Advanced Intelligent Systems

ADVANCED INTELLIGENT SYSTEMS Open Access

www.advintellsyst.com

RESEARCH ARTICLE OPEN ACCESS

Optimizing 3D Bin Packing of Heterogeneous Objects Using Continuous Transformations in $SE(3)$

Michele Angelini | Marco Carricato

Department of Industrial Engineering, University of Bologna, Bologna, Italy

Correspondence: Marco Carricato (marco.carricato@unibo.it)

Received: 3 October 2025 | **Revised:** 13 March 2026 | **Accepted:** 16 March 2026

Keywords: 3D bin packing problem (3D-BPP) | heuristic methods | neural networks | optimization | physics simulation | signed-distance function (SDF) | static stability

ABSTRACT

With the growth of online e-commerce platforms, the challenge of automatically packing objects into confined spaces has attracted increasing attention from the scientific community. This paper presents an algorithm tackling the bin packing problem of heterogeneous objects or, more precisely, the task of finding appropriate poses for items whose geometry can be traced back to primitive convex shapes, when these are to be inserted inside a box-shaped bin. Since this problem is strongly NP-complete, finding a solution in practical timeframes is not trivial for industrial applications, in which boxes must be filled in the span of seconds or minutes. This paper presents a heuristic-driven optimization problem that leverages a point-cloud representation of the bin and signed-distance functions of the items to be packed. Solution is sought in a continuous subset of $SE(3)$, including both continuous translations and continuous rotations. To enhance robustness, the static stability of the items in the box is ensured through a mesh-based physics simulator. The proposed approach can be used, with suitable variants, for both offline and online packing. Performance is evaluated through simulations conducted within the physics simulator, evaluating the algorithm performance in different scenarios.

1 | Introduction

The bin packing problem (BPP) is a well-studied combinatorial problem tackled by mathematicians since the late 1930s [1]. In its simplest form, it consists of grouping a set of real numbers sampled in the interval $[0, 1]$ so that the numbers in each group do not add up to more than 1. Variations and extensions of this problem to multidimensional domains find multiple applications throughout different industry fields dealing with scheduling, resource allocation, stock cutting, ship loading, and many more [2].

One field in which the BPP is fundamental and gaining traction in recent years is e-commerce. In this context, customers remotely select several items that are to be packed in a cardboard box, ready to be shipped. In this case, the BPP is formulated in a 3D context, in which heterogeneous volumes (defined by the items' geometries) must be placed in a box-like container, while complying with stability and feasibility constraints. This paper presents an algorithm that finds appropriate item poses as the solutions to a heuristic-driven optimization problem. The algorithm leverages signed-distance functions (SDFs) and neural-network approximations to represent objects, point clouds to assess the occupancy of the box, and mesh-based physics simulations to determine static stability and geometric interference. The approach is capable of:

- dealing with objects with distinct primitive convex shapes, such as cuboids, cylinders, prisms, and pyramidal frusta,
- finding transformations involving both continuous translations and continuous rotations of the objects to be placed in the box,
- being suitable for both offline and online packing, with appropriate variants,

- finding an optimized bin size in case of offline packing,

thus allowing for solutions that would be considered infeasible with state-of-the-art algorithms.

The paper is structured as follows. Section II gives a formal description of the problem and conducts a brief review of the state of the art, highlighting the innovative contributions of the algorithm presented in the paper. Section III explores the main features of the algorithm, namely the item and box modeling strategy, as well as the optimization problem. Section IV describes the algorithm pipeline, giving a complete view of the proposed solution. Section V presents and discusses simulation results. Section VI draws conclusions and sketches future developments.

2 | Related Works and Contributions

The 3D-BPP, as formulated for the e-commerce context, aims at finding suitable poses for 3D heterogeneous volumes inside a bigger, box-shaped volume. Let $I = \{\mathcal{O}_i\}_{i=1}^{N}$ be a set of $N \in \mathbb{N}_+$ volumes $\mathcal{O}_i \subset \mathbb{R}^3$ (here, $\mathbb{R}^3$ represents the 3D Euclidean space). The geometry of the volumes in I is assumed to be known *a priori*. The container is represented by the set $\mathcal{B} = [0, L] \times [0, W] \times [0, H] \subset \mathbb{R}^3$, with L, W, and H defining the size of the box. The volumes of $\mathcal{B}$ and of the generic object $\mathcal{O}_i$ are denoted, respectively, as $|\mathcal{B}|, |\mathcal{O}_i| \in \mathbb{R}_+$. The *final pose* of each object $\mathcal{O}_i$ is represented by a rigid-body transformation from an arbitrary initial pose, described by a 4×4 homogeneous matrix $\mathbf{T}_i \in SE(3)$, where $SE(3)$ is the special Euclidean group. For computational purposes, $\mathbf{T}_i$ is parameterized through a 6-element vector $\boldsymbol{t}_i$ comprising 3 translational displacements along the coordinate axes and 3 angles following the tilt-torsion convention [3]. The transformed object is denoted as $\mathbf{T}_i\mathcal{O}_i$. The set of transformations of all volumes in I is described by $\mathcal{T} = \{\mathbf{T}_i\}_{i=1}^{N} \in SE(3)^N$, which is referred to as the *items/objects, configuration*.

The 3D-BPP is subject to two geometric constraints:

- **containment constraint**: each object in I should lie within $\mathcal{B}$ after applying the transformation $\mathbf{T}_i$:

$$\mathbf{T}_i\mathcal{O}_i \subset \mathcal{B} \quad , \forall i \in \{1, 2, \ldots, N\} \tag{1}$$

- **non-overlapping constraint**: each object in I should not overlap with any other object after applying the corresponding transformations in $\mathcal{T}$:

$$\mathbf{T}_i\mathcal{O}_i \cap \mathbf{T}_j\mathcal{O}_j = \varnothing, \quad \forall i, j \in \{1, 2, \ldots, N\}, i \neq j \tag{2}$$

The 3D-BPP goal is to find $\mathcal{T}$ that maximizes the *packing-utility* function

$$U(\mathcal{T}, \mathcal{B}, I) = \frac{\sum_{i=1}^{N} |\mathcal{O}_i|}{|\mathcal{B}|} \tag{3}$$

while satisfying geometric constraints (1) and (2), namely finding a solution to the optimization problem:

$$\begin{aligned} \arg\max_{\mathcal{T}} \quad & U(\mathcal{T}, \mathcal{B}, I) \\ \text{s.t.} \quad & \mathbf{T}_i\mathcal{O}_i \subset \mathcal{B}, \quad \forall i \in \{1, 2, \ldots, N\} \\ & \mathbf{T}_i\mathcal{O}_i \cap \mathbf{T}_j \cdot \mathcal{O}_j = \varnothing, \quad \forall i, j \in \{1, 2, \ldots, N\}, i \neq j \end{aligned} \tag{4}$$

in which the solution space is a subset of $SE(3)$ for any given object.

In the *online* variant of the 3D-BPP, items arrive sequentially and must be packed immediately, without knowledge of future items. This setting reflects practical scenarios in which the item stream is dynamically generated and prior sorting is unfeasible. In contrast, the classical *offline* 3D-BPP assumes full knowledge of all items beforehand, allowing for sorting strategies that can significantly improve packing efficiency [4].

Different approaches have been developed to deal with the 3D-BPP, progressing from axis-aligned cuboids (namely, parallelepipeds whose edges are parallel to the coordinate axes) to the placement of highly irregular meshes. In particular, different modeling techniques have been adopted to represent the generic object $\mathcal{O}_i$ and the space included in $\mathcal{B}$.

2.1 | Object and Space Representation

The first approaches tackled a simplified problem dealing with cuboid-shaped objects [5–13]. Under this assumption, only 3 parameters are needed to describe the object, namely its principal dimensions: length, width, and height. Reinforcement-learning approaches have been proposed for the online packing of cuboids, where packing decisions are learned through sequential interaction with the environment. In this line of work, the packing task is formulated as a Markov decision process and placement policies are learned using deep reinforcement learning [14]. Subsequent work [15] extends this framework by incorporating stability constraints and robotic feasibility considerations. More recently, explicit planning approaches have been explored [16], formulating the problem as a search over packing configuration trees rather than relying on learned policies, and achieving promising results.

The cuboid-packing model finds suitable applications in logistic problems dealing with pallet stacking, or ship and truck loading contexts. However, the efficacy of this representation falls short when dealing with objects whose geometry sways from box-like designs.

To overcome the limitation of cuboid-shaped modeling, different representation techniques may be adopted. Heightmaps (HMs), similarly to depthmaps, contain information related to the distance of an object from a reference plane [17]. A HM can be formalized as a 2D grid in which each cell contains a single depth value, capturing exactly one visible face of a 3D object. Depth is treated as a continuous variable along the line normal to the viewing plane, while the remaining two axes are discretised according to the grid's spatial resolution. By only encoding information related to visible features from a viewing plane, opposite faces and hidden features remain unknown and must be inferred or ignored.

Object voxelization turns the space containing the object into a 3D grid, where each point is assigned a binary value depending on the corresponding voxel being included or not in the object [18]. Although this structure permits fast volume evaluation

and collision checks through simple counting and logical operations, its main limitation lies in memory consumption: the number of voxels, and thus the storage required, grows cubically as linear resolution is refined, leading high-resolution models to be represented by high-dimensional, sparsely populated tensors that place a considerable burden on memory resources.

The memory overhead can be partially alleviated by octree-based spatial partitioning [19], particularly through adaptive octrees. An adaptive octree is a hierarchical data structure that recursively decomposes 3D space into octants, refining only those octants that contain parts of the object while leaving empty regions at a coarser level.[1] This non-uniform discretization allows for efficient representation and processing of complex geometries while minimizing memory usage in empty areas.

HMs, voxelization, and octrees all discretize space on a fixed grid. A limiting aspect of these representations is their inability to accommodate rigid-body rotations: any change in the object orientation requires either a complete recomputation of the related grid values or an alternative modeling strategy.

To handle arbitrary rotations, objects can be represented as point clouds, that is, sets of surface-sampled points [20]. Since each point is described in Cartesian coordinates, rotation can be easily applied by using the appropriate rotation matrix. However, the representation accuracy depends on the number of sampled points, and the surface between the latter cannot be directly inferred, requiring different strategies to define object volumes and check for collisions.

To overcome the latter issue, mesh-like structures composed of oriented polygons can be used to model the object surface, thereby forming a piecewise-linear approximation of the geometry between sampled points, namely the mesh vertices [21]. While overcoming point-cloud limitations, the polygon count scales rapidly with the number of sampled points, preventing dense meshes from being used. Moreover, classical meshes describe shape, but not how that shape is embedded in space, so that they do not allow answering spatial queries related to the distance of the object from other surfaces, unless extra data structures are added.

Unlike *explicit* representations such as meshes or point clouds, which store geometric information directly through vertices, faces, and connectivity, *implicit* representations define shapes as the zero-level set of continuous mappings. A common and powerful implicit representation is the SDF [22], which maps each point in space to its signed distance (SD), that is, the shortest distance from the object's surface, with the negative or positive sign indicating whether the point respectively lies inside or outside the object volume, and the zero-level set representing the surface boundary. This formulation provides smooth gradients, enables efficient surface queries, and is well-suited for optimization and integration into differentiable systems such as neural networks. A limitation of the SDF representation, however, is the inability to directly apply rotations to the model.

Further information on object modeling can be found in [23], which reviews different strategies for 3D object representations.

The representations discussed in this subsection can also be employed to model the space included in $\mathcal{B}$. Since most packing algorithms consider one object at a time, a representation that includes information on both $\mathcal{B}$ and the already placed objects is necessary. This is usually achieved by limiting the space description to $\mathcal{B}$, and leveraging heightmaps, voxelization, or SDFs to describe occupancy.

2.2 | 3D-BPP

Cuboid-based packing exploits symmetries to reduce the search space, relying on axis-aligned configurations and reducing the number of considered rotations to at most 6. This leads to mixed-integer combinatorial problems that reduce the (translational) continuous solution space to $\mathbb{R}^3$, and approach the rotational aspect as a combinatorial problem for a single object. Leveraging this simplified representation, exact solutions to the *offline*, axis-aligned cuboid-based problem have been found through non-deterministic, polynomial-time algorithms [5]. Nonetheless, the time constraints of practical applications led the scientific community to leverage the simplified representation to design time-efficient, heuristic-driven algorithms. Fleszar [6] developed a mixed-integer linear programming (MILP) algorithm to address the BPP problem with conflicts and item fragmentation: heuristics are used to define a first-attempt solution that accommodates the specific constraints of the application, while the MILP algorithm is used to refine the solution. Static stability has been addressed by Le Jean et al. [9], introducing balancing constraints to the 3D-BPP to ensure stacking and transportation feasibility. Kagerer et al. [24] proposed BED-BPP, a benchmarking dataset for cuboid 3D-BPPs, designed to evaluate algorithms under real-world constraints using industrial grocery logistics data.

However, approximating the heterogeneity represented by the geometries of items in an e-commerce setting to cuboid-shaped items shows several limitations: the actual static stability of the items cannot be assessed due to the unrealistic model, only a limited number of rotations is considered when looking for a transformation, and space occupancy is largely overestimated, leading to suboptimal configurations.

Romanova et al. [25] proposed a nonlinear optimization approach to pack concave polyhedrons into a cuboid of minimal volume, using radical-free quasi-phi-functions [26] to model non-overlap, containment, and distance constraints. Cui et al. [27] presented an application of voxelization to the 3D-BPP, leveraging a spectral analysis of the voxelized grid for fast occupancy detection. However, both works [25] and [27] disregard the static stability of final and intermediate configurations, making them more suited to 3D-printing applications, where a supporting structure can be printed. Zhuang et al. [28] adopted the occupancy-detection technique from [27], combining it with the Deepest-Bottom-Left-Fill (DBLF) heuristic [29] to generate initial solutions in a discrete search space, which were then refined using a mesh-based physics simulator to improve stability and packing efficiency.

Other approaches addressed the packing of irregular geometries via constructive heuristics paired with suitable modeling strategies. Wang et Hauser [30] employed a non-increasing bounding-box heuristic to sort items, followed by a score metric computed by combining the object HM from an underneath plane and the bin HM from an overhead plane. Pan et al. [31] made use of GPU-accelerated computation of truncated SDFs to represent the bin

and already-placed objects, which are then paired to object HMs to rank placements.

In recent years, due to the complexity of the problem, the adoption of deep learning (DL) approaches has been growing. Liu et al. [32] trained a DL algorithm to infer sequences and optimal rotations among a limited set: objects were represented by extending a sparse point cloud to box-like volumes around points; optimal placement was then computed according to a heuristic aimed at maximizing the ratio between items' volumes and their circumscripted box volume. In [33], Zhao et al. made extensive use of reinforcement learning paired with HM representations of both objects and the bin to score a set of placements defined according to a specifically designed heuristic, while a mirroring network was used to infer an optimal sequence of items to be placed.

2.3 | Paper Contributions

This paper advances the 3D-BPP literature by addressing key challenges in packing operations. Existing methods that handle diverse item shapes while ensuring post-placement stability typically discretize the operation space and restrict item orientations to a limited set of rotations, thereby reducing problem dimensionality. On the contrary, this paper tackles the 3D-BPP by designing an optimization problem in a continuous subset of $SE(3)$, capable of handling a variety of different primitive shapes. The optimization aim is to find the objects' configuration $\mathcal{T}$ that maximizes the packing utility. Our algorithm leverages the efficiency of SDFs spatial queries to drive the optimization problem, using neural-network approximations to further enhance computation speed; occupancy of the bin is instead modeled through a point-cloud representation, exploiting its suitability to handle continuous rotations and overcoming the limitations of the SDF model. Finally, a physics simulator is used to ensure stability and improve the tightness of the solution, thus increasing packing utility.

These characteristics enable the algorithm to search for item transformations over arbitrary subsets of $SE(3)$, evaluate static stability after each placement, and maintain manageable computational cost. Moreover, the algorithm flexibility makes it suitable for both *offline* and *online* packing, with appropriate distinctions. In particular, in the latter problem, real physical interactions with the objects, such as gravity and contact forces from a robotic manipulator, replace the physics simulator.

The presented approach is tested in simulation with different combinations of primitive shapes, such as cylinders, prisms, and pyramidal frusta of arbitrary dimensions. The complete set of object models and simulation results is publicly accessible in [34].

3 | Model

We assume that a triangular-mesh representation is available for each object to be packed. In the *offline* setting, a heuristic is first applied to order the item set, whereas no such pre-sorting is possible in the *online* variant. Once the items are ordered (or arrive sequentially in the *online* setting), a bin representation is generated, together with a model for each item. Then, N optimization sub-problems are solved, each restricted to a distinct portion of the overall search space; the solutions to each sub-problem are sorted according to a performance metric. Solution feasibility is verified, and the first feasible solution is chosen as optimal; if no valid solution is found, a refined optimization process takes place near promising solutions. The process is executed successively for each item, with the bin representation updated after each placement.

3.1 | Item Model

At different stages of the algorithm, different object representations are used:

- $\mathcal{O}_i^M$ is the mesh itself;
- $\mathcal{O}_i^P$ is a point cloud sampled on the mesh surface;
- $\mathcal{O}_i^{SDF}$ is a SDF representation of the object.

$\mathcal{O}_i^M$ is used to extract $\mathcal{O}_i^P$ and $\mathcal{O}_i^{SDF}$, and is also employed within the physics simulator during *offline* packing.

$\mathcal{O}_i^P$ is employed in the *offline* settings, where it is used to update the bin representation (see Section 3.2). Points are generated by uniformly sampling the triangular faces, with the number of points per face weighted proportionally depending on the face area.

$\mathcal{O}_i^{SDF}$ is used during the optimization process to efficiently compute SDs (see Section 3.4). $\mathcal{O}_i^{SDF}$ is built according to [35], employing a multi-layer perceptron (MLP) neural network to approximate the SDFs of each item. The model is trained before starting the optimization. In this way, the ability of MLPs to approximate any continuous function [36] is leveraged to fit $\mathcal{O}_i^{SDF}$ and thus generate a *continuous* SDF that can be queried multiple times efficiently through GPU vectorization capabilities (see Section 3.4).

The network training is conducted for each object in a supervised fashion, with a precomputed ground-truth dataset. The dataset is composed by the union of two sets of labeled points, characterized by different sampling criteria. Labels are the SDs values computed from the considered points to $\mathcal{O}_i^M$, normalized with respect to an arbitrary maximum bin dimension b_{max} to stabilize training [37].

The two sampling criteria, one uniform and one distance-based, select points within a cubic region of side length $2b_{max}$, centered at the geometric barycenter of the object. The *uniform* sampling uses a grid of n_{train} equidistant points. The *distance-based* sampling builds an adaptive octree around the object. The octree is progressively refined as the center of each octant gets closer to the item surface, up to a maximum tree depth w_{oct}. For each leaf of the octree, its center is sampled and its (normalized) SD is used as a label. Conditions for octree subdivision after the first one are evaluated based on the SD values computed on the corners of each octant, namely $\boldsymbol{d}_C = [d_1, \ldots, d_8]$. Subdivision is carried out if either of these conditions is verified:

$$\min(\boldsymbol{d}_C) \cdot \max(\boldsymbol{d}_C) < 0 \tag{5a}$$

$$\min(|\boldsymbol{d}_C|) < \frac{\sqrt{3}l}{2} \tag{5b}$$

in which l is the side-size of the considered octant, and $|\boldsymbol{d}_C|$ refers to element-wise modulus. Equation (5a) identifies octants inside which the SD changes sign, meaning that the object boundaries are intersecting the octant, and a higher-density grid is required to achieve better accuracy on object surfaces. Equation (5b) ensures that at the end of the process the half-diagonal of the octant is smaller than the minimum absolute value of the corner SDFs, leading to a higher density of sampled points near the object boundaries; subdivision under this condition is carried out only if $\min(|\boldsymbol{d}_C|) < d_{thresh}$, with d_{thresh} being a suitable threshold that avoids excessive refinement far away from the object surface.

The union of the two sets provides high precision near the object's surface while keeping the size of the dataset limited. Relying only on uniform sampling would lead to high-density points in regions far from the object, where precision is less critical. Conversely, using only distance-based sampling would lead to a sparse dataset in distant regions, resulting in unreliable approximations when SDs grow larger.

This dataset is then used to train an MLP network for each object. Each MLP consists of 3 hidden layers, each containing 512 nodes. A Rectified Linear Unit (ReLU) activation function (AF) [38] is applied to every hidden layer, followed by weight normalization [39]. The output layer uses a hyperbolic-tangent AF, binding the predicted value to $[-1, 1]$. During optimization (see Section 3.4), values are multiplied by the normalization factor b_{max} to obtain the actual SD. Since the MLP is a composition of linear maps and continuous AFs, the resulting mapping $f : \mathbb{R}^3 \mapsto [-1, 1]$ is continuous. This feature allows SDs to be computed at any point in $\mathbb{R}^3$, regardless of the object orientation. Another advantage of neural-network representations is given by the PyTorch library [40], which allows for batched inference of SDs, greatly reducing per-point SD computation time. Figure 1 shows a reconstruction of the zero-level sets extracted from the SDF-based representation of three objects, computed using the described method.

The loss function used during training is a mean-square-error loss. Network training is carried out with a batch size of 32 points, and stopped when reaching 200 epochs or a validation-loss value under $2E-6$ per batch. Parameters are updated through Adam optimizer [41], with a learning rate of 0.001. Object models, such as trained models, meshes, and point clouds, are publicly available in [34].

3.2 | Bin Model

If $\mathbf{p} \in \mathbb{R}^4$ is the homogeneous representation of a point, the current bin $\mathcal{B}$ is represented by a point cloud $\mathbf{B} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$ of n points sampled from the solid volumes pertaining to both the bin and the items inside it. The point cloud is updated each time an additional item is placed in the bin. When dealing with an *offline* setting, the initial point cloud $\mathbf{B}^1$ is generated by regularly sampling points in a grid spanning the bottom and lateral surfaces of the bin. The point-cloud reference frame is defined with its origin at the center of the bin's bottom rectangle, its xy-plane coinciding with the bottom surface, and the z axis pointing upward. If $\mathbf{B}^i = \{\boldsymbol{p}^i_1, \ldots, \boldsymbol{p}^i_{n_i}\}$ is the point cloud representing $\mathcal{B}$ before object $\mathcal{O}_i$ is placed in the box, $\mathbf{B}^i$ is generated by the union of $\mathbf{B}^1$ with the point clouds of the objects already placed in the bin $\{\mathbf{T}^1\mathcal{O}^P_1, \ldots, \mathbf{T}^{i-1}\mathcal{O}^P_{i-1}\}$. In the *online* context, a depth camera is assumed to be positioned on top of the bin, through which a depth image is acquired any time a new object has to be inserted in the bin.

In both cases, the resulting point cloud represents the detected surfaces, not the underlying volumes. To represent filled volumes and occluded spaces (which would be unreachable when considering a top-down object insertion), additional points are added to the original cloud: they share the same x and y coordinates as the surface points, but have z coordinates linearly sampled from $z = 0$ up to the corresponding surface height, using a step size of ε.

It is to be noted that the model does not include the upper surface of the bin, that is, its lid. As explained in Section 3.4, the optimization process aims at maximizing the number of points in contact with the object, and the presence of a lid would cause the algorithm to return undesirable poses adjacent to it.

3.3 | Objective Function

The goal of the i-th optimization subproblem ($i = 1 \ldots N$) is to find the transformation $\mathbf{T}_i$ that, when applied to $\mathcal{O}_i$, minimizes a loss function $L(\mathbf{B}^i, \mathbf{T})$:

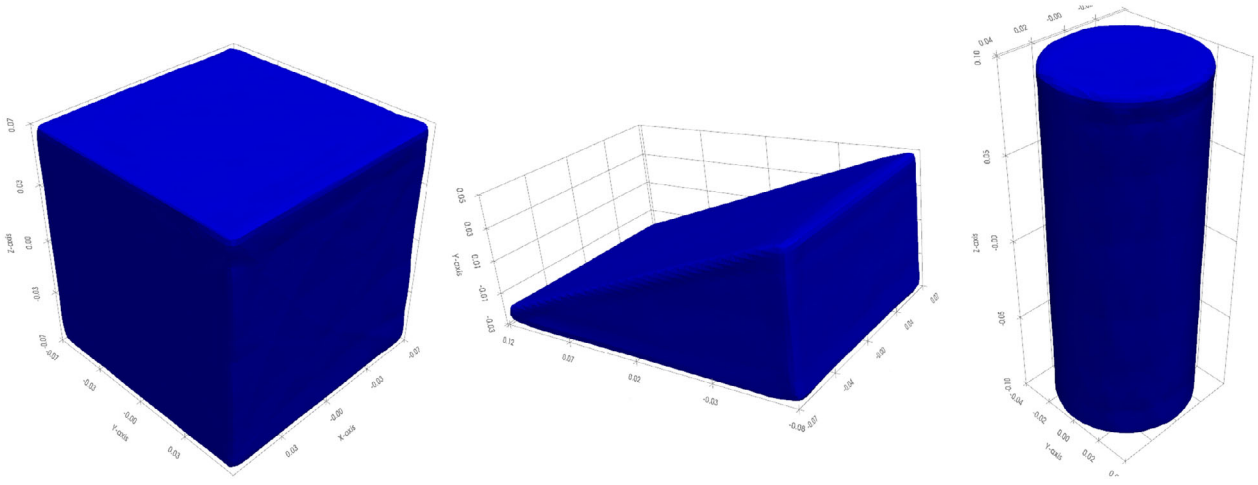


FIGURE 1 | Examples of shapes reconstructed from their SDF representation.

$$\mathbf{T}_i = \arg\min_{\mathbf{T}} \quad L(\mathbf{B}^i, \mathbf{T}) \tag{6}$$

where $\mathbf{T}$ is a generic rigid trasformation in $SE(3)$.

The loss function is designed according to a heuristic aimed at maximizing space utilization. Specifically, each object is assumed to be placed as close as possible to the surrounding surfaces of $\mathbf{B}^i$, thereby maximizing the number of contact points and minimizing the distances to neighboring surfaces. For this purpose, the loss function is formulated by evaluating the SD of points $\boldsymbol{p}_k^i$ in $\mathbf{B}^i$ to the transformed object, and summing all contributions. To regularize loss values among different optimization instances (as index i varies), the total sum is normalized by the number of points n_i in $\mathbf{B}^i$, yielding an average loss value. Accordingly, if $SD(\boldsymbol{p}_k^i, \mathcal{O}_i)$ is a function returning the SD of point $\boldsymbol{p}_k^i$ to $\mathcal{O}_i$, the overall loss function is:

$$L(\mathbf{B}^i, \mathbf{T}) = \frac{\sum_{k=1}^{n_i} \ell(SD(\boldsymbol{p}_k^i, \mathbf{T}\mathcal{O}_i))}{n_i} \tag{7}$$

where $\ell(SD(\boldsymbol{p}_k^i, \mathbf{T}\mathcal{O}_i))$ is a per-point loss function.

To efficiently compute SDs, we use the neural-network–based continuous approximation of O_i^{SDF} described in Section 3.1. This model enables the computation of the SD from $\boldsymbol{p}_k^i$ to $\mathbf{T}\mathcal{O}_i$ for any transformation $\mathbf{T}$ within a continuous subset of $SE(3)$. However, since applying rigid-body transformations directly to SDF models is nontrivial, we instead apply the inverse transformation $\mathbf{T}^{-1}$ to the bin point cloud. This allows us to compute SDs from the transformed points to the untransformed object, rather than the other way around, namely:

$$\ell(SD(\boldsymbol{p}_k^i, \mathbf{T}\mathcal{O}_i)) = \ell(SD(\mathbf{T}^{-1}\boldsymbol{p}_k^i, \mathcal{O}_i)) \tag{8}$$

$\ell(x)$ is a piecewise conditioning function designed to:

- be continuous in $\mathbb{R}$,
- exhibit a steep derivative when $x < 0$,
- attain a strict global minimum when $x = 0$,
- become stationary when $x > q$, with q being a predefined scalar threshold.

In particular, $\ell(x)$ is chosen as (Figure 2):

$$\ell(x) = \begin{cases} ax^2 & \text{if } x < 0 \\ sin[\frac{\pi}{q}(x - 0.5q)] + 1 & \text{if } 0 \le x \le q \\ 2 & \text{if } x > q \end{cases} \tag{9}$$

Parameter a controls the penalty severity for points inside the object ($x < 0$), and is selected to make the per-point loss steep enough to effectively penalize interpenetration, while still allowing other points to contribute meaningfully to the total loss. If a is too large, even slight interpenetration at a single point can produce disproportionately high loss values, overshadowing the influence of other points. Conversely, if a is too small, the loss may permit excessive interpenetration.

Parameter q defines a distance beyond which ℓ becomes flat, thus setting a spatial threshold around the object, allowing only points within this range to influence the optimization. Points beyond q contribute a zero gradient and thus do not affect the final result.

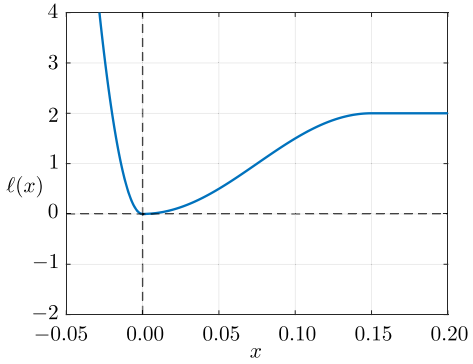


FIGURE 2 | Piecewise conditioning function $\ell(x)$ for $q = 0.15$ and $a = 5000$.

In other words, q ensures that the object is attracted mainly to nearby points, while those beyond q have no influence on its final configuration. To further penalize points of $\mathbf{B}^i$ lying within $\mathbf{T}\mathcal{O}^i$, a small negative offset c is added to the SD values before conditioning. This shifts the minimum of the per-point loss function slightly outside the object's surface, ensuring that configurations near contact are preferred over those leading to interpenetration. The tuning of parameters a, q and c is discussed in Appendix A.1.

The overall loss function becomes:

$$L(\mathbf{B}^i, \mathbf{T}) = \frac{\sum_{k=1}^{n_i} \ell[SD(\mathbf{T}^{-1}\boldsymbol{p}_k^i, \mathcal{O}_i) + c]}{n_i}, \quad c \le 0 \tag{10}$$

It is worth emphasizing that the designed loss function implements a soft, rather than a hard constraint, since interpenetration is penalized but not strictly prohibited. This has two key implications. First, it allows the optimizer to explore candidates near the constraint boundaries. Second, it requires a post-verification step to assess the feasibility of the proposed solutions and discard those allowing interpenetration.

Figure 3 shows the loss distribution obtained by plac ing a cube of side-length 0.1 m into a bin of size $\mathcal{B} = [-0.1, 0.1] \times [-0.1, 0.1] \times [0, 0.15]$ m, and evaluating rigid transformations restricted to the set $\{\mathbf{T} | \boldsymbol{t} = [x, y, z, 0, 0, 0]\}$, where x, y, z are the coordinates of the cube geometric center, $z = 0.05$ m, and x and y are linearly sampled in the interval $[-0.2, 0.2]$ m with a stepsize of 0.001 m. With the loss parameters fixed at $a = 5000$ m^{-2}, $q = 0.022$ m, and $c = 0$ m, the resulting loss map (yellow = low, red = high) reaches its global minimum of $L_{min} = 1.835$ at $\boldsymbol{t} = [0.0466, 0.0496, 0.05, 0, 0, 0]$. The evaluation was carried out using a point cloud of $n = 62500$ points to represent the bin. A notable feature of the loss landscape is the presence of multiple local minima, which results in the lighter-colored vertical bands visible in the zoomed-in view of the top-right quadrant shown in Figure 3b.[2]

3.4 | Optimizer and Optimization Variables Conditioning

To solve the optimization problem in Equation (6), we adopt the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [42], as implemented in the EvoTorch library [43]. CMA-ES is a

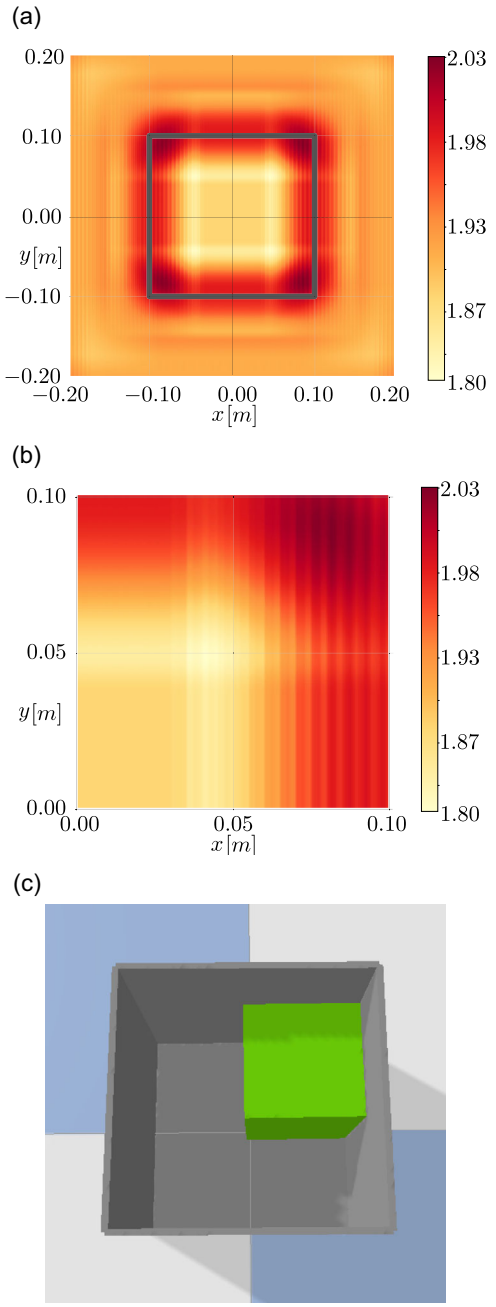


FIGURE 3 | Loss distribution and optimal placement of a cube in a box: (a) color map of the loss function (yellow = low, red = high) and bin boundaries (in gray); (b) zoomed-in view of the top-right quadrant of the bin, where the minimum loss is located; (c) optimal location of the cube in the top-right quadrant of the bin, where minimum loss is achieved.

stochastic, derivative-free, evolutionary optimization algorithm, in which multiple candidate solutions are evaluated at each iteration. The candidates for subsequent iterations are sampled from a probability distribution adapted from the loss values of previous generations.

The adoption of CMA-ES, as opposed to conventional gradient-based strategies, is motivated by its ability to effectively navigate the complex loss landscape induced by the sparsity of the point cloud, often characterized by numerous local minima (see the example in Figure 3). Moreover, as a derivative-free method, it does not require gradient computations and relies exclusively on loss-function evaluations for each candidate solution. By leveraging the GPU-based neural-network inference described in Section 3.1, these evaluations can be fully vectorized, allowing the parallel processing of multiple candidates and significantly accelerating each optimization step.

Since the optimization procedure is identical for each object to pack, the subscript/superscript i is omitted in this section to improve readability. Algorithm 1 reports a simplified representation of the procedure solving the optimization problem in Equation (6). The optimization variable $\boldsymbol{g}_{m,q} \in \mathbb{R}^6$ represents the m-th candidate solution in generation q, and is the input of a multivariate normal distribution. The maximum number of generations Q and the maximum number of candidates per generation M are the two main parameters of the optimization. First-generation candidates $\boldsymbol{g}_{m,1}$ are sampled according to a normal distribution characterized by an identity covariance matrix $\boldsymbol{C}$ and a null mean vector $\boldsymbol{\mu} = [0, 0, 0, 0, 0, 0]$. As such, each candidate element is sampled from $-\infty$ to $+\infty$, which is incompatible with the parameters in $\boldsymbol{t}$. Accordingly, each candidate $\boldsymbol{g}_{m,q}$ is decoded into the corresponding transformation variables $\boldsymbol{t}_{m,q}$ in two steps:

- first, each element of $\boldsymbol{g}_{m,q}$ is mapped onto a $[-1, 1]$ continuous set through a hyperbolic tangent function:

$$\boldsymbol{g}_{\text{norm}} = \tanh \boldsymbol{g}_{m,q} \tag{11}$$

- then, the result is linearly mapped onto a constrained continuous set $\{\boldsymbol{t} | \boldsymbol{t}_l \le \boldsymbol{t} \le \boldsymbol{t}_u\} \subset SE(3)$ in which $\boldsymbol{t}_l$ and $\boldsymbol{t}_u$ identify, respectively, a lower and upper bound for the transformation variables:

$$\boldsymbol{t}_{m,q} = (\boldsymbol{g}_{\text{norm}} + \mathbf{1}) \odot \frac{\boldsymbol{t}_u - \boldsymbol{t}_l}{2} + \boldsymbol{t}_l \tag{12}$$

in which $(\odot)$ represents element-wise multiplication, and $\mathbf{1}$ is a column vector of ones.

ALGORITHM 1 | Optimization procedure.

Require: $\mathbf{B}, \boldsymbol{g}_{m,1}, \boldsymbol{t}_u, \boldsymbol{t}_l, Q, M$

$\mathcal{P} \leftarrow \{\}$

for $q \leftarrow 1$ to Q do

  for $m \leftarrow 1$ to M do

    $\boldsymbol{g}_{\text{norm}} \leftarrow \tanh \boldsymbol{g}_{m,q}$

    $\boldsymbol{t}_{m,q} \leftarrow (\boldsymbol{g}_{\text{norm}} + 1) \frac{\boldsymbol{t}_u - \boldsymbol{t}_l}{2} + \boldsymbol{t}_l$

    $\mathcal{P} \leftarrow \mathcal{P} \cup \mathbf{T}_{m,q}^{-1} \mathbf{B}$

  end for

  $[\boldsymbol{g}_{1,q+1}, \ldots, \boldsymbol{g}_{M,q+1}] \leftarrow$ CMA-ES($BatchedLoss(\mathcal{P})$)

end for

$\boldsymbol{g}^* \leftarrow \arg\min_{\boldsymbol{g}} (L(\mathbf{B}, \boldsymbol{t}^g)), \boldsymbol{g} \in [\boldsymbol{g}_{1,1}, \ldots, \boldsymbol{g}_{Q,M}]$

$\boldsymbol{g}^*_{\text{norm}} \leftarrow \tanh \boldsymbol{g}^*$

$\boldsymbol{t}^* \leftarrow (\boldsymbol{g}^*_{\text{norm}} + 1) \frac{\boldsymbol{t}_u - \boldsymbol{t}_l}{2} + \boldsymbol{t}_l$

This process not only ensures compatibility of the solution with the domain of the chosen parameterization, but also allows the solution space to be restricted to an arbitrary subset of $SE(3)$, possibly excluding undesirable transformations from the solution space, such as those considered unfeasible by manipulation constraints.

The transformation parameters $\boldsymbol{t}_{m,q}$ are then used to find the transformed point cloud of the bin $\mathbf{T}_{m,q}^{-1}\mathbf{B} = \{\mathbf{T}_{m,q}^{-1}\boldsymbol{p}_1, \ldots, \mathbf{T}_{m,q}^{-1}\boldsymbol{p}_n\}$ for loss computation. Based on the loss values of the current generation, the optimizer updates the distribution parameters $\boldsymbol{\mu}$ and $\boldsymbol{C}$ to sample the next generation. This process is repeated for Q generations. The candidate $\boldsymbol{t}_{opt}$ that yields the lowest loss value across all generations is selected as the optimal solution.

To optimize the use of computational resources, and thus reduce time, SD inference and loss computation are executed in batches after merging the transformed point clouds of all candidates in a generation into a single list of points $\mathcal{P}$. Batch extraction is detailed in Algorithm 2, where:

- K is the number of points per batch, chosen so that inference of SDs of each batch maximizes GPU memory usage;
- $H = \lfloor nM/K \rfloor$ is the number of batches;
- $\mathcal{P}^h$ is the list of points in the h-th batch;
- $\boldsymbol{s}^h$ is the list of SD values related to the points in $\mathcal{P}^h$, inferred via the object SDF;
- $\boldsymbol{\ell}(\boldsymbol{s}) = [\ell(s), \quad \forall s \in \boldsymbol{s}]$ is the list of per-point losses computed across all SD values in $\mathbf{s}$, through Equation (9);
- ℓ_{pp} is the list of all per-point loss values across a generation;
- $\boldsymbol{\ell}_m = \ell(SD(\mathbf{T}_m^{-1}\mathbf{B}, \mathcal{O}))$ is the list of per-point losses related to the m-th candidate transformation, obtained after regrouping the elements of ℓ_{pp} according to the respective candidate;
- L_m is the overall loss function of the m-th candidate transformation, computed through Equation (10).

ALGORITHM 2 | Batched loss computation.

Require: $\mathcal{P}, M, K, n$

$\boldsymbol{\ell}_{pp} \leftarrow \{\}$

$H \leftarrow \lfloor nM/K \rfloor$

for $h \leftarrow 0$ to $H-1$ do

$\mathcal{P}^{h+1} \leftarrow \mathcal{P}[\,hK : (h+1)K\,]$

$\boldsymbol{s}^{h+1} \leftarrow [SD(\boldsymbol{p}, \mathcal{O}) | \boldsymbol{p} \in \mathcal{P}^{h+1}]$

$\boldsymbol{\ell}_{pp} \leftarrow [\boldsymbol{\ell}_{pp}; \ell(\boldsymbol{s}^{h+1})]$

end for

$\mathcal{P}^H \leftarrow \mathcal{P}[\,HK : nM\,]$

$\boldsymbol{s}^H \leftarrow [SD(\boldsymbol{p}, \mathcal{O}) | \boldsymbol{p} \in \mathcal{P}^H]$

$\boldsymbol{\ell}_{pp} \leftarrow [\boldsymbol{\ell}_{pp}; \ell(\boldsymbol{s}^H)]$

for $m \leftarrow 0$ **to** $M-1$ **do**

$\boldsymbol{\ell}_{m+1} \leftarrow \boldsymbol{\ell}_{pp}[\,mn : (m+1)n\,]$

$L_{m+1} \leftarrow \sum_{j=1}^{n} \boldsymbol{\ell}_{m+1}[j]/n$

end for

return $[L_1, \ldots, L_M]$

The algorithm thus allows the optimal transformation $\boldsymbol{t}_{opt}$ to be obtained in only QH parallelized computations. The influence of parameters M and Q on packing performance and computational time is discussed in Appendix A.2.

4 | Packing-Algorithm Pipeline

We first describe the *offline* packing pipeline, which is adapted to the *online* setting in Section 4.5. In the *offline* scenario, the packing algorithm proceeds according to the following steps:

1. **Initialization**: the items to be packed are sorted according to a given heuristic, and the model $\mathbf{B}^1$ of an initial bin is generated. The initial bin size is chosen according to a heuristic discussed in Section 4.1.
2. **Coarse search**: for each item, the optimization problem described in Section 3.4 is divided in 18 smaller subproblems. In each one of them the solution space is limited to an interval $[\boldsymbol{t}_l, \boldsymbol{t}_u]$ (see Section 3.4) to increase the likelihood of finding a feasible solution and decrease computational time. The solutions to the 18 subproblems are then ranked and sequentially tested for feasibility. The first feasible placement is accepted and the pipeline resumes at Step 4, otherwise the algorithm proceeds to Step 3.
3. **Refined search**: if all coarse candidates are infeasible, they are re-ranked by feasibility metrics and iteratively refined by instantiating additional optimization problems. This refinement uses a higher-resolution, localized bin representation centered on the coarse candidate, and limits the search space to a neighborhood of the considered coarse solution. If all the solutions to the refined problems are still infeasible, packing is aborted.
4. **Physics-based placement**: once a feasible placement is found, a top-down insertion is simulated in a physics engine, and an adjustment force is applied to improve compactness. After the simulation stabilizes, the new item configuration is recorded.
5. **Bin-state update**: the bin representation is updated, and the pipeline proceeds to the next item.

The packing attempt is considered successful when all items are packed, or unsuccessful when no feasible placement can be found for the current item. Once a packing attempt is successful, the loop is reinitialized with a smaller bin, whose dimensions are computed according to the current packing configuration. The new bin dimensions are initialized as the Axis-Aligned Bounding Box (AABB).[3] dimensions of the packed configuration multiplied by a factor of 1.1; the height is also increased by an additional 0.02 m. The slightly increased bin size with respect to the AABB dimensions allows the items to assume different configurations when compared to the previous iteration. In the case that the very first iteration is unsuccessful, the bin dimensions are increased by 0.01 m, and the loop is reinitialized until a successful attempt is found. Iterations on bin sizes stop

when either a packing attempt other than the first is unsuccessful or the AABB volume of the new configuration is bigger than the AABB volume of the previous one. The final item configuration is the one that reached the minimum AABB volume while being successful, and its AABB is chosen as the minimum bin size capable of containing the item.

4.1 | Initialization

The initialization includes item pre-sorting and the generation of $\mathbf{B}^1$. There are different methods used for sorting items in the literature, mainly through heuristics and the use of learned representations [44]. In this work, a purely geometric heuristic is adopted, ranking objects by the sum r of their axis-aligned extents. For each model $\mathcal{O}_i^P$, r is computed as:

$$r = (x_{\max} - x_{\min}) + (y_{\max} - y_{\min}) + (z_{\max} - z_{\min}) \tag{13}$$

where $(x_{\min}, x_{\max}, y_{\min}, y_{\max}, z_{\min}, z_{\max})$ are the extreme coordinates of points in the object's point cloud $\mathcal{O}_i^P$. Because these values depend on the used coordinate frame, a canonical reference frame is defined whose origin lies at the point that minimizes the maximum distance ρ_{max} from $\mathcal{O}_i^M$ vertices, that is, its Chebyshev center[4] [45], and whose axes coincide with the principal directions of the object inertia tensor. Items are then sorted in descending order of r, so that larger values receive higher priority. This follows from the largest-base-dimensions criterion [46], which ranks objects by the size of their supporting area to maximize stacking stability. Because the item orientation is decided only during placement, prioritizing objects with larger r-scores allocates floor area first to those that could present the widest supporting footprint, preventing the packing surface from being prematurely fragmented by smaller pieces.

The bin size for the first iteration is determined based on the volumes $|\mathcal{O}_i|$ of all items in the set and the maximum ρ_{max} recorded among all items (ρ^*_{max}). In particular, it is chosen as a cube with volume $8\sum_i V_i$ and side no smaller than ρ^*_{max}:

$$L = W = H = \max\left(2\sqrt[3]{\sum_i |\mathcal{O}_i|}, \rho^*_{max}\right) \tag{14}$$

In accordance with the item model described in Section 3.1, a maximum threshold equal to b_{max} is also imposed to bin dimensions, so that the bin size is compatible with the trained item representation. It should be noted that, in most real-world applications, only a discrete set of bin sizes is available. In such cases, iterations should begin from the smallest bin that satisfies the above conditions. Bin point cloud $\mathbf{B}^1$ is generated in accordance with Section 3.2.

4.2 | Coarse Search

After sorting, the first item is passed to the *coarse search*. To accelerate this step, the bin point cloud $\mathbf{B}^i$ is downsampled through a voxel-grid filter [47] to a subset $\mathbf{B}^i_d$, using a voxel size equal to $\Gamma \cdot \rho_{max}$; the lower resolution, and thus the lower number of inferences needed to compute SDs, allows for a faster optimization step. The influence of parameter Γ on the algorithm performance is discussed in Appendix A.2. The dependence of the voxel size on ρ_{max} prevents oversampling of larger objects and excessive downsampling of smaller ones. The bin base area is then partitioned in a 3×3 grid, with the center of the j-th cell being (x_c^j, y_c^j), $j = 1, \ldots, 9$. If $\mathbf{\Delta}_1^T = [\mathbf{\Delta}_{1,t}^T, \mathbf{\Delta}_{1,r}^T]$ is a 6-element range-limiting vector, with $\mathbf{\Delta}_{1,t}^T = [\frac{L}{6}, \frac{W}{6}, \frac{H}{2}]$ limiting translations and $\mathbf{\Delta}_{1,r}^T = [\pi, \frac{\pi}{4}, \pi]$ limiting rotations, and $\theta_1 = 0$ and $\theta_2 = \frac{\pi}{2}$ are two preferred tilt angles, for every pair (j,k), with $k = 1, 2$, a search space can be defined whose center is $\boldsymbol{t}_c^{j,k} = [x_c^j, y_c^j, \frac{H}{2}, 0, \theta_k, 0]^T$ and whose lower and upper boundaries are:

$$\boldsymbol{t}_l^{j,k} = \boldsymbol{t}_c^{j,k} - \mathbf{\Delta}_1, \quad \boldsymbol{t}_u^{j,k} = \boldsymbol{t}_c^{j,k} + \mathbf{\Delta}_1 \tag{15}$$

This gives 18 search spaces that jointly cover the whole bin.[5] Given the loss function defined in Equation (10), considering all points in $\mathbf{B}^i$ would lead to many unnecessary calculations. Specifically, the points that lie farther than $\delta = \rho_{max} + q - c$ from the boundaries of the translational part of the search space, with ρ_{max} being the maximum distance of any point of the object from the origin of its reference frame, cannot contribute to the shape of the loss function. Accordingly, the computation of the per-point loss value is limited to points included in the set $\mathbf{B}^i_d \cap [\boldsymbol{b}_l^j, \boldsymbol{b}_u^j]$, in which:

$$\begin{aligned} \boldsymbol{b}_l^j &= [x_c^j, y_c^j, \tfrac{H}{2}]^T - \mathbf{\Delta}_{1,t} - [\delta, \delta, \delta]^T, \\ \boldsymbol{b}_u^j &= [x_c^j, y_c^j, \tfrac{H}{2}]^T + \mathbf{\Delta}_{1,t} + [\delta, \delta, \delta]^T \end{aligned} \tag{16}$$

thus reducing the number of SD computations required by the optimization process. It is worth noticing that minimizing δ reduces the number of points requiring a SD computation, thus motivating the definition of a reference system minimizing ρ_{max}.

As highlighted in Section 3.3, the loss function measures placement quality, essentially imposing soft constraints on infeasible poses. As such, the optimizer described in Section 3.4 might propose configurations sticking out of the bin or colliding with already-placed items. Computing solutions for 18 different regions increases the likelihood that at least one candidate will be feasible. Once all 18 solution candidates $\boldsymbol{t}_h^{*,c}$, $h = 1, \ldots, 18$, are evaluated, a ranking strategy must be adopted to select the best candidate. A straightforward ranking metric could be the loss value in Equation (10). However, since $L(\mathbf{B}^i, \mathbf{T})$ is averaged over the number of points in the point cloud, ranking by its value favors sectors with fewer points, biasing the results toward non-tight configurations. Accordingly, if n_j is the number of points included in $\mathbf{B}^i_d \cap [\boldsymbol{b}_l^j, \boldsymbol{b}_u^j]$, $n_{\max}$ is the maximum value of n_j over $j = 1, \ldots, 9$, and $L^*_{j,k}$ is the optimal loss value registered for the pair (j,k), then the surrogate loss used to rank different solutions is:

$$\phi_{j,k} = \frac{n_j L^*_{j,k} + 2(n_{\max} - n_j)}{n_{\max}} \tag{17}$$

where $n_{\max} - n_j$ additional points are assigned a per-point loss of 2, as with points located farther than q from the object surface. The surrogate loss aims at making loss values of different sectors comparable with each other, by artificially reproducing the addition of far-away points without needing further SD computations.

Solutions are ranked according to $\phi_{j,k}$ and, following this order, are tested for feasibility in two stages:

1. **Containment test**: the AABB of the transformed item is compared with the AABB of the bin. If the item's AABB is not fully contained inside the bin's AABB, the pose is rejected as it violates Equation (1).
2. **Overlap test**: a volume intersection query is performed between the candidate item and all previously placed items. Any intersection indicates an overlap and thus a violation of Equation (2). Collision check is performed through the FCL library [48].

The first candidate that passes both tests becomes the target pose for the current item, and the pipeline moves on to the *physics-based placement*. If no feasible solution is found, the algorithm proceeds to the *refined search* step.

4.3 | Refined Search

The refined search aims at finding feasible solutions near a solution $\boldsymbol{t}_j^{*,c}$ of the coarse search. This is done iteratively over the solutions of the coarse search, by ranking them according to feasibility checks and loss values. Specifically, a reduced feasibility test is conducted by scaling the objects to 85% of their actual volume and checking for *containment* and *overlap* (see Section 4.2). The solutions are then grouped based on test results and ranked according to the severity of constraint violation. The first group includes solutions where both the *containment* and *overlap* tests pass. The second group includes those where the *containment* test passes, but the *overlap* test fails. The third group consists of solutions where the *overlap* test passes, but the *containment* test fails. Finally, the fourth group includes solutions where neither test passes. Violating the *overlap* constraint is considered less severe than violating the *containment* constraint. This distinction is motivated by the heuristic outlined in Section 3.3, where optimal configurations tend to be in a small neighborhood of configurations violating the *overlap* constraint, but not the *containment* constraint. In each group, items are then ranked according to loss values before the refined search is conducted, starting from the first item of the first group to the last item of the fourth group, unless a feasible solution is found first.

The optimization problem (Section 3.4) for the refined search limits its search space to a neighborhood of the solution $\boldsymbol{t}_h^{*,c}$ and leverages a subset $\mathbf{B}_{d,r}^i$ of the bin $\mathbf{B}^i$ with an increased resolution. The latter is obtained by downsampling the point cloud through the same voxel-grid filter used in the *coarse search*, using a reduced voxel size equal to $\gamma\rho_{max}, \gamma < \Gamma$ (the influence of parameter γ on the algorithm performance is discussed in Appendix A.2). In particular, if $\boldsymbol{\Delta}_2^T = [\boldsymbol{\Delta}_{2,t}^T, \boldsymbol{\Delta}_{2,r}^T]$ is a 6-element range-limiting vector with $\boldsymbol{\Delta}_{2,t} \in \mathbb{R}^3$ limiting translations and $\boldsymbol{\Delta}_{2,r} \in \mathbb{R}^3$ limiting rotations, the considered search space spans the interval $[\boldsymbol{t}_l^r, \boldsymbol{t}_u^r]$ in which:

$$\boldsymbol{t}_l^r = \boldsymbol{t}_h^{*,c} - \boldsymbol{\Delta}_2, \quad \boldsymbol{t}_u^r = \boldsymbol{t}_h^{*,c} + \boldsymbol{\Delta}_2 \tag{18}$$

and, like in the *coarse search* described in Section 4.2, the subset of $\mathbf{B}^i$ is limited to points that potentially contribute to shaping the loss function. Specifically, if $\boldsymbol{t}_{h,t}^{*,c} \in \mathbb{R}^3$ is the translational part of $\boldsymbol{t}_h^{*,c}$, the computation of the per-point loss value is limited to points included in the set $\mathbf{B}_{d,r}^i \cap [\boldsymbol{b}_{l,r}, \boldsymbol{b}_{u,r}]$, in which:

$$\boldsymbol{b}_{l,r} = \boldsymbol{t}_{h,t}^{*,c} - \boldsymbol{\Delta}_{2,t} - [\delta, \delta, \delta]^T, \quad \boldsymbol{b}_{u,r} = \boldsymbol{t}_{h,t}^{*,c} + \boldsymbol{\Delta}_{2,t} + [\delta, \delta, \delta]^T \tag{19}$$

After each solution refinement, feasibility is tested again without scaling, and the algorithm proceeds to the *physics-based placement* if the solution is feasible; otherwise, packing is interrupted.

4.4 | Physics-based Placement and Bin Update

Once a target configuration is found, the physics simulation environment is leveraged to simulate a top-down placement. This is done by rotating the mesh object according to the final configuration and initializing it in the scene with an offset $h = 0.05$ m on the z coordinate. A fixed vertical velocity $v = 0.01$ m s^{-1} is then applied to the object until the target value z_{target} is reached, or the time elapsed is greater than $1.2\frac{h}{v}$, or contact is detected. After that, an adjustment force, aimed at minimizing gaps between adjacent items, is applied to the center of mass of the object and directed towards the bin bottom-left corner until object velocities fall under a threshold of 0.001 m s^{-1} or 10 seconds of simulation time elapse. Lastly, the physics simulation is left running with only gravity and contact forces acting on items until the latter velocities fall under a threshold, implying that items are no longer moving. Item configuration is finally checked for *containment*, and if the check passes, the new item configuration is recorded. Otherwise, the item configuration is reinitialized as it was after the optimization step, and the process is repeated without the application of the adjustment force.

Once the new configuration is recorded, the bin point cloud $\mathbf{B}^{i+1}$ is generated by merging $\mathbf{B}^i$ with the point cloud $\mathcal{O}_i^P$ and applying the filling policy (Section 3.2).

4.5 | Online Adaptation

In the *online* variant, three considerations influence the pipeline. First, the algorithm does not know in advance the items that will arrive later, so it cannot presort them: each object is handled as it appears. Second, the bin size is fixed, and thus there can be no iterations on it. Third, the pose chosen for an item is executed immediately, since simulation data cannot be used to represent the actual item configuration in the bin. Indeed, the physics simulator cannot perfectly mimic real dynamic interactions, and even small errors in contact predictions might lead to very different final configurations, making the simulated configurations diverge from reality.

A volume-based collision test relies on the knowledge of item poses, which is extracted from the simulation in *offline* packing. In an *online* scenario, this information is not available, and the *overlap* test is replaced with a rough SD check: if any sample point of the transformed bin representation $\mathbf{B}^i$ has a negative SD with respect to the considered item, the pose is rejected. Likewise, instead of updating the bin model $\mathbf{B}^{i+1}$ by adding the CAD meshes of the objects, a new depth image is acquired after each placement, and the bin point cloud $\mathbf{B}^{i+1}$ is reinitialized from it. Although less accurate, this model preserves a usable geometric representation without requiring precise monitoring of item configuration. Lastly, the adjustment force is removed,

since any force applied by the real manipulator would likely be offset from the object's center of mass, generating overturning moments that could compromise the optimized pose.

In practical terms, the pipeline changes as follows:

- item sorting during initialization (Section 4.1) and iterations on bin sizes are removed;
- the *overlap* test is replaced by checking the presence of points leading to negative SDs, and thus intersecting the object (Section 4.2);
- the *reduced overlap* test (before *refined search*) is replaced by checking the presence of points leading to SDs below $0.15\rho_{max}$ (Section 4.3);
- top-down insertion is achieved in reality instead of simulation, and the adjusting force is removed (Section 4.4);
- bin representation is extracted by acquiring a depthmap after object placement is completed and converting it to a point cloud before applying the filling policy (Section 4.4).

TABLE 1 | Parameters employed during simulations.

Parameter	Value	Measurement unit	Reference section
b^{max}	0.4	m	3.1
n_{train}	100,000	/	3.1
w_{oct}	9	/	3.1
d_{thresh}	0.05	m	3.1
a	200,000	/	3.3
q	0.03	/	3.3
c	−0.006	m	3.3
Q	50	/	3.4
M	200	/	3.4
Γ	0.25	/	4.2
γ	0.125	/	4.3
$\mathbf{\Delta}_{2,t}$	0.03, 0.03, 0.03	m	4.3
$\mathbf{\Delta}_{2,r}$	0.1, 0.1, 0.1	rad	4.3
v	0.05	m s^{-1}	4.4

5 | Simulations

In order to verify the effectiveness of the proposed approach, a simulation campaign is carried out, leveraging the PyBullet [49] environment to reproduce physical interactions. Simulations are carried out on a desktop computer equipped with 32 GB of RAM, a NVIDIA GTX 3070 graphic card with 6 GB of VRAM, and an Intel i7-10700KF processor. The algorithm is implemented in Python, prioritizing development flexibility over low-level performance optimization. The simulation parameters, chosen to balance accuracy and computational speed based on the authors' experience, are summarized in Table 1. A sensitivity analysis on parameter variations is discussed in Appendices A.1 and A.2.

Three sets of simulations are generated, assessing the performance under different scenarios. The first, discussed in Section 5.1 and referred to as the *space-utilization set*, evaluates the overall packing capabilities of the algorithm, specifically its ability to place the largest possible number of items inside a fixed-size bin, and does not include the bin-size iteration process.

The second set of simulations, described in Section 5.2 and referred to as the *box-minimization set*, evaluates the algorithm's ability to determine the smallest possible box capable of containing an assigned set of items.

The third set of simulations, described in Section 5.3 and referred to as the *online packing set*, provides a preliminary assessment of the algorithm effectiveness in a *simulated online* scenario.

Each set of simulations is repeated using items sampled from 5 different groups, classified as follows:

- group A: cuboids;
- group B: cylinders;
- group C: equilateral prisms;
- group D: scalene triangular prisms and right pyramidal frusta.

Appendix A.3 contains the complete list of randomly generated items employed in the simulations. They define 3 item families:

- the *cuboid* family contains items sampled from group A;
- the *regular* family contains items sampled from groups A, B, and C;
- the *irregular* family contains items sampled from all groups.

Two different metrics are used to assess packing performance, namely *Packing Utility* (PU, U_b) and the *Maximum-Height Utility* (MHU, U_{mh}):

$$U_b = \frac{\sum_i |\mathcal{O}_i|}{|\mathcal{B}|}, \quad U_{mh} = \frac{\sum_i |\mathcal{O}_i|}{A_b\, h_{\max}} \tag{20}$$

where $|\mathcal{O}_i|$ is the volume of item i extracted from its mesh $\mathcal{O}_i^M$, $|\mathcal{B}| = LWH$ is the bin volume, $A_b = LW$ is the bin base area, $h_{\max}$ is the maximum height reached by the items packed in the box. The two volume–utility metrics are selected to mirror the main operational regimes found in modern logistics centers. U_b assesses pure space-filling efficiency when bin dimensions are assigned, a common constraint in warehouses where box resizing is not possible. U_{mh} relates to scenarios in which only the bin footprint is fixed, but the height can be trimmed, rewarding algorithms that minimize the bin height space while respecting a prescribed base area.

The complete set of object models and simulation results, including item configurations, execution times and evaluation metrics, is publicly accessible in [34].

5.1 | Space-Utilization Simulation Set

The aim of this simulation set is to pack the largest number of items in a fixed-sized bin, in order to assess the space-utilization capabilities of the packing algorithm.

In particular, 20 packing simulations are evaluated for each item family, with each simulation trying to pack the largest number of objects out of a set of 100 objects inside a bin whose dimensions are randomly sampled between 0.15 m and 0.3 m, without

iterating on bin sizes. Item sets are resampled for each simulation, and the total item volume is checked to ensure that the entire set cannot fit within the bin. Average results are summarized in Table 2, for each item family. In addition to average PU $\overline{U}_b$ and average MHU $\overline{U}_{mh}$, the average time $\bar{t}$ required for a simulation, as well as the average time required to evaluate each item $\bar{t}_{item}$, is reported. The distributions of U_b and U_{mh} across the tests is reported via boxplots in Figure 4 for each item family. Examples of packed configurations are shown in Figure 5. As one can intuitively expect, both $\overline{U}_b$ and $\overline{U}_{mh}$ decrease with an increase in item geometric complexity, showing that more complex shapes are packed with higher difficulty. Moreover, $\overline{U}_{mh}$ is always higher than $\overline{U}_b$, due to a residual vertical space in the box that can be trimmed off. The average simulation time is around 1 hour, with an average of roughly 36*s* per packed item.

5.2 | Box-Minimization Simulation Set

The box-minimization simulation set aims at reproducing the fulfillment of an average e-commerce order. The number of items to pack is based on an average e-commerce order length: Jin et al. [50] analyzed a set of more than 3 million users' purchasing sessions pertaining to a popular e-commerce platform, which resulted in an average item-per-order number of 4.2. Accordingly, item-set lengths are randomly sampled between 3 and 7. For each item family, 30 packing simulations are carried out and evaluated. In this scenario, U_b is computed as the volume of the maximum AABB capable of containing the packed items, and thus U_b and U_{mh} coincide. The average results are summarized in Table 3. The average initial PU $\overline{U}_b^0$, representing the average PU at the end of the first packing iteration, is reported to show the contribution of the reiterating process on bin sizes. $\bar{t}$ is the average time required to complete a simulation, whereas $t_{\text{item}}^{\text{all}}$ denotes the average simulation time per item, accounting for the time required by all box-size iterations. The distribution of U_b across the tests is reported via boxplots in Figure 6. Examples of packed configurations are shown in Figure 7. Compared to the *space-utilization* simulation set, performance degrades slightly with item-shape complexity, with $\overline{U}_b$ ranging from a maximum of 0.601 for the cuboid family to a minimum of 0.460 for the irregular family. The average PU increases by 37.5% from the first iteration when considering the cuboid family, and by 24.7% and 31.1% when considering, respectively, the regular and irregular families, showing that iterating over bin sizes is useful, increasing space utilization of roughly 30%.

5.3 | Online Simulation Set

To evaluate the algorithm's performance in online scenarios, we use the same setup as in the *space-utilization* simulations, with appropriate modifications for online packing, in particular removing item pre-sorting and the adjustment force (Section 4.5). Twenty packing simulations, each trying to pack the largest

TABLE 2 | Average results for the *space-utilization* set of simulations.

Item family	$\overline{U}_b$	$\overline{U}_{mh}$	$\bar{t}$, min	$\bar{t}_{item}$, sec
cuboid	0.677	0.732	56.97	34.18
regular	0.621	0.653	57.65	34.59
irregular	0.609	0.619	56.40	33.84

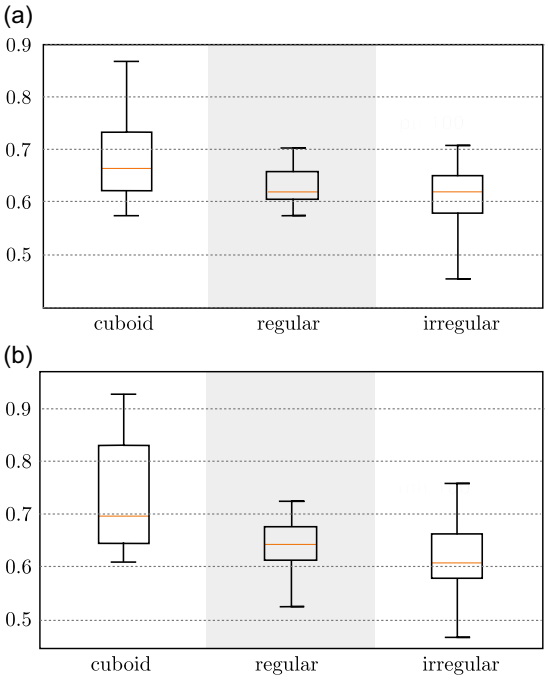


FIGURE 4 | Boxplots showing the distribution of U_b (a) and U_{mh} (b) across the *space-utilization* simulation set, for each item family. The orange line defines the median value, the box shows the interquartile range, and the whiskers extend to the minimum and maximum values (within a span of 1.5 times the interquartile range after the upper and lower bounds of the box, to exclude outliers).

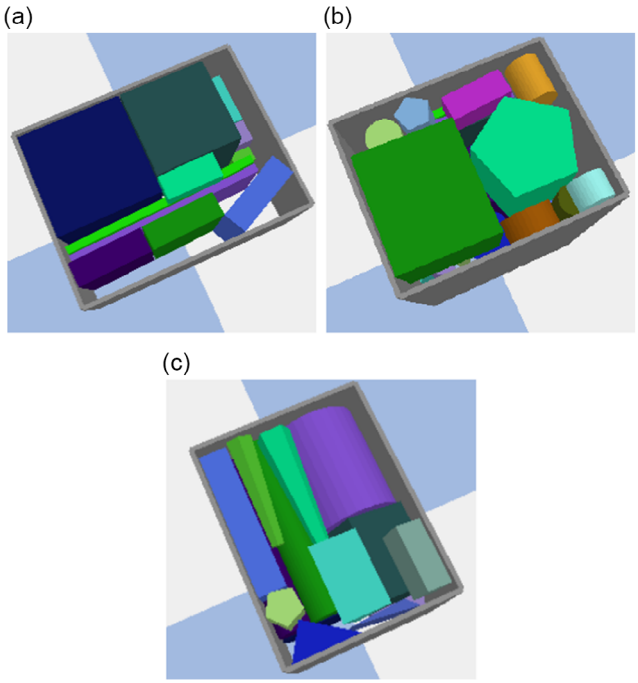


FIGURE 5 | Examples of packed items for different item families in the space-utilization simulation set: (a) cuboid family; (b) regular family; (c) irregular family.

TABLE 3 | Average results for the *box-minimization* set of simulations.

Item set	$\overline{U}_b$	$\overline{U}_b^0$	$\bar{t}$, sec	$\bar{t}_{item}^{all}$, sec
Cuboid	0.601	0.437	258.56	48.56
Regular	0.475	0.381	249.64	45.39
Irregular	0.460	0.351	227.61	42.38

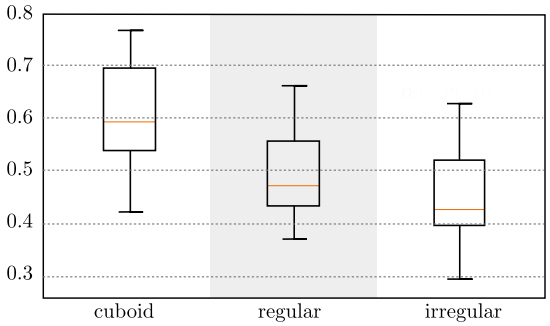


FIGURE 6 | Boxplots showing the distribution of U_b across the *box-minimization* simulation set, divided by item family. The orange lines define the median values, the boxes show the interquartile range and the whiskers ends at the last observed value that falls inside a range of 1.5 times the interquartile range after the bound (upper or lower) of the box.

number of objects out of a set of 100 objects inside a bin whose dimensions are randomly sampled between 0.15 m and 0.3 m are evaluated for each item set, without iterating on bin sizes. Item sets are resampled for each simulation, and the total item volume is checked to ensure that the entire set cannot fit within the bin. Average results are summarized in Table 4, for each item family. In addition to $\overline{U}_b$ and $\overline{U}_{mh}$, the average time $\bar{t}$ required for a simulation to complete is reported. The distributions of U_b and U_{mh} across the tests is reported via boxplots in Figure 8 for each item family. Examples of packed configurations are shown in Figure 9.

TABLE 4 | Results for the *online* set of simulations. U_{mh} is the *maximum-height utility*, U_b is the *packing utility*, and t is the simulation time. All values are averaged over the simulation set results.

Item set	$\overline{U}_b$	$\overline{U}_{mh}$	$\bar{t}$, min	$\bar{t}_{item}$, sec
Cuboid	0.463	0.495	41.00	24.60
Regular	0.439	0.453	39.54	23.72
Irregular	0.357	0.365	39.20	23.52

The same trend observed for U_b and U_{mh} in the *space-utilization* set is repeated here, with lower performance scores for higher geometric complexity and slightly lower values for U_b compared to U_{mh}. The overall poorer results compared to the *space-utilization* set are primarily due to the lack of item-sequence optimization and the adjustment force. The impact of removing the latter is clearly illustrated by comparing the results of the cuboid family in the *space-utilization* set (Figure 5a) and the *online* set (Figure 9a): while Figure 5a shows tightly packed items due to the adjustment force, Figure 9a displays a more loosely arranged configuration. The average simulation time is slightly smaller than in the *space-utilization* counterpart, since it is no longer necessary to wait for items to stabilize

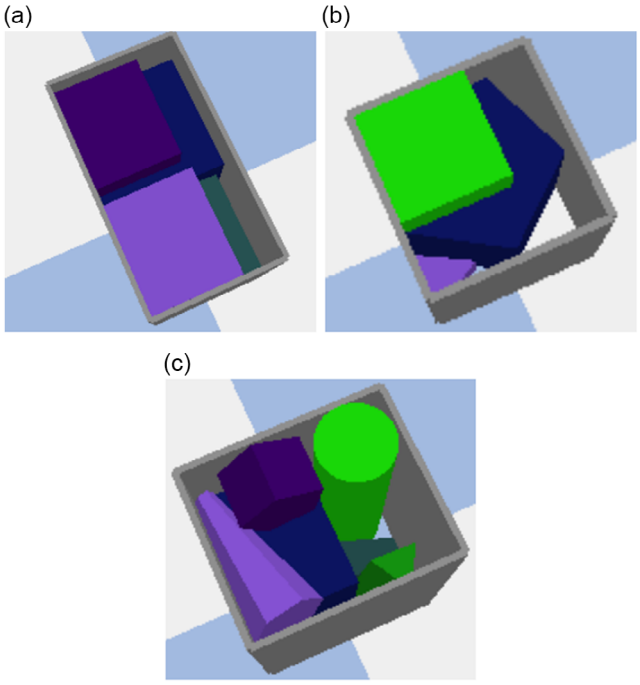


FIGURE 7 | Examples of packed items for different item families in the box-minimization simulation set: (a) cuboid family; (b) regular family; (c) irregular family.

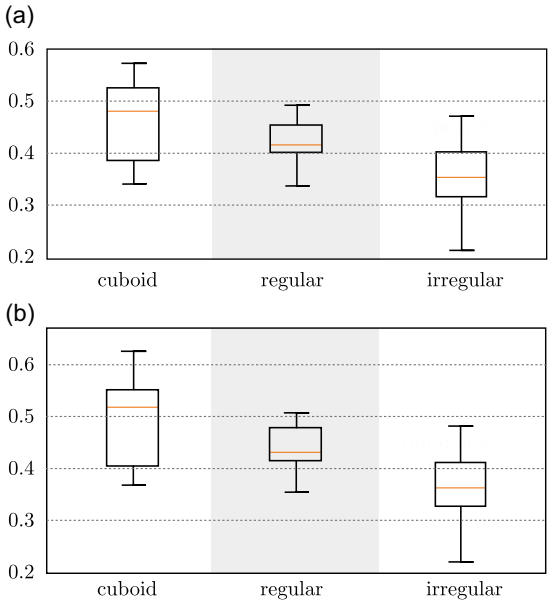


FIGURE 8 | Boxplots showing the distribution of U_b (a) and U_{mh} (b) across the *online* simulation set, for each item family. The orange line defines the median value, the box shows the interquartile range, and the whiskers extend to the minimum and maximum values (within a span of 1.5 times the interquartile range after the upper and lower bounds of the box, to exclude outliers).

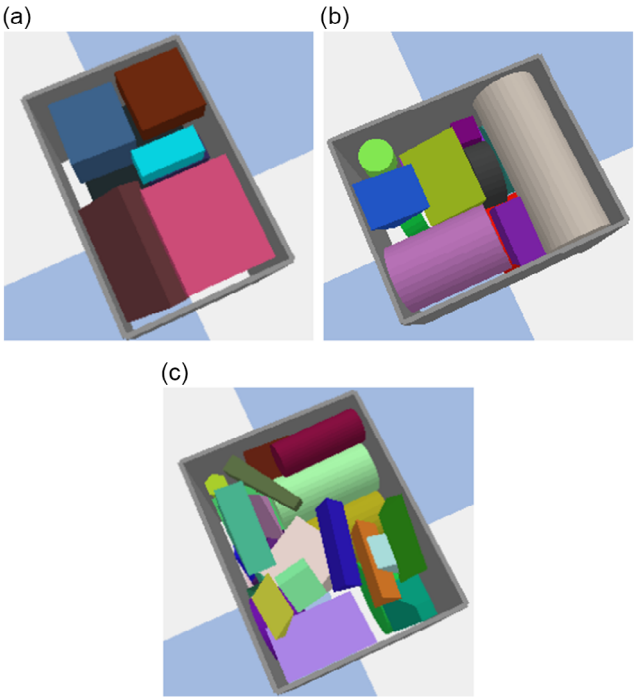


FIGURE 9 | Examples of packed items for different item families in the *online* simulation set: (a) cuboid family; (b) regular family; (c) irregular family.

during the adjustment-force application, resulting in an average of approximately 25 s per packed item.

6 | Conclusions

This paper introduced a novel method aimed at optimizing box-packing tasks, in both offline and online packing scenarios. The algorithm is capable of handling objects with several primitive convex shapes (cuboids, cylinders, prisms, pyramidal frusta) and finding optimal item poses in an arbitrary continuous subset of $SE(3)$. The use of neural-network-approximated SDFs, coupled with a carefully designed optimization search, enables the determination of optimized item configurations within limited computation time. In *offline* packing, prior knowledge of the items to pack allows pre-sorting strategies to improve efficiency, while a physics simulator can assess item stability after each placement. The algorithm identifies both the best item configuration and the optimal bin dimensions. In the *online* variant, pre-sorting and real-time physics simulations are not possible, yet the algorithm still determines an optimal item configuration. To the best of our knowledge, this combination of capabilities is absent from the current state of the art.

The proposed algorithm has been tested in simulations with 3 different families of objects (cuboids; regular objects such as cylinders and equilateral prisms; irregular objects such as scalene triangular prisms and pyramidal frusta) on 3 different scenarios, assessing its performance under various working conditions. When dealing with irregular objects, our solution demonstrates competitive performance, achieving an average packing efficiency of 51.5% in a set of tests aimed at assessing space utilization performance and 46.0% when trying to fulfill a simulated e-commerce order. These values increase to 62.1% and 47.5%, respectively, with regular objects and to 67.7% and 60.1% when the objects are cuboids. The *online* variant achieved a 35.7% packing efficiency in its most challenging configuration, that is, with irregular objects, and a 46.3% packing efficiency when dealing with cuboid shapes. In all cases, average computational time to find a suitable configuration did not exceed 50 s per item on a consumer-grade computer.

Future work will focus on addressing the main limitation of the proposed approach, namely the class of object geometries currently considered. In particular, extensions to concave shapes will be investigated through convex decomposition techniques or alternative geometric representations that enable the handling of more complex geometries. Moreover, efforts will be directed towards reducing computational time by leveraging improved hardware capabilities and by reimplementing computationally intensive components in compiled languages such as C++. Further improvements could be achieved through GPU acceleration of geometric operations such as point-cloud manipulation and heuristic pruning strategies to reduce the explored orientation search space. Additionally, the current network used for SD approximation does not allow for fine geometric details to be accurately captured, and alternative techniques, such as those proposed in [51], may prove more suitable for items requiring higher accuracy. Another promising direction is the incorporation of manipulation constraints into the definition of the considered subsets of $SE(3)$, enabling the exploration of solutions that are not only geometrically valid and statically stable, but also feasible from a manipulation perspective. Finally, integrating the algorithm into a real-world packing platform will allow evaluation of its performance under actual conditions.

Author Contributions

Michele Angelini: conceptualization (lead), data curation (lead), methodology (lead), writing – original draft (lead), writing – review & editing (lead). **Marco Carricato**: conceptualization (supporting), funding acquisition (lead), project administration (lead), supervision (lead), writing – review & editing (supporting).

Acknowledgments

The authors gratefully acknowledge the R&I Department of I.M.A. S.p.A. for providing the facilities and technical support essential to the development of the presented algorithm.

Open access publishing facilitated by Universita degli Studi di Bologna, as part of the Wiley - CRUI-CARE agreement.

Funding

This study was supported by Italian Ministry of University and Research (MUR) through the FISA 2023 grant 'Advanced manipulation, grasping and control for enhanced machinery autonomy and reconfigurability in pharmaceutical processes and e-commerce packaging (APACHE)' (Grant CUP J53C25000520001).

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

The data that support the findings of this study are openly available in Zenodo at https://doi.org/10.5281/zenodo.17226420, reference number 17226421.

Endnotes

[1] In an octree, a *node* represents a cuboid region of space, a *branch* is a node that is subdivided into smaller regions, a *leaf* is a node that is not further subdivided, and the *tree depth* is the number of levels from the root node to the deepest leaf. Each *branch* is subdivided exactly into 8 parts, called *octants*.

[2] Horizontal bands of lower values are less evident but, nonetheless, present. This is due to the approximated (learned) representation of the SDF that leads to asymmetries in the object representation.

[3] The AABB of a mesh is the solid obtained by taking the Cartesian product of the intervals given by the minimum and maximum coordinates of the mesh vertices on each axis.

[4] The Chebyshev center of a set of points $\mathcal{H}$ is the solution to the minimization problem $\arg\min_{\boldsymbol{P}^*}(\rho_{max})$ where $\rho_{max} = \max_{\boldsymbol{P}\in\mathcal{H}}(|\boldsymbol{P}^* - \boldsymbol{P}|)$. Since the latter is a convex problem, any optimization algorithm quickly converges to the solution.

[5] Each search space spans a cuboid translational space whose height is the height of the bin, and its width and length are a third of the respective dimensions of the bin, leading to sectors never being fully filled.

References

1. L. V. Kantorovich, "Mathematical Methods of Organizing and Planning Production," *Management Science* 6, no. 4 (1960): 366–422, https://doi.org/10.1287/mnsc.6.4.366.

2. M. Delorme, M. Iori, and S. Martello, "Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms," *European Journal of Operational Research* 255, no. 1 (2016): 1–20, https://doi.org/10.1016/j.ejor.2016.04.030.

3. I. A. Bonev, J. Ryu, "A New Approach to Orientation Workspace Analysis of 6-Dof Parallel Manipulators," *Mechanism and Machine Theory* 36, no. 1 (2001): 15–28, https://doi.org/10.1016/S0094-114X(00)00032-X.

4. H. Xiong, K. Ding, W. Ding, et al., "Enhancing Robotics Online 3D Bin Packing: A Comparative Study of Conventional Heuristic and Deep Reinforcement Learning Approaches," in *IEEE 20th International Conference on Automation Science and Engineering (CASE)* (IEEE, 2024), 4083–4089, https://doi.org/10.1109/case59546.2024.10711723.

5. S. Martello, D. Pisinger, and D. Vigo, "The Three-Dimensional Bin Packing Problem," *Operations Research* 48, no. 2 (2000): 256–267, https://doi.org/10.1287/opre.48.2.256.12386.

6. K. Fleszar, "A MILP Model and Two Heuristics for the Bin Packing Problem with Conflicts and Item Fragmentation," *European Journal of Operational Research* 303, no. 1 (2022): 37–53, https://doi.org/10.1016/j.ejor.2022.02.014.

7. J. M. H. Arango, G. Pantoja-Benavides, S. Valero, et al., "Approaches for the On-Line Three-Dimensional Knapsack Problem with Buffering and Repacking," *Mathematics* 12, no. 20 (2024): 3223, https://doi.org/10.3390/math12203223.

8. S. Krisadee and W. Jindaluang, "An Improvement of a Heuristic Algorithm for 3D Bin-Packing Problem," in *28th International Computer Science and Engineering Conference (ICSEC)* (IEEE, 2024), 1–6, https://doi.org/10.1109/icsec62781.2024.10770638.

9. A. L. Jean, N. Brauner, O. Briant, et al., "Stability Constraints in a 3D Knapsack Problem with Non Parallelepipedic Items," *Computers & Industrial Engineering* 189 (2024): 109943, https://doi.org/10.1016/j.cie.2024.109943.

10. M. Mhiri, "A Mixed-integer Programming Model for the Bin Packing Problem with Piecewise Linear Loading Cost and Time Windows," in *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)* (IEEE, 2024), 395–399, https://doi.org/10.1109/ieem62345.2024.10857005.

11. H. Salamati-Hormozi, A. H. Kashan, and B. Ostadi, "A Three-Dimensional Bin Packing Problem with Item Fragmentation and Its Application in the Storage Location Assignment Problem," *4OR* 22, no. 4 (2024): 483–536, https://doi.org/10.1007/s10288-024-00576-6.

12. Y. P. Tsang, D. Y. Mo, K. T. Chung, et al., "DeepPack3D: A Python Package for Online 3D Bin Packing Optimization by Deep Reinforcement Learning and Constructive Heuristics," *Software Impacts* 23 (2025): 1–4, https://doi.org/10.1016/j.simpa.2024.100732.

13. C.-C. Wong, T.-T. Tsai, and C.-K. Ou, "Integrating Heuristic Methods with Deep Reinforcement Learning for Online 3D Bin-Packing Optimization," *Sensors* 24, no. 16 (2024): 5370–5382, https://doi.org/10.3390/s24165370.

14. H. Zhao, Q. She, C. Zhu, et al., "Online 3D Bin Packing with Constrained Deep Reinforcement Learning," in *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, (Association for the Advancement of Artificial Intelligence (AAAI), 2021), 741–749, Held Virtually, https://doi.org/10.1609/aaai.v35i1.16155.

15. H. Zhao, C. Zhu, X. Xu, et al., "Learning Practically Feasible Policies for Online 3D Bin Packing," *Information Sciences* 65, no. 112105 (2021): 1–14, https://doi.org/10.1007/s11432-021-3348-6.

16. H. Zhao, J. Xu, K. Yu, et al., "Deliberate Planning of 3D Bin Packing on Packing Configuration Trees," *The International Journal of Robotics Research* 0 (2025): 1–22, https://doi.org/10.1177/02783649251380619.

17. K. Muller, P. Merkle, and T. Wiegand, "3-D Video Representation Using Depth Maps," *Proceedings of the IEEE* 99, no. 4 (2011): 643–656, https://doi.org/10.1109/jproc.2010.2091090.

18. M. Aleksandrov, S. Zlatanova, and D. J. Heslop, "Voxelisation Algorithms and Data Structures: A Review," *Sensors* 21, no. 24 (2021): 8241, https://doi.org/10.3390/s21248241.

19. D. Meagher, "Geometric Modeling Using Octree Encoding," *Computer Graphics and Image Processing* 19, no. 2 (1982): 129–147, https://doi.org/10.1016/0146-664x(82)90104-6.

20. A. Lamba and R. Bhalla, "A Review of various 3-D Modelling Techniques and an Introduction to Point Clouds," in *International Conference on Emerging Trends in Artificial Intelligence and Smart Systems* (THEETAS, 2022), https://doi.org/10.4108/eai.16-4-2022.2318159.

21. A. Shamir, "A Survey on Mesh Segmentation Techniques," *Computer Graphics Forum* 27, no. 6 (2008): 1539–1556, https://doi.org/10.1111/j.1467-8659.2007.01103.x.

22. W. Zhang and Y. Zhou, "Level-Set Functions and Parametric Functions," *The Feature-Driven Method for Structural Optimization* (Elsevier, 2021), 9–46, https://doi.org/10.1016/b978-0-12-821330-8.00002-x.

23. Z. Wang, "3D Representation Methods: A Survey," arXiv preprint arXiv:2410.06475 (2024), https://doi.org/10.48550/ARXIV.2410.06475.

24. F. Kagerer, M. Beinhofer, S. Stricker, et al., "BED-BPP: Benchmarking Dataset for Robotic Bin Packing Problems," *The International Journal of Robotics Research* 42, no. 11 (2023): 1007–1014, https://doi.org/10.1177/02783649231193048.

25. T. Romanova, J. Bennell, Y. Stoyan, et al., "Packing of Concave Polyhedra with Continuous Rotations Using Nonlinear Optimisation," *European Journal of Operational Research* 268, no. 1 (2018): 37–53, https://doi.org/10.1016/j.ejor.2018.01.025.

26. Y. Stoyan, A. Pankratov, and T. Romanova, "Quasi-Phi-Functions and Optimal Packing of Ellipses," *Journal of Global Optimization* 65, no. 2 (2016): 283–307, https://doi.org/10.1007/s10898-015-0331-2.

27. Q. Cui, V. Rong, D. Chen, et al., "Dense, Interlocking-Free and Scalable Spectral Packing of Generic 3D Objects," *ACM Transactions on Graphics* 42, no. 4 (2023): 1–14, https://doi.org/10.1145/3592126.

28. Q. Zhuang, Z. Chen, K. He, et al., "Dynamics Simulation-Based Packing of Irregular 3D Objects," *Computers & Graphics* 123 (2024): 103996, https://doi.org/10.1016/j.cag.2024.103996.

29. L. Wang, S. Guo, S. Chen, et al., "Two Natural Heuristics for 3D Packing with Practical Loading Constraints," *PRICAI 2010: Trends in Artificial Intelligence*, (Springer, Berlin, Heidelberg, 2010), 256–267, https://doi.org/10.1007/978-3-642-15246-7_25.

30. F. Wang and K. Hauser, "Dense Robotic Packing of Irregular and Novel 3D Objects," *IEEE Transactions on Robotics* 38, no. 2 (2022): 1160–1173, https://doi.org/10.1109/tro.2021.3097261.

31. J.-H. Pan, K.-H. Hui, X. Gao, et al., "SDF-Pack: Towards Compact Bin Packing with Signed-Distance-Field Minimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (2023), 10612–10619, https://doi.org/10.1109/IROS55552.2023.10341940.

32. H. Liu, L. Zhou, J. Yang, et al., "The 3D Bin Packing Problem for Multiple Boxes and Irregular Items Based on Deep Q-Network," *Applied Intelligence* 53, no. 20 (2023): 23398–23425, https://doi.org/10.1007/s10489-023-04604-6.

33. H. Zhao, Z. Pan, Y. Yu, et al., "Learning Physically Realizable Skills for Online Packing of General 3D Shapes," *ACM Transactions on Graphics* 42, no. 5 (2023): 1–21, https://doi.org/10.1145/3603544.

34. M. Angelini and M. Carricato, "*Supplementary Material to Optimizing 3D Bin Packing of Heterogeneous Objects Using Continuous Transformations in SE(3)*," (2025), https://doi.org/10.5281/zenodo.17226421.

35. J. J. Park, P. Florence, J. Straub, et al., "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (IEEE, 2019), https://doi.org/10.1109/CVPR.2019.00025.

36. T. Menzies, E. Kocagüneli, L. Minku, et al.," Using Goals in Model-Based Reasoning," *Sharing Data and Models in Software Engineering* (Elsevier, 2015), 321–353, https://doi.org/10.1016/b978-0-12-417295-1.00024-2.

37. Y. A. LeCun, L. Bottou, G. B. Orr, et al., "Efficient BackProp," *Neural Networks: Tricks of the Trade*, 2nd ed. (Springer Berlin Heidelberg, 2012), 9–48, https://doi.org/10.1007/978-3-642-35289-8.

38. T. Salimans and D. P. Kingma, "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks," *Advances in Neural Information Processing Systems* (Neural information processing systems foundation, 2016), 29, https://doi.org/10.48550/arXiv.1602.07868.

39. Y. Bai, "RELU-Function and Derived Function Review," *SHS Web of Conferences*, ed. A. Luqman, Q. Zhang and W. Liu, (EDP Sciences, 2022), 144, 8026–8037, https://doi.org/10.1051/shsconf/202214402006.

40. A. Paszke, S. Gross, F. Massa, et al., "PyTorch: an imperative style, high-performance deep learning library," in *33rd International Conference on Neural Information Processing Systems* (Neural information processing systems foundation, 2019), https://doi.org/10.48550/arXiv.1912.01703.

41. D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations* (Ithaca, NY: ArXiv, 2015), https://doi.org/10.48550/arXiv.1412.6980.

42. N. Hansen and A. Ostermeier, "Completely Derandomized Self-Adaptation in Evolution Strategies," *Evolutionary Computation* 9, no. 2 (2001): 159–195, https://doi.org/10.1162/106365601750190398.

43. N. E. Toklu, T. Atkinson, V. Micka, et al., "Evotorch: Scalable Evolutionary Computation in Python," (arXiv preprint arXiv:2302.12600, 2023), https://doi.org/10.48550/arXiv.2302.12600.

44. S. Ali, A. G. Ramos, M. A. Carravilla, et al., "On-Line Three-Dimensional Packing Problems: A Review of Off-Line and on-Line Solution Approaches," *Computers & Industrial Engineering* 168 (2022): 108122, https://doi.org/10.1016/j.cie.2022.108122.

45. M. V. Balashov, "Chebyshev Center and Inscribed Balls: Properties and Calculations," *Optimization Letters* 16 (2022): 2299–2312, https://doi.org/10.1007/s11590-021-01823-z.

46. C.-F. Chien and J.-F. Deng, "A Container Packing Support System for Determining and Visualizing Container Packing Patterns," *Decision Support Systems* 37, no. 1 (2004): 23–34, https://doi.org/10.1016/s0167-9236(02)00192-6.

47. R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation* (IEEE, 2011), https://doi.org/10.1109/icra.2011.5980567.

48. J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *2012 IEEE International Conference on Robotics and Automation* (St Paul, 2012), https://doi.org/10.1109/icra.2012.6225337.

49. J. Baumgärtner, M. Hansjosten, D. Schönhofen, et al., "PyBullet Industrial: A Process-Aware Robot Simulation," *Journal of Open Source Software* 8, no. 85 (2023): 5174–5180, https://doi.org/10.21105/joss.05174.

50. W. Jin, H. Mao, Z. Li, et al., "Amazon-M2: a multilingual multi-locale shopping session dataset for recommendation and text generation," in *37th International Conference on Neural Information Processing Systems* (NIPS '23, 2023), 8006–8026, https://doi.org/10.48550/arXiv.2307.09688.

51. T. Müller, A. Evans, C. Schied, et al., "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding," *ACM Transactions on Graphics* 41, no. 4 (2022): 102, https://doi.org/10.1145/3528223.3530127.

Appendix A

Because the proposed approach depends on several parameters, it is important to analyze how variations in these key quantities affect its ability to identify suitable item configurations. In particular, the tuning of the loss-function parameters introduced in Section 3.3 is discussed in Appendix A.1. The influence of the number of candidates per generation M and the number of generations Q in the optimization process described in Section 3.4, as well as the voxel-size parameters Γ and γ used in the coarse and refined searches of Sections 4.2 and 4.3, is analyzed in Appendix A.2. The objects employed in the simulation campaign described in Section 5 are listed in Appendix A.3.

Appendix A.1 Tuning of Loss Function Parameters

Parameters a, q, and c of the loss function (which depend on the SD measurement unit) are tuned by evaluating 5 different values for each one of them for a total of 125 combinations. Evaluation is performed by evaluating the 5 candidates shown in Table A1 on a set of 10 *box-minimization* simulations (Section 5.2), each trying to pack an item-set whose length is randomly sampled between 3 and 7 in the smallest box possible. Item sets vary across different *box-minimization* simulations, but repeat when considering different parameter candidates. Objects in the sets are randomly sampled from each group listed in Appendix A.3 by first sampling a group and then an item within the group, with repetitions allowed. Any parameter combination is evaluated through the $\overline{U}_b$ score defined in Section 5. The optimal parameter set, highlighted in Table A1, yields the highest performance with a $\overline{U}_b$ value of 0.440, in contrast to the lowest-performing configuration, which achieves only $\overline{U}_b = 0.254$. Across the full set of tests, the average $\overline{U}_b$ is 0.378, with a median of 0.380 and a standard deviation of 0.035, suggesting that performance fluctuations remain within a relatively narrow and stable range.

TABLE A1 | Candidate parameters evaluated during tuning.

Param.	Cand. #1	Cand. #2	Cand. #3	Cand. #4	Cand. #5
a	50,000	75,000	100,000	**200,000**	500,000
q	0	0.01	**0.03**	0.05	0.1
c	0	−0.004	**−0.006**	−0.007	−0.009

To further investigate the influence of the loss-function parameters on packing performance and computational effort, Table A2 reports the average results obtained during the tuning phase for each explored parameter value, evaluated using three metrics: the average box utilization $\overline{U}_b$, the average computational time per item $\bar{t}^{all}_{item}$ considering all box-size iterations, and the average computational time per item $\bar{t}^{best}_{item}$ considering only the smallest box for the considered item set (that is, the configuration yielding the highest $\overline{U}_b$). Comparing $\bar{t}^{all}_{item}$ and $\bar{t}^{best}_{item}$ provides insight into the number of iterations required for convergence: similar values indicate few box-size iterations, whereas large differences suggest that many iterations were needed to reach the optimal configuration. For each parameter, the values in Table A2 represent the average performance across all simulations in which that parameter assumes the given value, regardless of the values of the other parameters.

Parameter a has a positive impact on the achievable box utilization $\overline{U}_b$, which increases steadily as a grows from 50,000 to 200,000. However, larger values of a also lead to a noticeable increase in computational time, as reflected by higher $\bar{t}^{all}_{item}$ values. While $\bar{t}^{best}_{item}$ remains relatively stable, the increase in $\bar{t}^{all}_{item}$ suggests that a larger a leads to a higher number of iterations before convergence. This behavior likely occurs because higher values of a, corresponding to a steeper loss function near boundaries, tend to push the optimization towards solutions farther from object surfaces, thereby reducing the box-utilization improvement achieved at each iteration.

Parameter q exhibits a similar behavior, with $\overline{U}_b$ improving as q increases from 0 to 0.03, after which the space utilization score stabilizes. Beyond this value, increasing q mainly affects the computational cost, leading to higher values of both $\bar{t}^{all}_{item}$ and $\bar{t}^{best}_{item}$ due to the larger portion of space

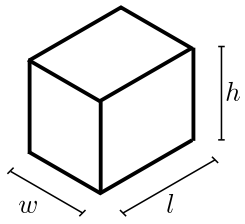
TABLE A2 | Average results obtained during the tuning campaign for different values of the loss-function parameters a, q, and c. Highlighted entries indicate the selected parameter set.

a	$\overline{U}_b$	$\bar{t}^{all}_{item}$, sec	$\bar{t}^{best}_{item}$, sec
50,000	0.347	29.0	17.1
75,000	0.360	27.0	15.9
100,000	0.365	26.6	15.7
200,000	**0.384**	**40.2**	**14.8**
500,000	0.377	48.2	16.4
q	$\overline{U}_b$	$\bar{t}^{all}_{item}$, sec	$\bar{t}^{best}_{item}$, sec
0	0.345	24.8	14.6
0.01	0.363	28.9	17.0
0.03	**0.377**	**33.3**	**19.6**
0.05	0.377	41.0	24.1
0.1	0.371	43.1	25.3
c	$\overline{U}_b$	$\bar{t}^{all}_{item}$, sec	$\bar{t}^{best}_{item}$, sec
0	0.330	32.0	18.8
−0.004	0.351	32.8	19.3
−0.006	**0.373**	**35.1**	**20.7**
−0.007	0.379	33.8	19.9
−0.009	0.375	37.3	21.9

TABLE A3 | Results of a space-utilization test when varying the main algorithm parameters. Each parameter is varied with respect to its reference value (100%) while the others are kept fixed. Reference values are highlighted.

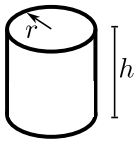
Γ, %	U_b	U_{mh}	t, min	t_{item}, sec
25	0.561	0.576	60.68	36.41
50	0.552	0.564	60.12	36.07
75	0.533	0.547	56.53	35.92
100	**0.559**	**0.574**	**57.12**	**34.27**
125	0.553	0.567	54.87	32.92
150	0.546	0.556	48.20	28.92
175	0.497	0.503	41.17	24.70
200	0.427	0.442	38.74	23.24
γ, %	U_b	U_{mh}	t, min	t_{item}, sec
25	0.593	0.601	54.55	32.73
50	0.563	0.568	60.12	36.07
75	0.566	0.577	56.38	33.83
100	**0.559**	**0.574**	**57.12**	**34.27**
125	0.558	0.565	56.37	33.82
150	0.535	0.542	59.92	35.95
175	0.544	0.557	56.64	33.98
200	0.536	0.544	55.4	33.24
M, %	U_b	U_{mh}	t, min	t_{item}, sec
25	0.517	0.522	23.14	13.88
50	0.534	0.543	32.59	19.55
75	0.541	0.553	47.49	28.49
100	**0.559**	**0.574**	**57.12**	**34.27**
125	0.557	0.562	68.73	41.24
150	0.571	0.579	71.60	42.96
175	0.544	0.548	93.19	55.92
200	0.538	0.544	117.90	70.74
Q, %	U_b	U_{mh}	t, min	t_{item}, sec
25	0.516	0.521	22.29	13.38
50	0.524	0.534	30.45	18.27
75	0.554	0.568	45.35	27.21
100	**0.559**	**0.574**	**57.12**	**34.27**
125	0.571	0.583	67.48	40.49
150	0.553	0.560	76.33	45.80
175	0.544	0.551	98.35	59.01
200	0.583	0.591	111.14	66.84

TABLE A4 | Set of objects included in group A: cuboids.



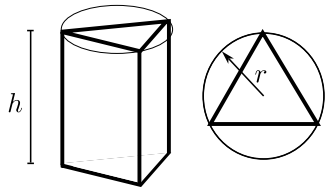
Item	*l*	*w*	*h*	#	Item	*l*	*w*	*h*	#	Item	*l*	*w*	*h*
cuboid_1	0.1	0.1	0.1	#	cuboid_16	0.11	0.09	0.03	#	cuboid_31	0.13	0.13	0.03
cuboid_2	0.1	0.1	0.05	#	cuboid_17	0.11	0.09	0.07	#	cuboid_32	0.13	0.13	0.07
cuboid_3	0.1	0.05	0.05	#	cuboid_18	0.11	0.09	0.09	#	cuboid_33	0.13	0.13	0.09
cuboid_4	0.03	0.03	0.03	#	cuboid_19	0.11	0.11	0.03	#	cuboid_34	0.13	0.13	0.11
cuboid_5	0.05	0.05	0.05	#	cuboid_20	0.11	0.11	0.07	#	cuboid_35	0.13	0.13	0.13
cuboid_6	0.07	0.03	0.03	#	cuboid_21	0.11	0.11	0.09	#	cuboid_36	0.15	0.1	0.1
cuboid_7	0.07	0.07	0.03	#	cuboid_22	0.11	0.11	0.11	#	cuboid_37	0.15	0.1	0.05
cuboid_8	0.07	0.07	0.07	#	cuboid_23	0.13	0.03	0.03	#	cuboid_38	0.15	0.05	0.05
cuboid_9	0.09	0.03	0.03	#	cuboid_24	0.13	0.07	0.03	#	cuboid_39	0.15	0.15	0.1
cuboid_10	0.09	0.07	0.03	#	cuboid_25	0.13	0.09	0.07	#	cuboid_40	0.15	0.15	0.05
cuboid_11	0.09	0.07	0.07	#	cuboid_26	0.13	0.09	0.09	#	cuboid_41	0.15	0.15	0.15
cuboid_12	0.09	0.09	0.09	#	cuboid_27	0.13	0.11	0.03	#	cuboid_42	0.16	0.03	0.03
cuboid_13	0.11	0.03	0.03	#	cuboid_28	0.13	0.11	0.07	#	cuboid_43	0.25	0.01	0.1
cuboid_14	0.11	0.07	0.03	#	cuboid_29	0.13	0.11	0.09	#	cuboid_44	0.25	0.05	0.1
cuboid_15	0.11	0.07	0.07	#	cuboid_30	0.13	0.11	0.11					

TABLE A5 | Set of objects included in group B: cylinders.



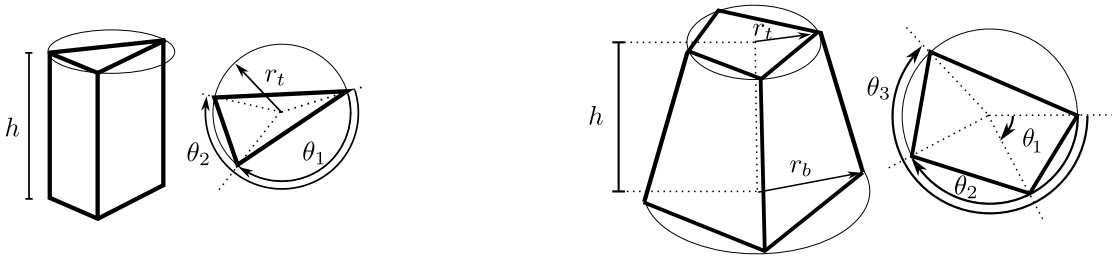
Item	*r*	*h*	#	Item	*r*	*h*	#	Item	*r*	*h*
cylinder_1	0.02	0.1	#	cylinder_9	0.03	0.03	#	cylinder_17	0.04	0.07
cylinder_2	0.02	0.2	#	cylinder_10	0.03	0.05	#	cylinder_18	0.04	0.12
cylinder_3	0.02	0.03	#	cylinder_11	0.03	0.07	#	cylinder_19	0.05	0.1
cylinder_4	0.02	0.05	#	cylinder_12	0.03	0.12	#	cylinder_20	0.05	0.2
cylinder_5	0.02	0.07	#	cylinder_13	0.04	0.1	#	cylinder_21	0.05	0.03
cylinder_6	0.02	0.12	#	cylinder_14	0.04	0.2	#	cylinder_22	0.05	0.05
cylinder_7	0.03	0.1	#	cylinder_15	0.04	0.03	#	cylinder_23	0.05	0.07
cylinder_8	0.03	0.2	#	cylinder_16	0.04	0.05	#	cylinder_24	0.05	0.12

TABLE A6 | Set of objects included in group C: equilateral prisms.



Item	n	r	h	#	Item	n	r	h	#	Item	n	r	h
prism_triang_1	3	0.018	0.096	#	prism_triang_14	3	0.042	0.092	#	prism_penta_8	5	0.03	0.072
prism_triang_2	3	0.023	0.036	#	prism_triang_15	3	0.043	0.068	#	prism_penta_9	5	0.033	0.064
prism_triang_3	3	0.024	0.057	#	prism_triang_16	3	0.044	0.07	#	prism_penta_10	5	0.035	0.045
prism_triang_4	3	0.024	0.079	#	prism_triang_17	3	0.045	0.032	#	prism_penta_11	5	0.037	0.046
prism_triang_5	3	0.024	0.097	#	prism_triang_18	3	0.045	0.074	#	prism_penta_12	5	0.04	0.064
prism_triang_6	3	0.03	0.082	#	prism_triang_19	3	0.048	0.032	#	prism_penta_13	5	0.042	0.046
prism_triang_7	3	0.031	0.098	#	prism_penta_1	5	0.015	0.074	#	prism_penta_14	5	0.044	0.075
prism_triang_8	3	0.032	0.051	#	prism_penta_2	5	0.017	0.058	#	prism_penta_15	5	0.046	0.089
prism_triang_9	3	0.032	0.068	#	prism_penta_3	5	0.019	0.046	#	prism_penta_16	5	0.049	0.081
prism_triang_10	3	0.037	0.078	#	prism_penta_4	5	0.021	0.099	#	prism_penta_17	5	0.051	0.032
prism_triang_11	3	0.041	0.055	#	prism_penta_5	5	0.024	0.087	#	prism_penta_18	5	0.043	0.07
prism_triang_12	3	0.041	0.076	#	prism_penta_6	5	0.026	0.055	#	prism_penta_19	5	0.056	0.056
prism_triang_13	3	0.042	0.08	#	prism_penta_7	5	0.028	0.092	#	prism_penta_20	5	0.058	0.091

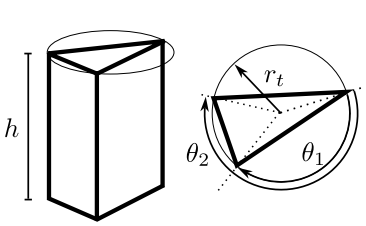
TABLE A7 | Set of objects included in group D: scalene triangular prisms and right pyramidal frusta.

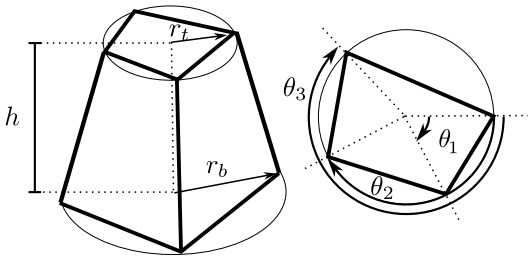


Item	n	h	r_b	r_t	θ	#	Item	n	h	r_b	r_t	θ
scal_triang_1	3	0.059	0.017	0.017	162 178	#	frustum_11	5	0.085	0.031	0.011	73 145 215 288
scal_triang_2	3	0.032	0.02	0.02	120 155	#	frustum_12	5	0.101	0.023	0.01	73 145 215 288
scal_triang_3	3	0.069	0.022	0.022	31 267	#	frustum_13	5	0.127	0.02	0.009	73 145 215 288
scal_triang_4	3	0.053	0.024	0.024	91 115	#	frustum_14	5	0.143	0.035	0.013	73 145 215 288
scal_triang_5	3	0.038	0.025	0.025	95 145	#	frustum_15	5	0.068	0.033	0.016	73 145 215 288
scal_triang_6	3	0.04	0.028	0.028	30 206	#	frustum_16	5	0.131	0.02	0.011	72 145 215 289
scal_triang_7	3	0.046	0.032	0.032	20 150	#	frustum_17	5	0.064	0.027	0.013	72 145 215 289
scal_triang_8	3	0.074	0.033	0.033	180 215	#	frustum_18	5	0.131	0.028	0.013	72 145 215 289
scal_triang_9	3	0.045	0.034	0.034	25 185	#	frustum_19	5	0.071	0.015	0.007	72 145 215 289
scal_triang_10	3	0.085	0.035	0.035	28 220	#	frustum_20	6	0.078	0.035	0.009	60 120 180 240 300
scal_triang_11	3	0.097	0.037	0.037	71 186	#	frustum_21	6	0.147	0.019	0.007	60 120 180 240 300
scal_triang_12	3	0.055	0.041	0.041	15 110	#	frustum_22	6	0.06	0.013	0.007	60 120 180 240 300
scal_triang_13	3	0.098	0.042	0.042	140 244	#	frustum_23	6	0.065	0.014	0.011	60 120 180 240 300
scal_triang_14	3	0.073	0.046	0.046	23 100	#	frustum_24	6	0.12	0.02	0.008	60 120 180 240 300

(Continues)

TABLE A7 | (Continued)





Item	n	h	r_b	r_t	θ	#	Item	n	h	r_b	r_t	θ
scal_triang_15	3	0.031	0.047	0.047	178 198	#	frustum_25	6	0.099	0.025	0.012	60 120 180 240 300
scal_triang_16	3	0.072	0.047	0.047	37 285	#	frustum_26	6	0.127	0.037	0.012	60 120 180 240 300
scal_triang_17	3	0.032	0.05	0.05	80 174	#	frustum_27	6	0.117	0.016	0.008	60 120 180 240 300
frustum_1	4	0.094	0.015	0.01	90 180 270	#	frustum_28	6	0.126	0.038	0.01	60 120 180 240 300
frustum_2	4	0.139	0.03	0.011	90 180 270	#	frustum_29	6	0.122	0.03	0.009	60 120 180 240 300
frustum_3	4	0.083	0.022	0.009	90 180 270	#	frustum_30	6	0.104	0.013	0.009	60 120 180 240 300
frustum_4	4	0.079	0.04	0.013	90 180 270	#	frustum_31	4	0.051	0.135	0.029	92 188 270
frustum_5	4	0.09	0.034	0.012	90 180 270	#	frustum_32	4	0.061	0.052	0.046	83 194 275
frustum_6	4	0.145	0.035	0.011	90 180 270	#	frustum_33	5	0.109	0.067	0.047	52 140 206 276
frustum_7	4	0.129	0.025	0.009	90 180 270	#	frustum_34	5	0.135	0.065	0.025	70 130 230 285
frustum_8	4	0.121	0.032	0.011	90 180 270	#	frustum_35	6	0.095	0.138	0.026	53 114 190 234 303
frustum_9	5	0.056	0.017	0.01	73 145 215 288	#	frustum_36	6	0.139	0.064	0.025	68 47 122 172 221 287
frustum_10	5	0.08	0.02	0.008	72 145 215 289	#						

considered when computing SDFs (see Equations (15) and (18)), without providing significant improvements in packing performance.

Finally, as parameter c decreases from 0 to -0.007, the box utilization $\overline{U}_b$ shows a clear improvement, indicating that moderately negative values of c favor configurations that better exploit the available space. However, further decreasing c does not provide additional benefits, as shown by the slight decrease in $\overline{U}_b$ at $c = -0.009$. While these improvements are accompanied by a moderate increase in computational time, related again to the influence of c on the portion of space considered when computing SDFs, the variations in $\overline{t}^{all}_{item}$ and $\overline{t}^{best}_{item}$ remain relatively limited compared to the gains in packing utilization.

Overall, the analysis suggests that the algorithm is relatively insensitive to moderate variations of the loss-function parameters. Nevertheless, selecting appropriate values for a, q, and c remains important in order to achieve a good compromise between packing performance and computational efficiency.

Appendix A.2. Sensitivity to parameter changes

The sensitivity of parameters M, Q, Γ, and γ is evaluated through a *space-utilization test*. Evaluation is performed by varying each parameter across increasing values within a simulation (Section 5.1), which attempts to pack a given item set inside a fixed-size box while maximizing the occupied volume. Objects in the item set are randomly sampled from the groups listed in Appendix A.3 by first selecting a group and then sampling an item within that group, with repetitions allowed. Performance is evaluated in terms of the U_b and U_{mh} scores defined in Equation (19) (see Section 5), as well as computational time. The experiment is designed as a sensitivity analysis in which one parameter is varied at a time while the others are kept fixed, allowing the impact of each parameter on packing performance to be assessed. In particular, the parameter values reported in Table 1 are used as reference values, and variations corresponding to 25%, 50%, 75%, 125%, 150%, 175%, and 200% of these values are evaluated.

The analysis results, reported in Table A3, highlight different roles for the algorithm parameters. Due to its role in the main optimization process, the coarse voxel size Γ significantly affects packing performance. Increasing Γ reduces computational time but leads to a noticeable decrease in space utilization due to the lower resolution of the point cloud employed in the coarse search. In contrast, the refined voxel size γ, which only affects the refinement stage of the algorithm, shows limited influence on both performance metrics and runtime, indicating that the algorithm is relatively robust to moderate variations of this parameter. The number of candidates per generation M and the number of generations Q mainly influence computational cost, which increases approximately linearly with their values. This behavior is expected, as these parameters directly affect the number of point cloud manipulations and SDF computations required during optimization. While moderate increases in M and Q can provide slight improvements in packing performance, larger values lead to diminishing returns and substantially longer runtimes. Overall, the reference parameter configuration (100%) provides a balanced trade-off between packing quality and computational efficiency.

Appendix A.3. Lists of items used in the simulations

The set of objects used during packing simulations comprises five different groups, whose elements are generated randomly:

- Group A: cuboids listed in Table A4, characterized by side dimensions $l, w, h,$;
- Group B: cylinders listed in Table A5, characterized by height h and radius r;

- Group C: equilateral prisms listed in Table A6, characterized by height h, number n of edges, and radius r of the circumscribed circumference;
- Group D: scalene triangular prisms and right pyramidal frusta listed in Table A7, characterized by number n of base and top vertices, height h, base radius r_b, top radius r_t, and sequence $\boldsymbol{\theta}$ of central angles between each vertex and the first vertex of the polygon. It is to be noted that in the case of scalene triangular prisms, $r_b = r_t$.