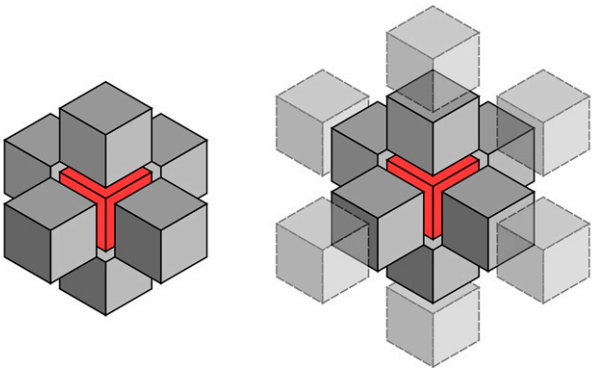
$$N_A(L, p, \delta) = \left\{\{l_1, \ldots, l_{p-1}, l_p', l_{p+1}, \ldots\} : l_p' \in \bigcup_{d=-\delta}^{\delta} \bigcup_{i=1}^{3} \{l_p + d\boldsymbol{e_i}\} \bigcap \mathcal{IFV}_{p,\mathcal{C}}\right\}, \quad (12)$$

where $\boldsymbol{e_i}$ denotes the vectors of the standard basis in $\mathbb{R}^3$. The parameter δ determines the number of voxels the reference point can be displaced. The special case where we consider all the points within the container in any axis aligned direction is denoted by $\delta = \delta_{max}$. In this case, the neighborhood is similar to the translation neighborhood proposed by Egeblad et al. (2007, 2009) for polyhedral items. The axis aligned direction neighborhood is illustrated in Figure 3.

Note that the neighbors of a feasible solution might be infeasible with respect to the overlap constraint, but they will not violate the containment constraint, as we impose the condition that items can only move within their inner-fit voxel.

3.3.2. Enclosing Cube Neighborhood. The enclosing cube neighborhood contains all the valid points in a

Figure 3. (Color online) Neighborhood of Axis Aligned Directions, $\delta = 1$ (Left) and $\delta > 1$ (Right)



Note. The point in the centre is the original reference point l_p, and the surrounding points are the possible reference points in the neighborhood.

cube centred at the reference point and with sides of $2\delta + 1$ voxels, as shown in Figure 4. More formally,

$$N_C(L,p,\delta) = \left\{ \{l_1,\dots,l_{p-1},l'_p,l_{p+1},\dots\} : l'_p \in \bigcup_{d\in\{-\delta,\dots,\delta\}^3} \{l_p + d\} \bigcap \mathcal{IFV}_{p,\mathcal{C}} \right\}. \quad (13)$$

For a large δ, this is equivalent to the complete inner-fit voxel of the item. In this case, the neighborhood can be seen as an insert move. We denote this situation by $\delta = \delta_{max}$.

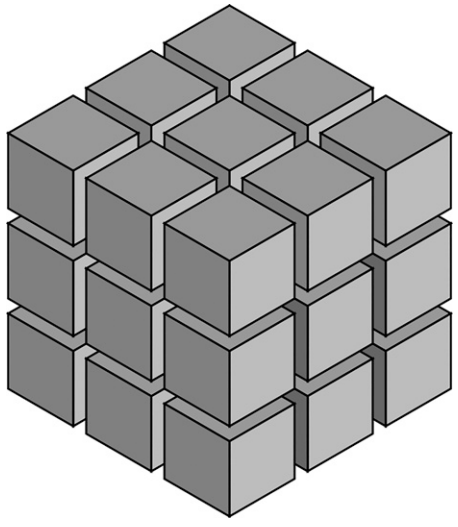
3.3.3. Item Swap Neighborhood. The swap neighborhood of an item is a full solution neighborhood. It represents all the possible swaps the item can make with other items. Some of the swaps might violate the containment constraint, because the swapped items could end up with parts outside the container. If this happens, the item is moved inwards along the axis (one voxel at a time) until the point is in its inner-fit voxel. This position, denoted by $l_p^{(q)}$ (the resulting position of p after swapping with q), is defined as follows,

$$l_p^{(q)} = \underset{l'\in\mathcal{IFV}_{p,\mathcal{C}}}{\arg\min} \|l_q - l'\|_1 \quad \forall p, q \in \mathcal{I}; p \neq q. \quad (14)$$

Then, the neighborhood contains all the possible swaps of a item p with others,

$$N_{swap}(L,p) = \{\{\dots,l_{p-1},l_q^{(p)},l_{p+1},\dots,l_{q-1},l_p^{(q)},l_{q+1},\dots\} : q \in \mathcal{I}, q \neq p\}. \quad (15)$$

Figure 4. Neighborhood of Enclosing Cube, $\delta = 1$



Note. The visible points are the reference points in the neighbors, whereas the original reference point is located in the centre of the cube.

3.4. Strategic Oscillation

A search algorithm using a layout-based neighborhoods is likely to produce solutions that contain overlap. Although allowing overlap helps navigate the solution space, it makes the problem effectively a bi-objective problem, because the overlap constraint is relaxed and added to the objective function. Solutions containing overlap are not feasible with respect to the original problem so we need to produce solutions with zero overlap without compromising on solution quality. Strategic oscillation (Glover and Marti 2013), has been used extensively in the two- and three-dimensional irregular packing literature; see, for example, Bennell and Dowsland (1999), Umetani et al. (2009), Imamichi et al. (2009), or Egeblad et al. (2009) to navigate this problem. The idea is that the algorithm oscillates between focusing primarily on completely removing the overlap and then primarily on reducing the height of the layout. Certain criteria allow the switch in focus during the course of the algorithm.

In each iteration of the strategic oscillation the algorithm works on an overlapping layout with fixed height, and tries to remove overlap. Once this has been achieved or the maximum number of local search iterations is reached, the height is modified as follows. If the solution still has overlap, the height will increase by one voxel and the algorithm will try to resolve the overlap with more space. If it succeeds, the height is reduced by a certain percentage (controlled by the parameter `dec_p` in our computational experiments) and the algorithm is run again, with the aim of finding nonoverlapping positions at a lower height.

Because the reduction of height leaves some items protruding above the top of the container, their position needs to be adjusted. We have implemented two strategies to achieve this: *move down* and *random reinsert*. With the *move down* strategy, protruding items

are simply moved downward the minimum amount until they fully fit inside the new container. Instead, with the *random reinsert* protruding items are removed from the layout and inserted in a random position of their inner-fit voxel. In the computational experiments, the strategy is selected by means of the algorithm parameter `SO_mode`.

3.5. Objective Function

While our objective is to minimize the height of the final layout, if we search over the layout and use strategic oscillation, intermediate solutions will have overlap. For this case, we propose an objective function whose main goal is to understand when a solution is better than another, in terms of leading to a nonoverlapping solution. In the nesting literature, typical objective functions include measuring the penetration depth of a pair of items (Bennell and Dowsland 1999, Imamichi et al. 2009, Umetani et al. 2009) or the exact amount of overlap (area in 2D or volume 3D) for each pair of items (Egeblad et al. 2009), among others. Our objective function is calculated based on the volume of the overlap of the bounding boxes,

$$F(\mathcal{I}, l_1, l_2, \ldots, l_n) = 1 - \frac{1}{\binom{n}{2}} \sum_{i,j \in \mathcal{I}, , i<j} \frac{Vol(B(i,l_i) \cap B(j,l_j))}{V_{\max}}, \quad (16)$$

where $B(i,l_i)$ is the bounding box of item i located in position l_i and $Vol()$ is a function that returns the voxel volume (number of voxels) of its argument. The constant $V_{\max} = \max_{i \in \mathcal{I}} Vol(B(i))$ is the maximum volume of any bounding box present in the instance and is used to scale the function value between zero and one. Dividing the sum by $(n/2)$, the number of pairs of items in the solution is used to average the overlap of the items. This function is very quick to evaluate and provides an intuitive idea of how difficult it is to separate a pair of overlapping items.

4. Search Algorithms

Using the building blocks described in the last section, we propose three different metaheuristic algorithms. The first is an iterated local search, which is based on searching over the item sequence and generating the layout by applying the constructive algorithm. The next two approaches are a tabu search (TS) and a variable neighborhood search, which are based on searching over the layouts of complete solutions.

4.1. Iterated Local Search

Iterated local search is a metaheuristic algorithm that iteratively searches neighborhoods, accepting improving moves, to find local optima. At each local optima, the search moves to a different part of the solution space by performing a kick (Lourenço et al. 2010).

We implement this algorithm to take advantage of the different rules that can be used in our constructive algorithm. The initial solution starts with a sequence of items sorted in order of decreasing height and the solution is constructed using the left-back placement rules to place all of the items. For the local search, we use the rule change neighborhood, $N_R(L)$, that involves changing one of the two placement rules (left/right or back/front) of an item and repacks using the constructive algorithm.

The kicks consist of performing a number of position swaps in the item order sequence (and their placement rules) at random. The number of swaps to be performed is controlled by an algorithm parameter, $\mathtt{n_{swaps}}$, which is determined in Section 5.3.

Although searching over the sequence can produce quick, efficient layouts, the solution space does not contain all possible solutions and some good layouts might be unreachable using these techniques. To overcome this, we propose two other algorithms that work on the layout of full solutions.

4.2. Tabu Search

Tabu search, introduced by Glover (1989), is one of the first metaheuristic and has been successfully applied to various combinatorial optimization problems, including irregular packing (Błazewicz et al. 1993, Bennell and Dowsland 1999). It uses neighborhoods to search the solution space by testing all solutions in the neighborhood of the current solution and moving to the best neighbor, whether it improves the objective function value or not. The solution resulting from the move is added to a tabu list so it cannot be revisited for a certain number of moves. In practice, an attribute of the move is stored as this is more efficient and achieves the desired outcome.

TS lends itself to the layout representation because the voxel neighborhoods are finite and can be evaluated in full with a low computational effort. Maintaining a tabu list is an effective way to prevent items moving back and forth to the same positions. Our TS works within the strategic oscillation framework searching for nonoverlapping solutions between each adjustment of the height constraint. Each height oscillation occurs after a predefined maximum number of accepted neighborhood moves or if a zero overlap solution is found by the TS.

The initial solution, L_0, is produced using the constructive algorithm (Section 3.1) as $L_0 = CA(S_0, R_0)$, where S_0 is the sequence of items from the instance ordered by decreasing volume and R_0 is the set of rules that assigns left-back to all the items. In the first step, the algorithm oscillates to create some overlap, as described in Section 3.4. In each iteration, the algorithm explores the axis aligned neighborhood of the solution, with parameter $\delta = \mathtt{d\_axis}$, whose value is determined in the computational experiments. When

a movement is performed, the opposite direction of the movement for that item is added to the tabu list to avoid the same item moving backward. The length of the tabu list, `TL` is also a parameter of the algorithm.

Note that to evaluate each move it is not necessary to calculate the objective function from Equation (16). Instead, it is sufficient to perform an update to evaluate the effect of the change in the solution quality. This test is very quick to perform, because it only involves recalculating the overlap terms for the item that has moved. The efficiency of this step allows for the testing of a large number of movements in a reasonable computational time.

To control the height, the tabu search will run on a given solution for a number movements (determined by parameter `nMoves`) or until it completely removes the overlap and then perform the oscillation.

4.3. Variable Neighborhood Search

Having designed a number of neighborhoods, it is natural to consider implementing a variable neighborhood search algorithm. This metaheuristic, introduced by Mladenović and Hansen (1997), makes use of a list of neighborhoods that are explored in order and works on the assumption that a local optimum for a neighborhood might not be a local optimum in an alternative neighborhood. VNS is implemented with strategic oscillation in the same way as TS.

The algorithm starts from an initial solution (created in the same way as in the previous section) that contains some overlap and searches the first neighborhood in the list. In our implementation, we apply a *steepest descent* rule; therefore, all the solutions in the neighborhoods are evaluated before moving to the best improving solution.

If there are no improving neighbors, the incumbent solution is labelled as a local optima for the current neighborhood. Then, VNS selects the next neighborhood in the list and explores it until it is again stuck in a local optima. If an improvement is found, the incumbent solution moves to it and VNS returns to the first neighborhood and continues the search.

When a solution is a local optima for all the neighborhoods, the algorithm applies a kick (or a shake) and returns to the first neighborhood in the sequence. After a certain number of kicks (parameter `nK`), the algorithm performs strategic oscillation in order to increase or decrease the maximum allowed height and the process is repeated.

We have implemented VNS with three neighborhoods that are searched in this order:

- Enclosing cube neighborhood, N_C with $\delta =$ `d_cube`
- Axis aligned neighborhood, N_A with $\delta =$ `d_axis`
- ILP model, with $B_p = N_C$ and a small δ value

The first two neighborhoods are explored for all the overlapping items. We have chosen to use the cube neighborhood with a small delta first, as it can provide very similar solutions by moving only one item by one voxel. The axis aligned neighborhood has the capacity to reach further away points and can often provide more dissimilar solutions, especially at early stages of the algorithm. Our experiments found this order of the first two neighborhood definitions to be more efficient because it finds the best local optima through comprehensively searching a small local neighborhood of the current solution before searching more widely. When the second neighborhood finds a better solution, the search returns to the focused optimization using the first neighborhood. If these neighborhoods cannot improve the solution, we run the ILP Model (3)–(9). The δ values used for the cube neighborhoods of the model are a parameter of the algorithm. We use a different parameter for the items that are overlapping ($\delta =$ `Ov_d`) and for the items that are not overlapping ($\delta =$ `NOv_d`). Both of these values are determined in Section 5.3. To avoid spending excessive computational time running the model, and taking into account that is only practical when using only small δ values, we have restricted its use to only once per strategic oscillation and only in layouts where there is at most one pair of overlapping items. The running time of the model is restricted to a maximum of 300 seconds.

Once we have hit a local optimum for all the neighborhoods and before oscillating, we disrupt the solution with the aim of exploring a different part of the solution space. This disruption, also called a kick, is performed in two steps: a shaking of the layout and an item swap.

To perform shaking, we move all overlapping items to a random location within a neighborhood N_C. The number of voxels, δ, that defines the size of the neighborhood is set such that it increases linearly with the number of disruptions used so far in the search. This technique allows us to have smaller disruptions in the early stages in order to guide the algorithm quickly to good solutions. However, in later stages of the algorithm, the disruptions are larger, allowing the search to escape from local optima. The parameter will vary between a minimum value of δ_{min}, which is the size of the starting neighborhood, and a maximum value of δ_{max}. The actual parameter used in the iteration is calculated as $\delta = max(\delta_{min}, \lfloor \rho(\delta_{max}+1) \rfloor)$, where ρ is the ratio between the kicks performed and the maximum kicks allowed for the current height. Suitable values for δ_{min} and δ_{max} are explored and selected in the computational experiments in Section 5.3.

The second step of the kick consists of swapping two overlapping items. To avoid the costly procedure of testing all possible swaps, we limit the amount of tests. We test the swap of each item in the layout with a number of items from the overlap list, determined by the parameter `test_swaps`. The overlapping items are chosen at

random and avoiding items of the same type. In total, we test a number between one and $n \times$ `test_swaps` swaps and perform the one that returns the best objective function, even if it does not improve the incumbent solution.

These two steps complement each other. The shaking part creates many overlapping positions and enables the search to explore movements of items that might otherwise stay in nonoverlapping positions and prevent the solution improving. The swap part produces a larger change in the layout, which might be difficult to obtain with moving items one at a time. Note that creating many overlaps will create a greater opportunity for swapping items to succeed than if we apply these two steps in an opposite order.

Each iteration of the VNS consists of a number of local searches followed by a kick. Therefore, in each iteration, we will find many local optima, possibly all of them containing overlap, for the same container height. Because the kick involves swapping one or more pairs of items, there is a risk that these swaps might not be beneficial in the long term. To avoid this situation, the kick is not always applied to the incumbent solution but to the best of the local optima found for the current height. By doing this, item swaps that do not lead to an improvement are not carried forward in the search.

5. Computational Experiments

In this section, we evaluate the performance of the proposed voxel-based algorithms. In Section 5.1, we describe the instances used for testing. In Section 5.2, we explore the limits of the ILP model to solve instances to optimality. In Section 5.3, we describe how we have performed the parameter configuration for the metaheuristic algorithms, along with the best configuration for each algorithm. Finally, in Section 5.4, we compare the performance between the algorithms and also against the state of the art in the literature.

5.1. Test Instances

We test our algorithms across a range of different instances to compare their performance in different scenarios. We found that in the 3D literature, researchers mainly use simple shapes, such as convex polyhedra (Pankratov et al. 2015, Stoyan et al. 2005), or nonconvex shapes that are defined by very few vertices (Stoyan et al. 2004, Egeblad et al. 2009, Liu et al. 2015). In our experiments, we solve some of these instances and adapt one popular instance from the two-dimensional literature with similar properties, *shapes0* from Oliveira et al. (2000). In order to test our algorithms more thoroughly, we also propose a set of randomly generated instances, called *blobs*, and two examples of complex realistic instances. Unlike standard academic instances, the *blobs* data sets are designed to include more challenging shapes, such as "smooth" surfaces or holes.

In Table 1, we summarise some of the key differences of these instances

5.1.1. Instances from the Literature.

To test our algorithms, we have collected the available instances from the literature, which tend to be simpler shapes. The first instance is a set of convex polyhedra that has been introduced by Stoyan et al. (2005) and later used in Egeblad et al. (2009). It consists of seven convex shapes of similar volume that are repeated a number of times each. Stoyan et al. (2005) defined three sets of items *Example1*, *Example2*, and *Example3*; but we only solve the latter two because *Example1* contains one item that has a dimension coinciding exactly with the width of the container and it becomes infeasible with our voxelization parameters. We call these instances *Stoyan2* and *Stoyan3*.

In order to find more challenging instances, Egeblad et al. (2009) mixed this set with some items available in the work of Ikonen et al. (1997), creating the Merged*i* instances, that we solve as well.

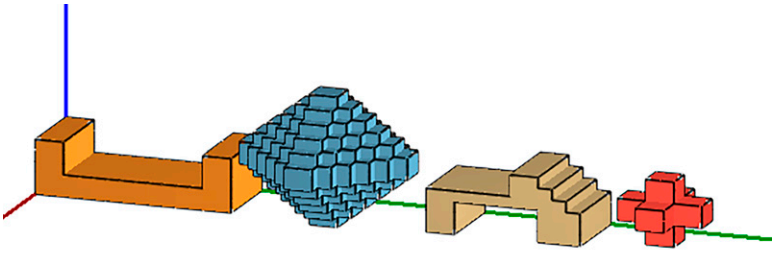
The other instances we consider are a set of nonconvex polyhedra from Stoyan et al. (2004) (*Experiment2* to *Experiment5*), which has been later solved by Liu et al. (2015) and a similar set of polyhedra proposed by Liu et al. (2015), *Experiment1*. However, in the original work, *Experiment1* has been solved only when allowing rotation; therefore, we do not solve exactly the same problem, because our orientations are fixed.

These instances are originally represented using the polygonal mesh approach; therefore, it is necessary to convert them to voxels before applying our algorithms. To perform this process, we have used the binvox software package (Nooruddin and Turk 2003, Min 2019), where we apply the *exact* setting to ensure the voxelized representation overestimates the original one; therefore, our solutions are always feasible. One important parameter used in the conversion is the resolution, that is, the size of the voxels used in the converted shapes. In order to solve each instance with the same resolution, we have chosen a voxel size

Table 1. Instances Solved and Their Features

Instance	Source	Convex	Holes	Items (types)
Stoyan2	Stoyan et al. (2005)	✓	✗	12 (7)
Stoyan3	Stoyan et al. (2005)	✓	✗	25 (7)
Merged*i*	Egeblad et al. (2009)	✗	✓	15–75 (15)
Experiment1	Liu et al. (2015)	✗	✓	36 (5)
Experiment*i*	Stoyan et al. (2004)	✗	✓	20–50 (10)
Blobs*i*	Generated	✗	✓	20 (10)
Shapes_3D	Adapted	✗	✗	43 (4)
Engine	Realistic	✗	✓	97 (56)
Chess	Realistic	✗	✗	32 (6)

Figure 5. (Color online) Instance *Shapes_3D*



such that the largest side of the container is voxelized with 150 voxels. Where there are several instances containing common items, we chose the largest side of the largest container among the instances so that the items have a common voxelization. This means that the smaller containers in the group of instances might not get an exact voxelization; therefore, we round the dimensions down, ending up with a slightly smaller container. One hundred fifty voxels are chosen, as it seemed a reasonable compromise between accuracy and solving speed. Nevertheless, as it will be shown in Section 5.4, the resolution chosen plays a key role in both the solution time and the quality of the results.

5.1.2. A Simpler Instance. To complete the tests with simpler shapes, we have also adapted a classical instance from the two-dimensional irregular packing literature, the set *shapes0* from Oliveira et al. (2000). This set consists of four simple distinct shapes (one convex and three nonconvex) and has been widely used in the research of the strip packing problem. To adapt these figures to 3D, we maintained the original proportions of the original shapes and voxelized them in a grid of one unit. For two of the shapes, we preserved their symmetry along the third axis. For the remaining two, we extruded the shape along the third axis so they had a size of six voxels. We call our instance *Shapes_3D*, and the final result can be seen in Figure 5. The container for this set has sides 20×20, and we maintained the same item description (43 items in total) and fixed the orientation.

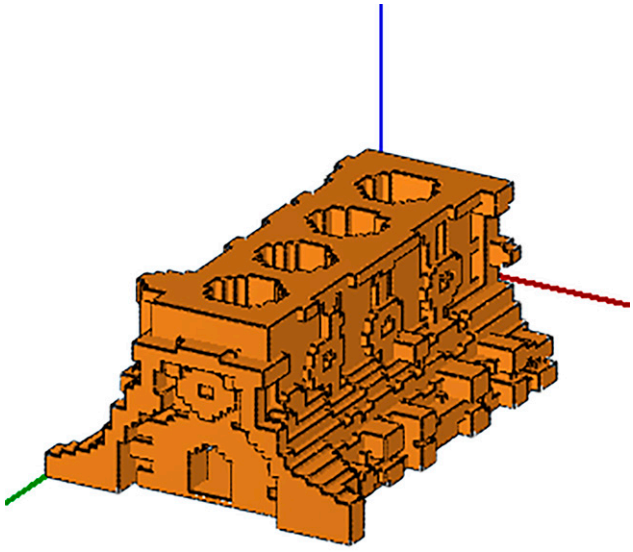
5.1.3. Realistic Instances. We also solve two instances based on models downloaded from a popular 3D printing online community. In order to solve a meaningful problem, we chose two designs that have shapes with a certain complexity and that are themselves a collection of different parts. We chose a realistic model of an engine by Harrell (2015) (see Figure 6 for an example of an item already voxelized) and a chess set by Edwards (2014).

Both of these instances were available as triangular mesh files. We converted them using the same process as described in Section 5.1.1; but as there was no container size provided, we chose the resolution differently. Based on the size of the items and because they were available in different scales, we decided to pack the *Engine* in a container with a base of 500×500 voxels, where one voxel represents six units, and the *Chess* set in a container of size 50×50, where one voxel is mapped to one unit of the original model.

5.1.4. Randomly Generated Instances. Our random instance generator extends to three dimensions the shape generation approach presented in Robidoux et al. (2011). We illustrate some of its steps in Figure 7 in an example to generate a 2D shape, because it is easier to appreciate in two dimensions. The following steps describe the method for a 3D item:

Step 1. Start with a three-dimensional matrix of zeros, which will be the container of the future item.

Figure 6. (Color online) Example Item from the *Engine* Instance



Step 2. Randomly assign one to the value of some elements in the matrix (Figure 7(a)).

Step 3. Connect the points in a random order with discretized lines of a certain thickness. The last point is connected with the first in order to create a loop (Figure 7(b))

Step 4. Apply a Gaussian blur to the matrix. This step changes the values of the elements so they are between zero and one (Figure 7(c))

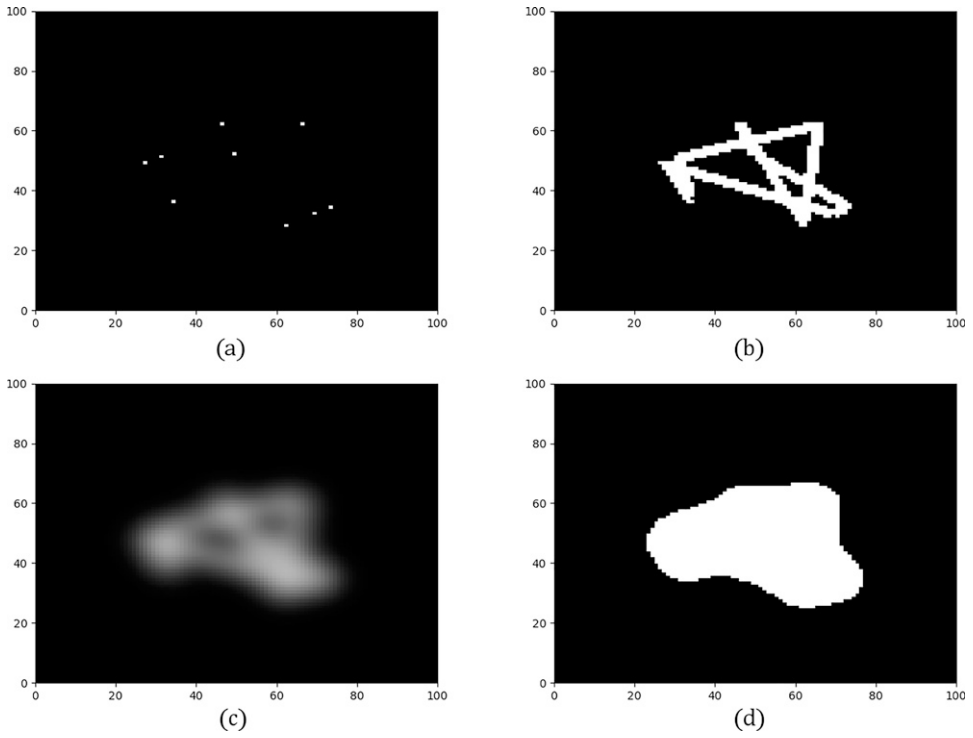
Step 5. To convert to binary values again, apply a threshold, where values less than a parameter (around 0.005) are set to zero and values over that parameter are set to one (Figure 7(d)).

There are a few parameters that can be controlled in this method. The first of them is the size of the matrix where the points are drawn. A larger size will mean a smoother shape but more costly to compute. We also need to decide how many and how to select the random elements that are initially set to one. More points give more opportunities for the shape to have irregularities; but too many will fill up the space of the shape too much and it can lose some interesting features, such as holes. For sampling the points, we have chosen a random uniform distribution in a cube; but there are other possible distributions. Robidoux et al. (2011) suggest using distributions in a disc (uniform or normal) if one wishes to get rotation invariant shapes, but this is not relevant for our purposes.

The way the points are connected can also impact the shape. In our case, we use straight lines; but other methods, such as splines, could give more smooth results. Another decision is how to add thickness to these lines. Looking again at Figure 7(b), we see that the lines in that case have more than one pixel in thickness. To explain this, let us think about the matrix as a grid of squares (or cubes, in 3D). When connecting two points, we first paint the squares of the grid intersected by the segment that connects the centre of the two points. By painting here we mean setting to one the corresponding matrix elements. To add thickness, we also paint the squares surrounding this initial line, if they are within a certain distance from it in any axis aligned direction. This distance is another parameter, which we call the thickness parameter. This results in painting with a kind of cross-hairs, the same concept as depicted in the neighborhood from Figure 3. Finally, both the standard deviation of the Gaussian blur and the threshold also have an impact on the outcome.

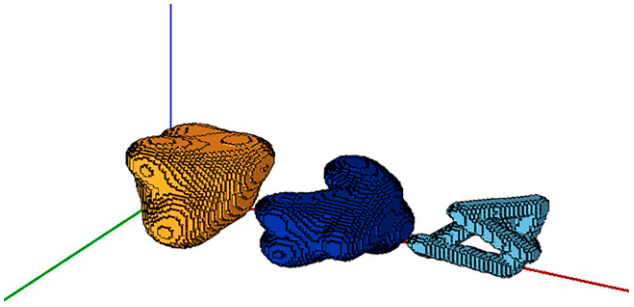
Following this technique, we have generated nine instances in total, with three different sizes and three shape styles for each size. We handpicked the parameters in order to define the three different styles, which we have called *round*, where items are closer to spheres or ellipsoids; *peaked*, where shapes have more

Figure 7. Example of the Steps to Generate a Blob in Two Dimensions



Note. In (a), nine points are drawn randomly; in (b), they are connected in a closed loop; in (c), a Gaussian blur is applied (the values between zero and one are represented by a shade of grey); and in (d), values are set to either zero or one, depending on a threshold value.

Figure 8. (Color online) Three Items from the Medium Instances with Style Round (Left), Neutral (Centre), and Peaked (Right)



pronounced concavities and more frequently holes; and *neutral*, which is somehow a middle ground between the two. In Figure 8, we show one item representing each different style. The full description of the parameters used is reported in Table 2, where intervals indicate that the parameter has been randomly chosen in that interval.

The matrix size in Table 2 is indicative, as applying the Gaussian blur might set to positive some values outside the original cube where the points were sampled (this happens, for example, if one of the points was in the corner of the cube). To acknowledge this fact and ensure the shapes remained smooth, we allowed a maximum size of $100 \times 100 \times 100$ for any shape, which seemed sufficient to avoid such cases.

Based on the size of the largest item from each set, we chose the size of the container base to be (in voxels) 78×78 for the small instances, 84×84 for the medium instances, and 90×90 for the large instances.

5.2. Testing the ILP Model

The model presented in Section 2.4 can provide an optimal solution if solved to optimality. However, the large number of variables for the proposed instances makes it often impractical to solve. In order to test its boundaries, we have chosen the four items from the *Shapes_3D* instance and generated all of the 155 instances that feature each item type repeated between zero and three times and whose total number of items is between four and seven. The container size was set to 20×20 voxels. We have then solved them for a maximum of 10 hours of computational time, using IBM® ILOG® CPLEX® 12.6.1 and providing the bottom-left-back upper bound.

In eight of the instances, the bottom-left-back algorithm found a solution such that the height used corresponds to the height of the largest item and, therefore, the solution is optimal. In Table 3, we can observe that the percentage of solved instances clearly decreases when adding items, to the point that only 43% of the instances with seven items can be solved in the given time. In Figure 9, we show one instance with seven items solved to optimality. The gap obtained when the instances were not solved to optimality is usually large, which suggests that the difficulty of solving the derived models increases quickly.

5.3. Metaheuristic Algorithms Configurations

In this section, we describe the parameters we have used to configure our algorithms for the computational experiments. The configuration was done by using the software package "irace" (López-Ibáñez et al. (2016)) with a budget of 1,000 one-hour runs for each method. The single parameter tested for ILS is reported in Table 4.

In Tables 5 and 6, we report the tests performed for TS and VNS, respectively. Note that the parameters for strategic oscillation (`SO_mode` and `dec_p`) have been tested independently for both methods, as they could (although they do not, in this case) differ on their best value for different search strategies.

The large value of `test_swaps` suggests that it is beneficial testing as many swaps as possible during the kick in VNS. The values of the parameters found by irace are the ones we use in the remainder of the computational experiments.

5.4. Algorithm Comparison

We have implemented all our algorithms in C++ and run them in a single 2.6 GHz core with 16 GB of

Table 2. Parameters Used to Generate the Blobs Instances

Name	*Size*	*Style*	*Matrix size*	*Points*	*Thickness*	*Filter σ*	*Threshold*
blobs1	Small	Round	$25 \times 25 \times 25$	[2,20]	3	[3,5]	0.05
blobs2	Small	Neutral	$25 \times 25 \times 25$	[2,15]	[1,3]	[1,2]	0.01
blobs3	Small	Peaked	$25 \times 25 \times 25$	[3,7]	1	[1,2]	0.005
blobs4	Medium	Round	$35 \times 35 \times 35$	[4,30]	[4,5]	[4,5]	0.05
blobs5	Medium	Neutral	$35 \times 35 \times 35$	[4,25]	[3,4]	[2,3]	0.01
blobs6	Medium	Peaked	$35 \times 35 \times 35$	[3,10]	[1,3]	[1,2]	0.005
blobs7	Large	Round	$45 \times 45 \times 45$	[4,40]	[4,5]	[4,5]	0.05
blobs8	Large	Neutral	$45 \times 45 \times 45$	[4,3]	[3,4]	[2,3]	0.01
blobs9	Large	Peaked	$45 \times 45 \times 45$	[3,14]	[1,3]	[1,2]	0.005

Table 3. Summary of the Results Achieved by Solving the ILP Model on Instances with Between Four and Seven Items

Items	Instances	Opt	Percentage
4	31	27	87%
5	40	30	75%
6	44	26	59%
7	40	17	43%
Total	155	100	65%

Note. Opt, the number of instances solved to optimality in each set.

memory, part of the IRIDIS High Performance Computing Facility. All of our experiments allow one hour of computational time, and we report both the best and the average results obtained after 10 runs. Because all the methods tested use nofit voxels, they have been precalculated and the generation and loading time is not included in the reported time of the algorithms.

It is worth noting that as a result of the voxelization of items, we may be packing slightly larger items into a slightly smaller container. This has two consequences. First, some configurations of the items that may be feasible in a mesh representation may not be feasible in the voxel representation with a given fidelity, which may include the best layout in the literature. Second, comparing solutions with the literature on the basis of volume utilisation no longer makes sense as the total volume of the items is not the same. For example, for the instance *Merged1*, the total volume of our voxelized version of the items is 15% larger than the original. To address the second point, all our experiments compare the height of the packing layout.

The results for the *blobs* instances are shown in Table 7. We can observe that VNS obtains on average the best results. The average height is 162.64 voxels, which is 1.7% better than the TS and 8% better than the ILS. As for the best results, in seven out of nine instances, the VNS obtained the best solution. This pattern is also observed in the *Experiment* set (Table 9), where VNS is again superior to all the other methods. All three voxel-based methods obtain the best result in the literature for these instances. The same trend is repeated on the realistic instances and the adapted *Shapes*, reported in Table 8. All methods here obtain the same value for *Shapes_3D*, although ILS seems to be more consistent. For the realistic instances *Chess* and *Engine*, VNS obtains both the best and most consistent results.

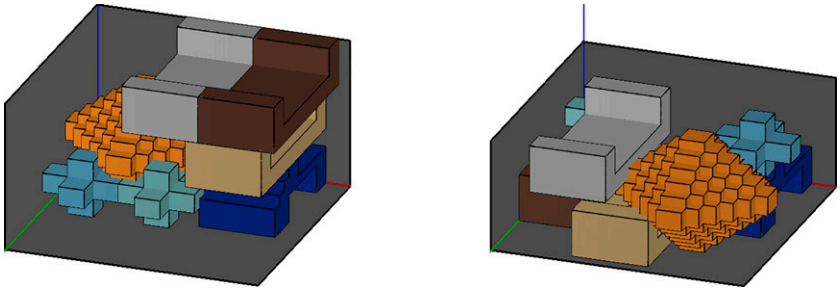
In Figure 10, we show some of the best results found for this group of instances; in Table 9, we report the results for the *Experiment* instances.

All voxel-based methods perform better than both HAPE3D (Liu et al. 2015) and the phi-function based methods (Stoyan et al. 2004). Among them, VNS has the best and most consistent results. In Table 10, we report the results for the *Stoyan* and *Merged* instances.

Although the voxel-based methods obtain better results than the phi-functions, the algorithm 3DNest, based on a polygonal mesh, obtains slightly better results for all instances but one. We note that the 3DNest algorithm permits a small overlap between items (0.1%). In order to distinguish whether the differences in quality for the *Merged* instances could be linked to the resolution chosen when voxelizing the items, we performed another set of tests where VNS uses finer resolutions and longer search times. We considered two extra resolutions, *fine* and *extrafine*, where the largest side of the largest container base is voxelized with 200 and 250 voxels, respectively, and allowed a maximum time computational of four hours. The results for these extended tests are shown in Table 11.

These illustrate how increasing the accuracy of the representation closes the gap between 3DNest and VNS, and with finer resolutions VNS is able to improve the average solution quality by 1.4%. As a result, we are finding the best known solutions for the largest instances (60 and 75 items) while being within less than 1% of the height in the three smaller instances. Because VNS does better on larger problems, this suggests our approach is more scalable.

Figure 9. (Color online) Initial Solution from Bottom Left Back (Left, 16 Voxels Height) and Optimal Solution (Right, 12 Voxels Height) for One Instance with Seven Items



Note. Solving time was 78 seconds.

Table 4. Parameter Tested for the ILS and the Best Values Found by irace

Parameter	Description	Tested values	Best
`n_swaps`	Number random item swaps performed between iterations	$[1,25] \subset \mathbb{N}$	4

Table 5. Parameters Tested for the TS Algorithm and the Best Values Found by irace

Parameter	Description	Tested values	Best
`nMoves`	Number of tabu search movements performed before strategic oscillation	10^i, $i = 2,3,4,5,6$	10^4
`d_axis`	Parameter δ used for the axis aligned neighborhood (N_A)	1, 3, 5, 10, ∞	∞
`TL`	Length of the tabu list	$[1,30] \subset \mathbb{N}$	13
`SO_mode`	Strategy used to relocate protruding items during strategic oscillation	*MD*, *RR*	*MD*
`dec_p`	Percentage of height reduced in strategic oscillation	$[1,25] \subset \mathbb{R}$	7%

Note. In strategic oscillation, *MD* stands for *move down* and *RR* stands for *random reinsert*.

Table 6. Parameters Tested for the VNS and the Best Values Found by irace

Parameter	Description	Tested values	Best
`nK`	Number of kicks before strategic oscillation	$[1,100] \subset \mathbb{N}$	78
`K_dmin`	Value of δ_{min} in VNS kicks	$[1,3] \subset \mathbb{N}$	1
`K_dmax`	Value of δ_{max} in VNS kicks	$[2,10] \subset \mathbb{N}$	8
`SO_mode`	Strategy used to relocate protruding items during strategic oscillation	*MD*, *RR*	*MD*
`dec_p`	Percentage of height reduced in strategic oscillation	$[1,25] \subset \mathbb{R}$	7%
`test_swaps`	Number of swaps tested for each item during the kick	$[1,50] \subset \mathbb{N}$	48
`d_cube`	Parameter δ used for the cube neighborhood (N_C)	$[1,10] \subset \mathbb{N}$	5
`d_axis`	Parameter δ used for the axis aligned neighborhood (N_A)	1, 3, 5, 10, ∞	∞
`Ov_d`	When running the model, δ value used for the cube neighborhood of overlapping items	$[1,4] \subset \mathbb{N}$	1
`NOv_d`	When running the model, δ value used for the cube neighborhood of nonoverlapping items	$[1,2] \subset \mathbb{N}$	2

Note. In strategic oscillation, *MD* stands for *move down* and *RR* stands for *random reinsert*.

Table 7. Comparison of Results (Final Height, in Voxels) for the Blobs Instances

Instance	ILS		TS		VNS	
	Best	*Avg*	*Best*	*Avg*	*Best*	*Avg*
blobs1	79	79.64	75	76.90	**74**	74.90
blobs2	64	65.91	**59**	60.90	**59**	59.60
blobs3	47	47.73	**44**	45.20	**44**	44.50
blobs4	211	214.18	202	205.60	**187**	191.70
blobs5	221	228.91	206	211.30	**202**	205.00
blobs6	91	94.82	88	89.10	**86**	88.00
blobs7	337	343.36	**309**	323.40	312	323.50
blobs8	357	368.82	**329**	338.10	330	340.50
blobs9	146	149.36	136	138.60	**133**	136.10
Av.		176.97		165.46		**162.64**

Note. Best solutions are highlighted in bold.

Table 8. Comparison of Results (Final Height, in Voxels) for the *Shapes_3D and Realistic Instances*

Instance	ILS		TS		VNS	
	Best	*Avg*	*Best*	*Avg*	*Best*	*Avg*
Shapes_3D	**52**	52.73	**52**	53.00	**52**	53.20
Engine	78	79.36	77	78.70	**75**	78.10
Chess	144	145.55	139	141.40	**131**	135.10

Note. Best solutions are highlighted in bold.

Table 9. Comparison of Results (Final Height) for the Instances Found in St04 (Stoyan et al. 2004) and HAPE3D (Liu et al. 2015)

Instance	ILS		TS		VNS		St04	HAPE3D
	Best	*Avg*	*Best*	*Avg*	*Best*	*Avg*	*Best*	*Best*
Exp1	36.40	38.58	34.13	35.12	34.27	35.37	—	31.2[a]
Exp2	29.40	31.10	**28.47**	30.17	**28.47**	29.40	32	36.9
Exp3	42.70	46.05	41.77	43.54	**41.30**	42.65	49	55.6
Exp4	62.30	65.35	56.23	59.62	**54.83**	59.52	63.21	71.8
Exp5	77.00	81.41	74.20	76.16	**72.33**	76.81	78.7	92

[a]The result of 31.2 for the instance *Exp1* is obtained by allowing eight different rotations, which are not allowed in the other algorithms.
Note: Best solutions are highlighted in bold.

Figure 10. (Color online) Best Result Obtained by ILS for *Shapes_3D* (Left, Height of 52 Voxels); by TS for *Engine* (Centre, Height of 77 Voxels); and by VNS for *Chess* (Right, Height of 131 Voxels)

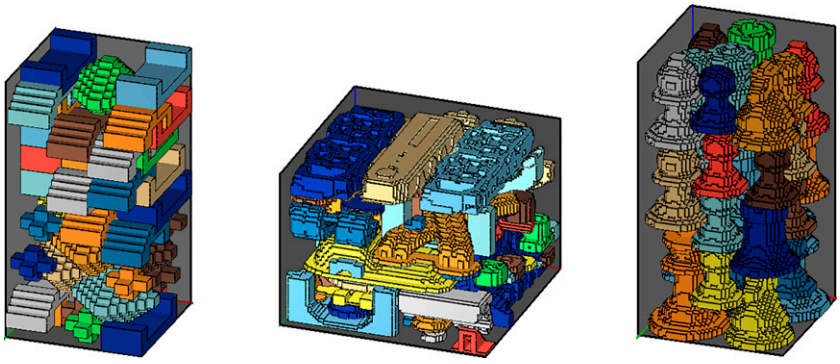


Table 10. Comparison of Results (Final Height) for the Instances Found in St05 (Stoyan et al. 2005) and 3DNEST (Egeblad et al. 2009)

Instance	ILS		TS		VNS		St05	3DNEST
	Best	*Avg*	*Best*	*Avg*	*Best*	*Avg*	*Best*	*Best*
Example2	22.72	23.30	21.33	21.60	21.23	21.70	30.92	**19.83**
Example3	34.99	35.65	30.93	32.35	**29.76**	31.53	45.86	29.82
Merged1	20.52	21.19	18.47	19.72	18.30	18.67	—	**16.99**
Merged2	24.62	25.58	22.40	23.27	22.40	22.64	—	**21.97**
Merged3	28.73	29.48	27.02	28.16	24.62	26.26	—	**24.04**
Merged4	30.61	31.86	28.73	30.27	28.56	29.44	—	**27.80**
Merged5	34.20	35.65	30.61	31.70	30.61	31.12	—	**30.13**

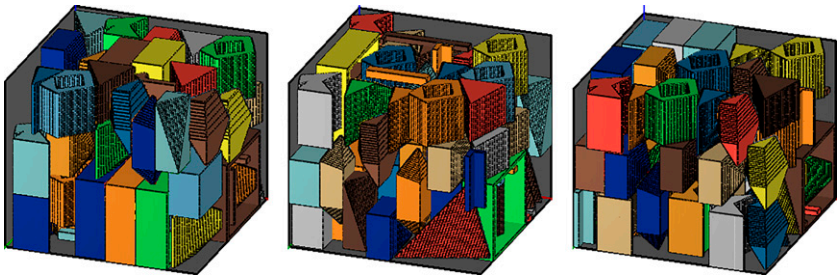
Note. Best solutions are highlighted in bold.

Table 11. Extended Results for the *Merged* Instances, Using More Time and Finer Resolutions for VNS

Instance	VNS-*Standard*	VNS-*Fine*	VNS-*Extrafine*	3DNest
Merged1	18.30	17.95	17.13	**16.99**
Merged2	22.40	22.19	22.06	**21.97**
Merged3	24.62	24.24	24.32	**24.04**
Merged4	28.56	28.21	**26.47**	27.80
Merged5	30.61	30.27	**29.24**	30.13
Av	24.90	24.57	**23.84**	24.17

Note. Av and Avg both stand for Average.

Figure 11. (Color online) Best Layouts Found for the *Merged5* Instance with Three Different Resolutions: *Standard* (Left, Height of 30.61), *Fine* (Centre, Height of 30.27), and *Extrafine* (Right, Height of 29.24, Best Known Value for This Instance)



The best layouts obtained for each resolution for the instance *Merged5* can be seen in Figure 11.

6. Conclusion

In this article, we explore the potential of voxel-based approaches for solving the three-dimensional irregular open dimension problem. We develop the concept of nofit voxel, in which we base our algorithms. The usage of voxels and the nofit voxel allow for a quick and robust way of dealing with complex items in packing problems.

With these tools at hand, we formulate the problem as an ILP model that is solved to optimality for small instances but can also be used with a compaction/separation strategy. We propose a constructive algorithm and an iterated local search that extends the quality of the constructive algorithm. To further improve the packing quality, we propose two metaheuristic techniques that search over the space of solutions containing overlap. We collected a set of instances from the literature and added our own to create a benchmark set of instances with different characteristics where we could test our algorithms. We show that both metaheuristic techniques are very competitive, even when compared with more complex and accurate geometrical representations, such as the polygonal mesh or the phi-functions. The choice of voxels and the nofit voxel to represent and pack the items is backed by our computational experiments, where we show that our results outperform most of the preexisting literature. The only exception is the algorithm 3DNest, which finds similar quality solutions than a variable neighborhood search with a fine resolution. This also shows, however, that the ability of voxels to use different representation accuracy depending on the quality/speed of packing required makes voxel-based approaches more suitable for real-world problems, where the items to be packed might be of higher complexity than the standard academic instances. Finally, we are able to solve efficiently real-life instances containing complex geometries as typically used in the 3D printing industry, where we show that both the tabu search and the variable neighborhood search algorithms can find competitive results, with significant improvement over our constructive-based iterated local search.

Acknowledgments

The authors acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work.

References

Abeysooriya RP, Bennell JA, Martinez-Sykora A (2018) Jostle heuristics for the 2D-irregular shapes bin packing problems with free rotation. *Internat. J. Production Econom.* 195:12–26.

Art R (1966) An approach to the two dimensional irregular cutting stock problem. Unpublished thesis, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA.

Attene M (2015) Shapes in a box: Disassembling 3D objects for efficient packing and fabrication. *Comput. Graphics Forum* 34(8):64–76.

Baert J, Lagae A, Dutré P (2014) Out-of-core construction of sparse voxel octrees. *Comput. Graphics Forum* 33(6):220–227.

Baumers M, Tuck C, Wildman R, Ashcroft I, Rosamond E, Hague R (2013) Transparency built-in: Energy consumption and cost estimation for additive manufacturing. *J. Indust. Ecology* 17(3): 418–431.

Bennell JA, Dowsland KA (1999) A tabu thresholding implementation for the irregular stock cutting problem. *Internat. J. Production Res.* 37(18):4259–4275.

Bennell JA, Dowsland KA (2001) Hybridising tabu search with optimisation techniques for irregular stock cutting. *Management Sci.* 47(8):1160–1172.

Bennell J, Oliveira J (2008) The geometry of nesting problems: A tutorial. *Eur. J. Oper. Res.* 184(2):397–415.

Bennell Ja, Oliveira JF (2009) A tutorial in irregular shape packing problems. *J. Oper. Res. Soc.* 60:S93–S105.

Bennell JA, Dowsland KA, Dowsland WB (2000) The irregular cutting-stock problem: A new procedure for deriving the no-fit polygon. *Comput. Oper. Res.* 28(3):271–287.

Bennell J, Scheithauer G, Stoyan Y, Romanova T (2010) Tools of mathematical modeling of arbitrary object packing problems. *Ann. Oper. Res.* 179(1):343–368.

Błazewicz J, Hawryluk P, Walkowiak R (1993) Using a tabu search approach for solving the two-dimensional irregular cutting problem. *Ann. Oper. Res.* 41(4):313–325.

Burke E, Hyde M, Kendall G, Woodward J (2010) A genetic programming hyper-heuristic approach for evolving 2-D strip

packing heuristics. *IEEE Trans. Evolutionary Comput.* 14(6):942–958.

Byholm T, Toivakka M, Westerholm J (2009) Effective packing of 3-dimensional voxel-based arbitrarily shaped particles. *Powder Tech.* 196(2):139–146.

Cagan J, Degentesh D, Yin S (1998) A simulated annealing-based algorithm using hierarchical models for general three-dimensional component layout. *Comput.-Aided Design* 30(10):781–790.

de Korte A, Brouwers H (2013) Random packing of digitized particles. *Powder Tech.* 233:319–324.

Dickinson JJK, Knopf GKG (1998) Serial packing of arbitrary 3D objects for optimizing layered manufacturing. *Proc. SPIE* 3522: 130–138.

Dowsland KA, Dowsland WB, Bennell JA (1998) Jostling for position: Local improvement for irregular cutting patterns. *J. Oper. Res. Soc.* 49(6):647–658.

Edelkamp S, Wichern P (2015) Packing irregular-shaped objects for 3D printing. Hölldobler S, Krötzsch M, Peñaloza R, Rudolph S, eds. *KI 2015: Adv. Artificial Intelligence: 38th Annual German Conf. AI*, Lecture Notes in Computer Science, vol. 9324 (Springer International Publishing, Cham, Switzerland), 45–58.

Edwards T (2014) Classic chess set from glChess. Accessed December 1, 2019, https://www.thingiverse.com/thing:322616.

Egeblad J, Nielsen BK, Brazil M (2009) Translational packing of arbitrary polytopes. *Comput. Geometry* 42(4):269–288.

Egeblad J, Nielsen BK, Odgaard A (2007) Fast neighborhood search for two- and three-dimensional nesting problems. *Eur. J. Oper. Res.* 183(3):1249–1266.

Egeblad J, Garavelli C, Lisi S, Pisinger D (2010) Heuristics for container loading of furniture. *Eur. J. Oper. Res.* 200(3):881–892.

Glover F (1989) Tabu search-part I. *ORSA J. Comput.* 1(3):190–206.

Glover F, Laguna M (2013) Tabu Search* In: Pardalos P, Du DZ, Graham R, eds. *Handbook of Combinatorial Optimization* (Springer, New York, NY), 3261–3362.

Gomes A, Oliveira JF (2002) A 2-exchange heuristic for nesting problems. *Eur. J. Oper. Res.* 141(2):359–370.

Gomes AM, Oliveira JF (2006) Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *Eur. J. Oper. Res.* 171(3):811–829.

Griffiths V, Scanlan JP, Eres MH, Martinez-Sykora A, Chinchapatnam P (2019) Cost-driven build orientation and bin packing of parts in selective laser melting (SLM). *Eur. J. Oper. Res.* 273(1): 334–352.

Harrell E (2015) Toyota 4 cylinder engine 22RE, complete working model. Accessed December 1, 2019, http://www.thingiverse.com/thing:644933.

Hur SM, Choi KH, Lee SH, Chang PK (2001) Determination of fabricating orientation and packing in SLS process. *J. Materials Processing Tech.* 112(2-3):236–243.

Ikonen I, Biles W, Kumar A, Wissel J, Ragade R (1997) A genetic algorithm for packing three-dimensional non-convex objects having cavities and holes. *Proc. 7th Internat. Conf. Genetic Algorithms* (Morgan Kaufmann Publishers, East Lansing, Michigan), 591–598.

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

Jia X, Williams R (2001) A packing algorithm for particles of arbitrary shapes. *Powder Tech.* 120(3):175–186.

Lemus E, Bribiesca E, Garduño E (2015) Surface trees–Representation of boundary surfaces using a tree descriptor. *J. Visual Comm. Image Representation* 31(C):101–111.

Liu X, Liu JM, Cao AX, Yao ZL (2015) HAPE3D–A new constructive algorithm for the 3D irregular packing problem. *Frontiers Inform. Tech. Electronic Engrg.* 16(5):380–390.

López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L, Birattari M, Stützle T (2016) The irace package: Iterated racing for automatic algorithm configuration. *Oper. Res. Perspect.* 3:43–58.

Lourenço HR, Martin OC, Stützle T (2010) *Iterated Local Search: Framework and Applications* (Springer, Boston), 363–397.

Ma Y, Chen Z, Hu W, Wang W (2018) Packing irregular objects in 3D space via hybrid optimization. *Comput. Graphics Forum* 37(5):49–59.

Mangiaracina R, Marchet G, Perotti S, Tumino A (2015) A review of the environmental implications of B2C e-commerce: A logistics perspective. *Internat. J. Physical Distribution Logist. Management* 45(6):565–591.

Mazareanu E (2020) Parcel traffic worldwide by sector 2015-2019. https://www.statista.com/statistics/737418/parcel-traffic-worldwide-by-sector/.

Milenkovic V, Daniels K, Li Z (1992) Placement and compaction of nonconvex polygons for clothing manufacture. *Proc. Fourth Canadian Conf. Comput. Geometry* (Memorial University of Newfoundland, St. John's NL, Canada), 236–243.

Min P (2019) Binvox. http://www.patrickmin.com/binvox or https://www.google.com/search?q=binvox.

Mladenović N, Hansen P (1997) Variable neighborhood search. *Comput. Oper. Res.* 24(11):1097–1100.

Mundim LR, Andretta M, Carravilla MA, Oliveira JF (2018) A general heuristic for two-dimensional nesting problems with limited-size containers. *Internat. J. Production Res.* 56(1-2):709–732.

Nooruddin FS, Turk G (2003) Simplification and repair of polygonal models using volumetric techniques. *IEEE Trans. Visualization Comput. Graphics* 9(2):191–205.

Oliveira JF, Gomes AM, Ferreira JS (2000) TOPOS–A new constructive algorithm for nesting problems. *OR Spectrum* 22(2):263.

Pankratov AV, Romanova TE, Chugay AM, Problems E, Acad N (2015) Optimal packing of convex polytopes using quasi-phi-functions. *Jixie Gongcheng Xuebao* 18(2):55–64.

Robidoux N, Stelldinger P, Cupitt J (2011) Simple random generation of smooth connected irregular shapes for cognitive studies. *Proc. Fourth Internat. C* Conf. Comput. Sci. Software Engrg.* (ACM Press, New York), 17–24.

Romanova T, Bennell J, Stoyan Y, Pankratov A (2018) Packing of concave polyhedra with continuous rotations using nonlinear optimisation. *Eur. J. Oper. Res.* 268(1):37–53.

Sánchez-Cruz H, López-Valdez HH, Cuevas FJ (2014) A new relative chain code in 3D. *Pattern Recognition* 47(2):769–788.

Scheithauer G, Stoyan YG, Romanova TY (2005) Mathematical modeling of interactions of primary geometric 3D objects. *Cybernetics Systems Anal.* 41(3):332–342.

Schwarz M, Seidel HP (2010) Fast parallel surface and solid voxelization on GPUs. *ACM Trans. Graphics* 29(6):1–10.

Stoyan YG (1983) Mathematical methods for geometric design. *Advances in CAD/CAM, Proc. PROLAMAT82, Leningrad, USSR, May 1982* (North-Holland, Amsterdam), 67–86.

Stoyan YG, Gil N, Pankratov A, Scheithauer G (2004) Packing non-convex polytopes into a parallelepiped. *Preprint MATH-NM-06-2004: Technische Universität of Dresden*.

Stoyan YG, Gil NI, Scheithauer G, Pankratov A, Magdalina I (2005) Packing of convex polytopes into a parallelepiped. *Optim.* 54(2): 215–235.

Terashima-Marín H, Ross P, Farías-Zárate CJ, López-Camacho E, Valenzuela-Rendón M (2010) Generalized hyper-heuristics for solving 2D regular and irregular packing problems. *Ann. Oper. Res.* 179(1):369–392.

Thompson MK, Moroni G, Vaneker T, Fadel G, Campbell RI, Gibson I, Bernard A, et al. (2016) Design for additive manufacturing: Trends, opportunities, considerations, and constraints. *CIRP Ann. Manufacturing Tech.* 65(2):737–760.

Toledo FM, Carravilla MA, Ribeiro C, Oliveira JF, Gomes AM (2013) The dotted-board model: A new MIP model for nesting irregular shapes. *Internat. J. Production Econom.* 145(2): 478–487.

Umetani S, Yagiura M, Imahori S, Imamichi T, Nonobe K, Ibaraki T (2009) Solving the irregular strip packing problem via guided

local search for overlap minimization. *Internat. Trans. Oper. Res.* 16(6):661–683.

Wäscher G, Haußner H, Schumann H (2007) An improved typology of cutting and packing problems. *Eur. J. Oper. Res.* 183:1109–1130.

Carlos Lamas-Fernandez is a lecturer in business analytics and management science in Southampton Business School. He is interested in the research of combinatorial optimisation problems arising in cutting and packing, health care, transportation and logistics.

Julia A. Bennell is the executive dean of Leeds University Business School and professor of operational research. She is an expert in optimisation problems for cutting and packing, scheduling and a range of transportation problems.

Antonio Martinez-Sykora is an associate professor in business analytics in Southampton Business School, where he is also the current director of the Centre of Operations Research, Management Science, and Information Systems (CORMSIS). He is a coordinator of the Euro Working Group on Cutting and Packing and his research interests are on exploring new solutions methods on packing problems, logistics and transportation, revenue management and scheduling and timetabling.