Dapper: Decompose-and-Pack for 3D Printing

Xuelin Chen[1] Hao Zhang[2] Jinjie Lin[1] Ruizhen Hu[3]
Lin Lu[1] Qixing Huang[4] Bedrich Benes[5] Daniel Cohen-Or[6] Baoquan Chen[1]

[1]Shandong Univ. [2]Simon Fraser Univ. [3]Shenzhen Inst. of Adv. Tech. [4]Toyota Tech. Inst. [5]Purdue Univ. [6]Tel Aviv Univ.

Abstract

We pose the *decompose-and-pack* or *DAP* problem, which tightly combines shape decomposition and packing. While in general, DAP seeks to decompose an input shape into a *small number* of parts which can be *efficiently* packed, our focus is geared towards 3D printing. The goal is to optimally decompose-and-pack a 3D object into a printing volume to minimize support material, build time, and assembly cost. We present *Dapper*, a global optimization algorithm for the DAP problem which can be applied to both powder- and FDM-based 3D printing. The solution search is top-down and iterative. Starting with a coarse decomposition of the input shape into few initial parts, we progressively pack a pile in the printing volume, by iteratively docking parts, possibly while introducing cuts, onto the pile. Exploration of the search space is via a *prioritized* and *bounded beam search*, with breadth and depth pruning guided by local and global DAP objectives. A key feature of Dapper is that it works with *pyramidal* primitives, which are packing- and printing-friendly. Pyramidal shapes are also more general than boxes to reduce part counts, while still maintaining a suitable level of simplicity to facilitate DAP optimization. We demonstrate printing efficiency gains achieved by Dapper, compare to state-of-the-art alternatives, and show how fabrication criteria such as cut area and part size can be easily incorporated into our solution framework to produce more physically plausible fabrications.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—[Curve, surface, solid, and object representations]

Keywords: Decompose-and-pack, 3D printing, pyramidal shape

1 Introduction

Decomposition and packing are two of the most important and well-studied optimization problems in geometry. In the classical setting, decomposition seeks to partition a given object into a small number of parts with each satisfying a desirable geometric property, and packing involves placing a set of given objects into a given container. Each of the two problems is difficult on its own and many instances of the two problems are known to be NP-hard. Combining the two problems is likely to increase the problem's complexity.

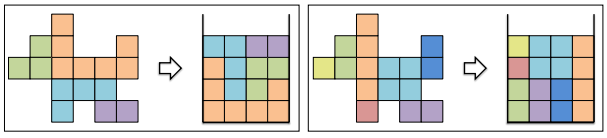
In this paper, we pose the *decompose-and-pack* (DAP) problem, which tightly combines shape decomposition and packing. In general, DAP seeks to decompose an input shape into a *small number* of parts which can be *efficiently* packed. The unique challenges to this problem are two-fold. First, unlike conventional packing, there is no prescribed set of parts to pack; the packing set, which is the decomposition itself, is continuously altered. At the same time, unlike classical shape decomposition, there is no pre-determined part property to drive the decomposition. It is unclear which stand-alone shape property implies "packability", which is a global property of the set of parts to be packed.

Figure 1: *A 3D Inukshuk model (left) is decompose-and-packed (middle) for powder-based 3D printing. The fabricated pieces are assembled and glued together to form the final object (right).*

The general DAP problem is quite intriguing and it can be instantiated in a variety of ways to serve different applications. The focus of our current work is geared towards 3D printing. Decomposition is a natural choice when printing a 3D object which is too large to fit into the printing volume [Luo et al. 2012]. However, our main motivation is to not only decompose, but also efficiently pack the components for printing to *save print time and material*. Specific to powder-based printing, for example, the main objective is to minimize the height of the packing configuration in a given print volume. For fused deposition modeling (FDM), the main objective is to minimize the volume of overhang which requires support material that is wasteful and at times, difficult to remove.

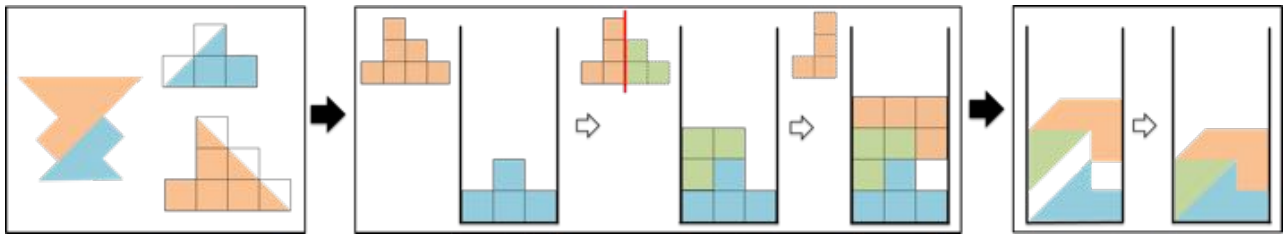
Aside from the geometry criteria, the DAP objectives must be balanced by *part count*, since otherwise the partition may contain many small parts that clearly pack well. As the parts must be glued or locked up by connectors to reproduce the input object, a low part count is preferred to reduce excessive seams, lower assembly cost, and improve integrity of the final product.

Given an input shape and necessary specifications of the printing volume, we define an objective function which evaluates all *partial* solutions to DAP. In a partial solution, a subset of the shape parts are packed in the volume, forming a *pile*. A complete DAP solution requires all parts to be packed. The objective function evaluates



(a) 4 pyramidal parts. (b) 7 boxes.

Figure 2: *Decompose-and-pack polyonimoe shapes using pyramidal primitives (a) vs. box primitives (b). Both can pack perfectly without gap, but fewer pyramidal primitives are required.*



(a) Pyramidal polycubes. (b) DAP with CAP and PAK moves. (c) Local refinement.

Figure 3: *Overview of Dapper algorithm. Starting with an initial pyramidal decomposition of the input shape into few parts, we voxelize the parts into polycubes (a). During global DAP optimization and through a bounded beam search, these parts, possibly cut, are progressively packed into the printing volume (enclosed by dark lines). Shown in (b) are two moves, the first a cut-and-pack (or* CAP*) and the second a pack (*PAK*). Finally, local refinement optimization further reduces gaps between the actual shape parts (c).*

any pile where both part count and fabrication efficiency, e.g., minimization of pile height, are accounted for.

We develop an algorithm, called *Dapper*, which seeks a globally optimal decompose-and-pack solution based on the objective function. The solution search is *top-down* and iterative, corresponding to top-down expansion of a search tree. We start with a coarse decomposition of the input shape into few initial parts. At each iteration, the algorithm performs a set of *docking tests* each of which finds the best way to match the boundary of a particular unpacked part, possibly while introducing a cut, to the boundary of the packed pile in order to minimize gap. A docking bears resemblance to playing the game of *Tetris*. However, here the Tetris pieces are not pre-determined, but computed on-the-fly. These docking tests yield a set of candidate moves, each with an associated *priority score* which reflects its potential to lead to good DAP solutions. The priority scores induce a *beam search* [Lowerre 1976] to progress the DAP solution. With the beam width controlling the width of the top-down expansion of the search tree, depth pruning is achieved by using a *bound* provided by the objective function.

Finding the optimal DAP solution over all possible decomposition and packing combinations is an intractable global optimization problem. Through a bounded beam search, we explore only a restricted search space; hence, any mentioning of a globally optimal DAP solution in our subsequent discussions surely refers to optimality within the search space explored. That being said, even with a modest beam width, the search space is still immense. To make the problem more tractable, we discretize the global search by voxelizing the shape parts and restricting Dapper to only work with *polycubes*. In turn, the polycubes can only be rotated by a 90°x angle during Tetris packing. After the global optimization phase, we peel the boxy boundaries of the polycubes, extract the actual shape parts, and "wiggle" them to pack more tightly. This final gap-reducing optimization only involves *local* refinement. An overview of all the key steps of Dapper can be found in Figure 3.

With the Dapper framework set up, an important algorithm design question still remains: what primitives should we work with for decomposition and packing? Decomposition into arbitrarily-shaped parts can lead to low part counts, but packing arbitrary shapes overly complicates the problem. On the other hand, boxes are "packing-friendly", but they may be too simple and induce too many parts in the decomposition. In this paper, we work with *pyramidal* primitives [Hu et al. 2014], extending their utility from individual fabrication to the DAP setting.

Pyramidal shapes are terrain-like shapes, 2D or 3D, with a flat base. They include boxes, but are more general and lead to decompositions with much fewer parts; see Figure 2. These shapes are not only packing-friendly, as each has a flat side and they pack well with proper docking, but also "printing-friendly", since no support material is needed to print them upright via layered additive fabrication. This factor is especially crucial when the decomposed parts are large and hardly allow much packing in a limited print volume. Docking tests between pyramidal parts reduce to matching between functions. Last but not the least, the set of pyramidal shapes are closed under vertical and horizontal cuts. In Dapper, we start with an initial pyramidal decomposition of the input shape, voxelize the resulting parts, and work with pyramidal polycubes and axis-aligned straight cuts throughout global optimization.

Dapper is quite general and equally applicable to 2D and 3D shapes and to shapes with arbitrary topology. However, with a focus on pyramidal primitives, we process *solid* presentations of 3D objects rather than shell forms. We demonstrate Dapper on numerous shapes, evaluate its performance, and show fabrication result.

2 Background and related work

We first cover some 3D printing basics, explaining the optimization criteria for efficient fabrication, then discuss existing works on decomposition and packing relevant to 3D printing. A 3D printer takes as input a free-form 3D object which is not confined by the printer's physical setup. At the same time, the shape of the printed mass inside the printing volume is also adjustable. As a result, there is room for clever optimization to possibly reoriented the input object, decompose it into 3D-printable parts, or intelligently arrange the resulting parts in the print volume.

Efficient 3D printing. Most 3D printing technologies are based on *additive manufacturing*, which involves sequential-layer material addition or joining under automated control [Wikipedia 2014]. As the layers are built bottom-up, "filler" layers must be produced to support object parts to be printed on top of others. For FDM-based printing, the support material is removed after printing and regarded as waste, while powder-based printing allows reuse of the filler material. More critical to powder-based printing is the maximum height of the printed mass since powder material has to fill the entire printing volume, layer by layer, up to that height.

Decomposition allows the fabrication of objects which do not fit inside the print volume. A smart packing allows multiple parts to be piled up and printed at a time, leading to better utilization of the print volume, material, and time. For powder-based printing, minimizing the maximum height of the pile is most essential, while a tight packing reduces time spent on laying out filler material. For FDM, a low pile is also desirable. The pile does not need to fill the printing volume, but it is preferably pyramidal [Hu et al. 2014] with a tight packing to minimize waste material and print time.

3D Packing. In engineering and manufacturing domains, some methods pack 3D parts for efficient layered fabrication [Dickinson and Knopf 2002] and rapid prototyping [Gogate and Pande 2008]. The general packing problem has many variants in many application settings, including packing/loading, scheduling, and routing [Crainic et al. 2012]. Most instances of the packing problem are NP-hard, hence heuristic-driven exhaustive or stochastic search including branch-and-bound, simulated annealing, and genetic algorithms are frequently adopted [Cagan et al. 2002].

The key difference between Dapper and these previous works is that in our problem, the shapes to be packed are not given *a priori*; they are the result of a simultaneous shape decomposition and are modified on-the-fly. The coupling not only changes the nature of the problem but also increases the search space significantly.

Decomposition for 3D printing. The problem of decomposing 3D objects into solid parts for efficient fabrication has drawn interests from graphics researchers recently. Perhaps the first attempt was made by Luo et al. [2012], where they develop a tool to partition a 3D model so that each part can fit in the printing volume. Factors including low part count, assemblability, and aesthetics of the cuts are taken into consideration, but the method is oblivious to support material or build time. The work by Hildebrand et al. [2013] is motivated by the direction bias of additive manufacturing. It poses the interesting problem of decomposing a 3D shape into few pieces so that each piece can be consistently sliced with small geometric error along one of three orthogonal slicing directions. Recent work by Hu et al. [2014] decomposes a 3D object into a small number of approximately pyramidal parts so that each can be printed by an FDM printer with low cost in time and material.

Dapper also computes solid decompositions in 3D and seeks low part counts. However, we add packing, a new dimension, to the decompose-to-print problem. Smart packing strategies allow printing multiple parts at a time, leading to better utilization of the print volume while incurring small time and material cost.

Boxelization. Recent work by Zhou et al. [2014] "boxelizes" a 3D shape by voxelizing it, computing a tree linkage structure over the voxels, and folding the linkage structure into a box. There is slight resemblance between boxelization and Dapper: voxelization and unlinking of voxels can be seen as a special form of decomposition; folding into a box is a special form of packing. The resemblance would stop there though as the two problems have different goals and operate on completely different primitives (linked voxels vs. discrete set of polycubes) and constraints.

Discrete scissor congruence. More closely related to DAP is the intriguing Scissoring Congruence problem, which disects two shapes into identical sets of segments. Note that this problem is identical to DAP if its target container has the same area/volume as the input shape. Zhou and Wang [2012] discretize the problem on a grid and solve Discrete Scissoring Congruence (DSC) by recursive *co-disection* of the two input shapes. Our DAP problem has a different goal: it decompose-and-packs an input shape into an open-ended volume as tightly as possible while minimizing part count. In contrast, DSC works with a fixed target container and seeks exact congruence without setting an objective to minimize the number of segments. Both problems require difficult global optimization. To counter local minima, the DSC solution applies random exploration of hundreds of pairs of initial seeds for the co-disection.

PackMerger. PackMerger [Vanek et al. 2014] appears to be the first work which considers improving a decomposition for better packing. The method first converts an input 3D object into shells, while Dapper works with solid decomposition. Moreover, PackMerger works with terahedralized input and imposes additional restrictions on the input object, possibly leading to uneven gluing boundaries. Dapper, on the other hand, only makes planar cuts.

Algorithmically, while Dapper follows a top-down search, PackMerger is bottom-up. In PackMerger, the initial set of segments is obtained with packing in mind, e.g., larger cut areas are preferred. However, the consideration of packing is only implicit, the decomposition is not coupled with any packing process. After the initial decomposition, PackMerger executes a merge-and-pack step which again decouples the two tasks; it corresponds to a "merge-and-*then*-pack" process. Specifically, this step randomly explores mergings between segments, and for each merging, recomputes a tight packing with a fixed set of segments. The search merely amounts to a random sampling of packing solutions. The final solution is selected as the merging configuration leading to the best packing.

In our work, we adhere to the true spirit of DAP, emphasizing the strong coupling, rather than interleaving the solution by solving a decomposition and *then* a packing, independently.

The main potential advantage of PackMerger over our algorithm is that it searches over *arbitrary* rotations of the shape pieces during packing. Dapper could allow this as well. However, the coupling of decompose and pack significantly increases the search space over any decoupled approach, making a refined discretization of the space of rotations prohibitive. As the experimental comparison results would later reveal, the quality and computational advantages afforded by Dapper with only axial rotations appear to outweight any gains offered by PackMerger with the consideration of arbitrary rotations. Indeed, the difference is mainly in the optimization or search strategy, not the discretization of search spaces.

3 Decompose-and-pack pyramidal shapes

Our goal is to optimally decompose-and-pack a 3D input object into a container, the printing volume, for efficient fabrication. The input to the Dapper algorithm is a *solid* 3D object, along with a target container which is a rectangular cuboid. In our coverage, to ease understanding of the algorithm, we mainly illustrate it in 2D. We assume that the container has sufficient height to admit a packing of all the decomposed parts. However, the horizontal dimensions of the container are limiting, so that the input shape cannot fit inside without being cut into smaller pieces. The challenge is to efficiently pack the input using a small number of pieces.

3.1 Overview

Dapper starts with an initial partitioning of the input shape into few pyramidal parts. The parts are voxelized into pyramidal polycubes. Dapper then applies a global DAP optimization over the polycubes. After packing the polycubes into the container, the positions of the enclosed shape parts are locally refined to obtain a lower and tighter packing. Figure 3 outlines the major steps of Dapper.

In the global optimization phase (Section 4.1), we decompose and pack pyramidal polycubes. We pictorially regard the container as a pile of packed polycubes. Each pile corresponds to a partial DAP solution and a complete solution is reached when all polycubes are packed. Our solution search is top-down and iterative. At each iteration, we generate a set of *candidate moves*, each of which involves packing a part onto the pile, possibly after a part decomposition. Iterative selection of candidate moves advances the DAP solution and forms a search tree. Expansion and pruning of the search tree are based on a *priority score* we define for each candidate move and a *global* objective function we define for each DAP solution.

Our solution strategy (Section 4.2) is a *bounded beam search*. At each iteration, we prioritize the set of candidate moves based on a scoring function $\mathscr{M}$. The set of top β moves define the front of a beam search [Lowerre 1976], with a user-defined beam width β. We define an objective function $\mathscr{O}$ for any DAP solution and store the objective function value for the best *complete* DAP solution found so far. This function value serves as a bound to prune certain branches of the search tree. The bounded beam search returns a complete DAP solution which attains the optimal value in the objective function $\mathscr{O}$ within the explored search space.

At each iteration of the search, the set of candidate moves are obtained by performing a set of *docking tests*. Docking searches for the geometric transformation which *best aligns* the convexities (respectively, concavities) of a part with the matching concativities (respectively, convexities) of a larger part. In our case, we dock pyramidal polycubes onto the pile in the container. Each candidate move allows an unpacked part to be docked directly, or first cut and then docked, onto the pile. All cuts are axis-aligned and straight, resulting in smaller polycubes which remain pyramidal. When searching for possible cuts, desirable constraints, such as those related to physical fabrication or assembly, can be imposed (see Section 4.4). When docking a polycube, we allow it to be rotated with angles that are multiples of 90°, like in *Tetris*.

In the final local refinement step (Section 4.5), we develop a continuous optimization scheme to reduce the gaps between the actual shape parts by rigidly transforming them. We introduce an optimization formulation, whose objective function includes various objective terms, and the constraint set ensures that the pieces do not collide with each other during their movement.

3.2 Use of pyramidal primitives

The Dapper algorithm described above could work with arbitrarily shaped primitives. In our paper, we advocate the use of *pyramidal* shapes for decomposition and packing. Pyramidal shapes are both packing- and printing-friendly, providing an ideal fit to our problem. At the same time, pyramidal shapes are more general than boxes to lead to decompositions with much fewer parts, yet they maintain a suitable level of simplicity to facilitate both docking and cutting during DAP optimization.

Simplicity and efficiency. Having a restricted class of primitives reduces the computational cost considerably. Pyramidal shapes gain their efficiency from their representational simplicity: they are 2.5D with a flat base; only one side (the top side) needs to be processed. This immediately suggests efficient matching or docking operations based on common image matching techniques.

Closure. Pyramidal shapes are *closed* under vertical and horizontal (in reference to their up orientations) cuts. Since we introduce axial cuts on-the-fly, it is vital that the pyramids are split into two pyramids. Note that convex shapes are also closed under arbitrary cuts, but they cannot be fully represented by a single depth image. In rare cases, a pyramidal shape can be split into more than two parts with a single cut. We then limit the cut to produce only two parts, to maintain a low part count.

Packing- and printing-friendly. Exact pyramidal parts can be printed in their upright orientations with no support material. Also, in case our DAP solution overflows the printing volume, the leftover piece after a horizontal cut can be printed separately without extra treatment to handle overhangs. Pyramidal shapes are also packing-friendly: not only are they one-sided, facilitating docking, also the opposite sides are flat and do not introduce any gaps to fill. Boxes might be a natural primitive to consider for DAP as they are both packing- and printing-friendly, but they are too simple to likely demand a decomposition with too many parts. Pyramidal shapes are almost as friendly as boxes, and at the same time, they are generic enough to yield more compact decompositions.

4 Dapper algorithm

Given an input shape S and a printing volume or container C, we obtain the initial set of parts by pyramidal decomposition [Hu et al. 2014] using a low part count. We upright-orient each initial pyramidal part along its up direction and voxelize the part at a user-specified resolution as follows. Let X^*, Y^*, and Z^* be the lengths of the tightest axis-aligned bounding box (AABB) of the input model S along the x, y, and z dimensions, respectively. We set the side length of the voxels as $R_{\text{vox}} \cdot \min(X^*, Y^*, Z^*)$. Without loss of generality, assume that the smallest dimension is x. Then we increase the y and z dimensions of the AABB so that they are multiples of $R_{\text{vox}} \cdot X^*$. The voxelization is uniform over the possibly expanded AABB and a voxel is set to 1 if any part of the input model intersects that voxel. By default, we set $R_{\text{vox}} = 0.1$.

The set of voxels tightly bound the pyramidal part with its base coinciding with voxel boundaries. The resulting shape for global DAP analysis is thus a pyramidal polycube with a designated up orientation. For simplicity, we use the term voxel to reference the grid-based representation in both 2D and 3D.

Next, we detail our algorithm for global DAP optimization, where the input consists of an initial set of pyramidal polycubes and the output is a packed pile in the given container C. Our coverage is focused on powder-based 3D printing and later, we explain how the global objective function and the search priority are adjusted for FDM-based printing (Section 4.3). Local refinement is executed after global DAP to locally optimize positioning of the actual shape parts to procude a lower pile and tighter packing.

4.1 Global DAP for powder-based printing

For powder-based 3D printers, the goal is to minimize the maximum height of the pile in the container C while ensuring a low part count. We formulate DAP for powder-based fabrication as,

$$P^* = \underset{\text{full pile } P}{\operatorname{argmax}}\, \mathscr{O}_{\text{PWD}}(P) = \underset{\text{full pile } P}{\operatorname{argmax}}\, \frac{H(C) - H(P)}{N^\alpha}, \quad (1)$$

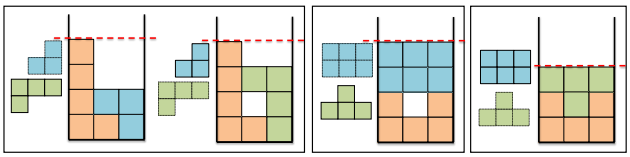
with the global objective function $\mathscr{O}_{\text{PWD}}$, where $H(P)$ denotes the maximum height of the pile P, N is the number of parts in the pile, $H(C)$ is a constant that represents the height of the container C, and α is a tunable parameter to trade-off between part count and pile height. The global optimization seeks to maximize the objective function $\mathscr{O}_{\text{PWD}}$. The optimal solution P^* is attained by a full pile, a pile consisting of all parts.

Each partial solution to DAP consists of a set $\mathcal{U}$ of unpacked polycubes and a packed pile P in the container consisting of a set of packed polycubes. The search space consists of *sequences* of moves supported by Dapper, where each sequence evolves the initial partial solution (all parts unpacked and empty pile) to a complete solution (all parts packed and a full pile).

Dapper considers two possible moves:

- PAK: This move packs a part from $\mathcal{U}$ onto the pile.
- CAP: This move cuts a part in $\mathcal{U}$ and packs one of the resulting pieces onto the pile.

Note that packing a part corresponds to a docking operation and docking can occur on the *top or side* of the pile P. Figure 3 shows



(a) (b) (c)

Figure 4: *A few options to define the priority score. (a) Maximum height is oblivious to moves which do not increase the maximum height of the pile; two such moves are shown, with the left one being more desirable for packing. (b) Maximizing total height gains as the priority prefers packing larger pieces which may leave vertical gaps. (c) Combining total height gains with a penalty on vertical gaps, a compact packing with smaller pieces can win.*

the two moves in action. All cuts are axis-aligned, straight, and result in one and only one additional part. It is easy to see that the set of pyramidal polycubes is closed under such cuts.

We assume that the container height $H(C)$ is sufficiently large so that all the decomposed parts can be packed. This is unrealistic in practice. With a fixed print volume height $H(C)$, the DAP search ought to terminate when the next part to be packed exceeds the allowed height. We do not insist on packing this part by cutting it. Instead, the remaining unpacked parts serve as input to the next Dapper iteration with an empty printing volume.

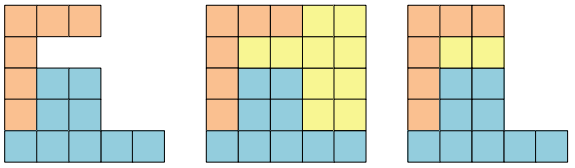
4.2 DAP search for powder-based printing

We explore the search space for DAP using a search tree, through a bounded beam search. The search tree is β-ary, where β is the beam search width. Each node of the tree stores a supported move (PAK or CAP), with the root starting as empty. At each node, we expand the search tree by generating β children, corresponding to the top β candidate moves according to a priority score $\mathscr{M}$. During the search, we keep track of P^* which is the best complete solution found so far, i.e., P^* minimizes the *global* objective function, which, for powder-based printing, is $\mathscr{O}_{\text{PWD}}$ as defined in (1).

The search terminates when a prescribed maximum number of attempted moves is reached or when all solutions have been exhausted. The optimization returns P^* as the final packing solution in the container. The path of nodes in the search tree leading to P^* form the winning sequence of moves. Each node corresponds to either a PAK or a CAP move. The set of cuts defined by the winning sequence of moves define a decomposition of the input shape, while the set of packing moves define the packing.

Priority score. As we build up a sequence of moves, the next move chosen should be one that tends to lead us towards a globally best final solution. We define a priority score to select the candidate next moves. The first choice for the priority is the global objective function. For powder-based printing, this would be $\mathscr{O}_{\text{PWD}}$. However, it is not a proper choice, since $\mathscr{O}_{\text{PWD}}$ depends on maximum height and it cannot distinguish between two moves neither of which increases the maximum height; see Figure 4(a).

Next to consider would be to maximize the *total height gain*. Specifically, for a voxel v in a part that is packed, the height gain at v is the difference between the container height $H(C)$ and the actual height of v in the packed pile. We use the term "gain" to indicate that the measured difference value represents the gain a voxel make by having been packed into the pile. For any voxel that belongs to an unpacked part, its height is assigned to be $H(C)$.



(a) Current pile. (b) Gap for powder. (c) Gap for FDM.

Figure 5: *Different measures of total gap for powder-based (b) and FDM-based (c) printing. The gap voxels are shown in yellow for the pile configuration given in (a).*

Clearly, maximizing total height gains prioritizes low packing on the pile. If gains are measured over all voxels in a part, then preference would also be given to larger parts, which is desirable. However, as illustrated in Figure 4(b), considering only height gains is insufficient in penalizing vertical gaps. These gaps require filler material, for both powder- and FDM-based printing, hence they should be reduced. Combining the factors discussed so far, we arrive at the following priority score $\mathscr{M}_{\text{PWD}}$ for packing a part q onto the pile P, geared towards powder-based printing,

$$\mathscr{M}_{\text{PWD}}(q, P) = \frac{\sum_{v \in P \oplus q} H_{\text{GAIN}}(v, P)}{n^{\alpha}} - \eta \cdot G_{\text{PWD}}(P \oplus q), \quad (2)$$

where $H_{\text{GAIN}}(v, P)$ is the height gain at voxel v in the combined pile $(P \oplus q)$ after q is packed onto P, $G_{\text{PWD}}(P \oplus q)$ is the total number of voxels accouting for all the vertical gaps or filler material necessary in the combined pile, n is the total number of packed as well as unpacked parts in the current partial DAP solution, and η is a tunable parameter as the penalty weight for vertical gaps. Moves associated with higher priority scores are preferred. Figure 4(c) shows that by combining the considerations of height gain and total gap, the priority score may prefer more compact packing.

The way total gap is measured in the priority score $\mathscr{M}$ depends on the type of printer. For powder-based printing, the powder is filled up to the height of the pile, hence the total gap is measured as the number of voxels needed to fill the current packing configuration up to the maximum height of the pile; see Figure 5(b).

Docking and cutting polycubes. Given the set $\mathcal{U}$ of unpacked parts and the current pile P, we perform docking tests for each part in $\mathcal{U}$. For each such part p, we consider six axis-aligned orientations and for each orientation, we performing docking tests against P from five directions (top, left, right, front, and back). When performing a docking test, we treat the two opposite sides of the part and pile as depth images. Docking test is like playing a *Tetris* game. We shift the part over the extent of the horizontal dimensions of the print volume and at each discrete grid position, we "drop" the part onto the pile in the volume.

Note that the docking problem in one-dimensional space is an instance of the string matching problem, which is well-known and for which there are very efficient solutions, e.g., the Aho-Corasick Algorithm [Aho and Corasick 1975]. This particular algorithm allows one to dock multiple pieces simultaneously onto a (1D) pile, while the matching is exact. There are also fast algorithms for inexact string matching and two-dimensional extensions of these fast matching methods are also available. The latter methods would be applicable to the docking problem we have at hand. However, for simplicity, we take an enumerative approach in this work.

In addition to docking tests, CAP moves also require making cuts. We only allow axis-aligned straight cuts to the polycubes and restrict the extent of the cuts, if necessary, to ensure that only one

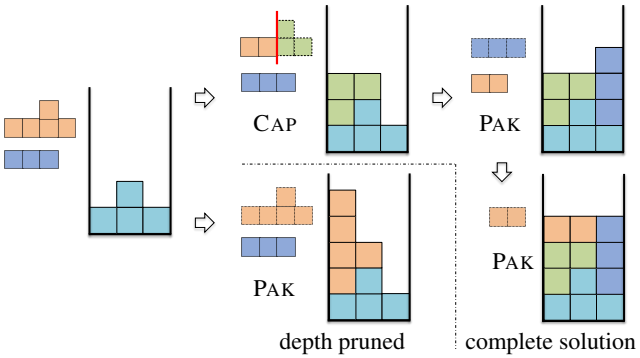


Figure 6: *An illustration of DAP search on a minimal example for powder-based printing; arrows show the search paths. The illustration starts at a partial solution (a node in the search tree) with two pieces (left) to process. By a CAP move and two following PAK moves, the top path runs into a complete solution, while the bottom path is depth pruned, in reference to the complete solution found.*

new part is generated with each cut. We also enumerate cuts along a given dimension, but assign a higher priority to balanced cuts.

Candidate moves. For each docking test, two moves are selected and placed into a pool of possible moves to consider as the candidates. One move accomplishes the lowest docking (maximizing total H_{GAIN}) and the other results in the least amount of total gap G_{PWD}. Among all selected moves in the pool, the algorithm chooses the top β moves, either corresponding to a CAP or a PAK, according to the priority score given in Eqn. (2).

Depth pruning. Limiting the search radii by beam width serves as a search *tree pruning in breadth*. Depth pruning, where an entire subtree of the search tree is cut off, can be accomplished by evaluting DAP solutions at a node and comparing it with the current best full DAP solution based on the global objective function $\mathscr{O}_{\text{PWD}}$. That is, pruning happens only after we already have at least one full solution. Specifically, let P denote the pile associated with the partial DAP solution corresponding to node d in the search tree. Let P^* be the *full* pile found so far which gives the maximum objective function value $\mathscr{O}_{\text{PWD}}$. Our pruning strategy is fairely simple:

$$\text{if } \mathscr{O}_{\text{PWD}}(P) < \mathscr{O}_{\text{PWD}}(P^*),$$

then the subtree at node d is pruned. The pruning strategy is valid only if, under the above condition, no further search expansion is necessary beyond node d. This would be the case as long as no further packing moves can improve the objective function $\mathscr{O}_{\text{PWD}}$.

The key enabling property of $\mathscr{O}_{\text{PWD}}$ which validates the above depth pruning is that, indeed, $\mathscr{O}_{\text{PWD}}$ *never increase* as the solution progresses, i.e., as the moves accumulate. With PAK and CAP moves, this property holds since both moves induce packing of a piece onto the pile. Any packing move would increase the part count, thus increasing the denominator of $\mathscr{O}_{\text{PWD}}$. As well, any packing move either increases the maximum pile height $H(P)$ or keeps it the same, thus decreasing the numerator of $\mathscr{O}_{\text{PWD}}$ or keeping it the same. In neither case, the objective function $\mathscr{O}_{\text{PWD}}$ could increase. Hence, if a partial pile P already loses to P^*, then it can never win over P^* by having more pieces packed onto it.

Figure 6 provides illustrations on a minimal example which includes several key ingredients of our DAP search such as the identification of candidate moves, docking, and depth pruning.

4.3 Global DAP for FDM-based printing

The global DAP optimization scheme described so far has been geared towards powder-based printing. To modify it to optimize for FDM-based printing, we only need to adjust the global objective function $\mathscr{O}$ and the priority score function $\mathscr{M}$.

For FDM-based printers, the goal is to minimize the total amount of overhang or support waste (material) in the packed pile. We thus change the problem formulation to:

$$P^* = \underset{\text{full pile } P}{\text{argmax}}\, \mathscr{O}_{\text{FDM}}(P) = \underset{\text{full pile } P}{\text{argmax}}\, \frac{V(S) - G_{\text{FDM}}(P)}{N^\alpha}, \quad (3)$$

where $V(S)$ is the total number of voxels for the input shape S, $G_{\text{FDM}}(P)$ is the total amount of vertical gaps (overhang) in the pile P, N is again the number of parts in the pile, and α is same as before, a trade-off parameter as in Eqn. (1).

While for powder-based printers, the filler material is filled up to the maximum height of pile, for FDM, the support only need to exist under some parts of the pile. Hence, the total gap $G_{\text{FDM}}(P)$ in (3) for FDM printers is defined as the total number of *empty* or gap voxels which exist *below* some voxels occupied by the current pile P in the printing volume. This is illustrated in Figure 5(c) to contrast (b) for powder-based printers.

Accordingly, we define the priority score $\mathscr{M}_{\text{FDM}}$ for FDM-oriented DAP search as

$$\mathscr{M}_{\text{FDM}}(q, P) = \frac{\sum_{v \in P \oplus q} H_{\text{GAIN}}(v, P)}{n^\alpha} - \eta \cdot G_{\text{FDM}}(P \oplus q), \quad (4)$$

for packing a part q onto the pile P, where n, the free parameter η, and the height gain $H_{\text{GAIN}}(v, P)$ are the same as in Eqn. (2), and $G_{\text{FDM}}(P \oplus q)$ is the total gap defined above for FDM. Note the symmetry that exists in our definitions of the priority scores. For both types of printers, both height-related criterion, H_{GAIN}, and gap related criterion, G_{FDM} or G_{PWD}, are included.

At last, it should not be hard to see that like $\mathscr{O}_{\text{PWD}}$, the global object function $\mathscr{O}_{\text{FDM}}$ also never increases with additional packing moves. All we need to note is that any packing move onto the pile P either increases the total gap $G_{\text{FDM}}(P)$ or keeps it the same. Consequently, depth pruning for FDM-oriented DAP search would employ a similar condition, $\mathscr{O}_{\text{FDM}}(P) < \mathscr{O}_{\text{FDM}}(P^*)$.

4.4 Fabrication criteria

If the only criterion for selecting cuts in Dapper is based on how well the resulting parts dock with the pile, the object pieces produced may possess undesirable characteristics during physical fabrication and assembly, when the 3D-printed pieces are glued or otherwise physically connected to reconstruct the input object. Previous works on decomposition for 3D printing, including Chopper [Luo et al. 2012] and PackMerger [Vanek et al. 2014], all consider fabrication constraints in one way or another.

Our current global DAP optimization framework allows straightforward incorporation of several fabrication constraints via simple thresholding when selecting candidate CAP moves during the solution search. These constraints include:

- **Cut area.** Small cut areas may cause instability when gluing parts over these areas and they make the insertion of physical connectors difficult. We add a threshold r_A, where any CAP move whose cut area is smaller than $r_A \cdot \min(\hat{X}\hat{Y}, \hat{Y}\hat{Z}, \hat{Z}\hat{X})$ is disallowed, where $\hat{X}$, $\hat{Y}$, and $\hat{Z}$ are the x, y, and z voxel dimensions of the input shape, respectively.

- **Part volume:** Small parts are also undesirable for fabrication. We add a threshold parameter r_V, where any CAP move which results in a polycube whose volume is smaller than $r_V * V(S)$ is disallowed. Recall that $V(S)$ is the number of voxels in the voxelization of the input shape S.
- **Part thickness:** To prevent thin structures which are easy to break, we add a threshold parameter r_T, where any CAP move which results in a polycube with some parts whose thickness is less than $r_T * V(S)$ is disallowed.

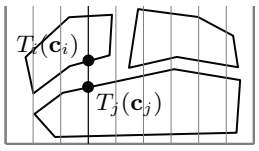
In our current algorithm, all three fabrication constraints are optional. Note that the fabrication constraints considered by PackMerger are precisely cut area and part volume. Any constraint that can be localized to the selection of CAP moves should be easy to incorporate into our solution. However, more global criteria, such as symmetry constraints, are not as straightforward to factor in.

4.5 Local refinement

The last step of the Dapper is a local refinement of the objects. The accuracy of the global decomposition and assembly procedure is restricted by the resolution of the grid. Thus, the goal of the local refinement step is to perform continuous optimization to further diminish the packing potential.

Specifically, suppose we have N pieces $P_i, 1 \leq i \leq N$, obtained from global DAP. We seek to optimize for a rigid transformation $T_i \in SE(3)$ for each piece P_i to improve the assembly and we formulate this as a constrained optimization. The objective function describes the packing potential in terms of minimizing the height and/or the vertical gaps. There are two optimization constraints: (i) the pieces should not penetrate each other, and (ii) the pieces are bounded by the container $B = (\mathbf{l}, \mathbf{u})$, where $\mathbf{l}$ and $\mathbf{u}$ describe the lower and upper corners respectively. Without losing generality, we translate the pieces and the container so that $\mathbf{l} = (0,0,0)^T$, after which the height of the container is u_z.

Packing potential. The packing potential includes two terms. The height term $f_{height} = h_z$ is given by the height of the pile. The vertical gap term discretizes the volume of the vertical gap. Specifically, we uniformly sample the base of the container using a grid and shoot a vertical beam up from each sample (see the inset figure). Each beam is divided by the packed pieces into multiple segments, whose endpoints lie on the piece boundaries. We collect all segments with endpoints from different pieces: $\mathcal{C} = \{(\mathbf{c}_i, \mathbf{c}_j) | \mathbf{c}_i \in P_i, \mathbf{c}_j \in P_j\}$. Without loss of generality, we assume the z coordinate of the location $T_i(\mathbf{c}_i)$ of c_i after transformation T_i is bigger than that of the location $T_j(\mathbf{c}_j)$. With this setup, it is easy to see that the volume of the vertical gap is estimated as



$$f_{gap} = r^2 \sum_{(\mathbf{c}_i, \mathbf{c}_j) \in \mathcal{C}} \mathbf{e}_z^T (T_i(\mathbf{c}_i) - T_j(\mathbf{c}_j)), \tag{5}$$

where r denotes grid resolution and $\mathbf{e}_z$ is the z axis. In our implementation, we set $r = 0.01$ times the width of the printing base.

The objective function combines the height and vertical gap terms:

$$f = f_{height} + \lambda f_{gap}. \tag{6}$$

The tradeoff parameter λ controls the relatively strength of these two terms. The height term is dominant for the powder printer, while for the FDM printer the gap term is dominant.

Constraints. It is straightforward to formulate the bounding constraint i.e., each piece is included in the bounding container:

$$\mathbf{0} \leq T_i(\mathbf{q}_{ik}) \leq \mathbf{u}, \quad \forall \mathbf{q}_{ik} \in CH(P_i), 1 \leq i \leq N \tag{7}$$

and $CH(P_i)$ denotes the convex hull of P_i.

The non-penetration constraints consider the signed distances from points of one piece to other pieces. In our implementation, we discretize the distance function using the point-to-plane distance [Chen and Medioni 1992], which serves as a first-order approximation of the distance function. More precisely, the non-penetration constraints are given by

$$(T_j^{-1} T_i(\mathbf{p}_{il}) - \mathbf{f}_{ijl})^T \mathbf{n}_{ijl} \geq \epsilon, \quad \begin{array}{l} \forall \mathbf{p}_{il} \in P_i, \\ 1 \leq i \neq j \leq N, \end{array} \tag{8}$$

where $\mathbf{f}_{ijl}$ is the closest point to $T_j^{-1} T_i(\mathbf{p}_{il})$ on P_j, and $\mathbf{n}_{ijl}$ is the normal direction at $\mathbf{f}_{ijl}$. When the rigid transformations are optimized, we also update $\mathbf{f}_{ijl}$ and $\mathbf{n}_{ijl}$ so that the first-order approximation is accurate. A user-specified threshold ϵ is used to define the gaps between different pieces. In our implementation, we set $\epsilon = 0.005$ times the width of the container.

Optimization via iterative linear programming. The above-described optimization problem is hard to solve due to the non-linearity of the rigid transformations. We propose to optimize sequentially, so that in each step we solve an easier problem where the rigid transformations are replaced by first-order approximations. After each iteration, we apply a projection operation to obtain rigid transformations for the next step. Note that the closest points and the segment endpoints are also updated. Such a procedure has been widely used in the context of rigid alignment of range scans (c.f. [Chen and Medioni 1992]).

Specifically, denote $T_i = (R_i, \mathbf{t}_i)$, where R_i and $\mathbf{t}_i$ are its rigid and translational components. Let $T_i^c = (R_i^c, \mathbf{t}_i^c)$ denote its value obtained from the previous step. We can approximate T_i in local neighborhood of T_i^c as (c.f. [Chen and Medioni 1992])

$$R_i \approx ((\mathbf{v}_i \times) + I_3) R_i^c, \quad \mathbf{t}_i \approx \mathbf{v}_i \times \mathbf{t}_i^c + \overline{\mathbf{v}}_i, \tag{9}$$

where $\times$ denotes the cross-product operator and $(\mathbf{v}_i, \overline{\mathbf{v}}_i)$ is a vector in the tangent space of $SE(3)$ at T_i^c.

Substituting (9) into (5), (7), (8), and ignoring the quadratic terms in $\mathbf{v}_i$ and $\overline{\mathbf{v}}_i$, we arrive at the following optimization at each step:

$$\begin{array}{ll} \underset{h_z, \{\mathbf{v}_i, \overline{\mathbf{v}}_i\}}{\text{minimize}} & h_z + \lambda r^2 \sum_{(\mathbf{c}_i, \mathbf{c}_j) \in \mathcal{C}} \mathbf{e}_z^T (\mathbf{v}_i \times \mathbf{c}_i + \overline{\mathbf{v}}_i - \mathbf{v}_j \times \mathbf{c}_j - \overline{\mathbf{v}}_j) \\ \text{subject to} & \mathbf{0} \leq \mathbf{q}_{ik}^c + \mathbf{v}_i \times \mathbf{q}_{ik} + \overline{\mathbf{v}}_i \leq \mathbf{u}, \quad \begin{array}{l} \forall \mathbf{q}_{ik} \in \text{CH}(P_i) \\ 1 \leq i \leq N \end{array} \\ & ((\mathbf{v}_i - \mathbf{v}_j) \times \mathbf{p}_{il}^c + (\overline{\mathbf{v}}_i - \overline{\mathbf{v}}_j) - \mathbf{f}_{ijl})^T \mathbf{n}_{ijl} \geq \epsilon, \\ & \forall \mathbf{p}_{il} \in P_i, 1 \leq i \neq j \leq N. \end{array} \tag{10}$$

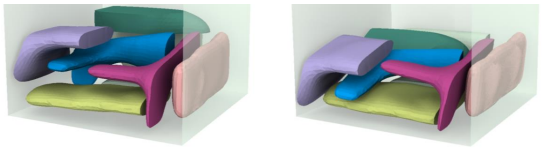
Here $\mathbf{p}^c$ denote the position of $\mathbf{p}$ after the previous step. Since the objective function and the constraints of (10) are linear, equation (10) is a linear program, which can be effectively solved. In our implementation, we employ the CVX package.

The solution to (10) is given by the optimal displacement vectors $\mathbf{v}_i, \overline{\mathbf{v}}_i$ for each piece i. Its rigid transformation is given by

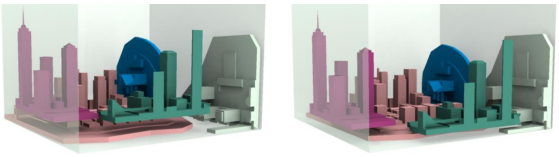
$$R_i = \exp(\alpha \mathbf{v}_i \times) R_i^c$$

and

$$\mathbf{t}_i = \exp(\alpha \mathbf{v}_i \times) \mathbf{t}_i^c + \alpha \overline{\mathbf{v}}_i,$$



(a) Local refinement with further height reduction of 25%.



(b) Local refinement with further gap reduction of 59%.

Figure 7: *Local refinement introduces further improvement for powder-based (top) and FDM printers (bottom), respectively.*

where $\exp(\cdot)$ denotes the matrix exponential map, i.e, the project operator. Parameter $\alpha \in (0, 1]$ is a value that ensures all constraints are satisfied under the resulting rigid transformations. In this paper, we determine α via bisection line search. This process is iterated until the rigid transformations become steady. In our experiments, 10-30 iterations are sufficient for convergence.

Figure 7 shows a few cases where the refinement step led to relatively more significant height reduction over the global optimization step. This further height reduction depends on the voxel resolution. Lower resolution voxelizations induce larger empty spaces in voxels, leading to larger gaps between shape parts after packing polycubes and more height reduction via local refinement.

5 Results

We show virtual and physically fabricated results obtained by Dapper, along with statistics and evaluation. Initial pyramidal decompositions were obtained by code provided by Hu et al. [2014]. We also make qualitative and quantitative comparisons to two related approaches: PackMerger [Vanek et al. 2014] and discrete scissors congruence [Zhou and Wang 2012]. More results and illustrations can be found in the video and supplementary material.

Setting up printing volume. Unless specified otherwise, we assume, for simplicity, that the printing volume is a rectangular cuboid with a square base. We set the default side length $\mathcal{L}_{max}$ of the base as $\mathcal{L}_{max} = 1.618 \cdot \mathcal{V}_{input}^{1/3}$, where $\mathcal{V}_{input}$ is the total volume of the 3D input object. For the 2D examples, we set the length of the bottom side of the container to $\mathcal{L}_{max} = 1.618 \cdot \mathcal{A}_{input}^{1/2}$, where $\mathcal{A}_{input}$ is the total area of the 2D input shape.

Note that these setups of the print volumes are not meant to reflect any design intent or physical meaning. Empirically, they seem to well constrain, but not overly constrain, Dapper to produce interesting and non-trivial solutions for the 3D models we tested. Other horizontal dimensions are also experimented with; we specify the parameters when appropriate. In general, altering these print volume setups does not significantly change the trends we are showing in our evaluation and comparative studies.

Parameters. Aside from the thresholds r_A, r_V, and r_T for setting up fabrication constraints, there are five tunable parameters in the core Dapper algorithm: exponent α on part count in the global objective function (1) or (3); beam search width β; the minimum strip width ω to bound exploration of cuts at each iteration; the penalty weight η on vertical gap in the priority score (2); and the weight λ placed on height minimization during local refinement.

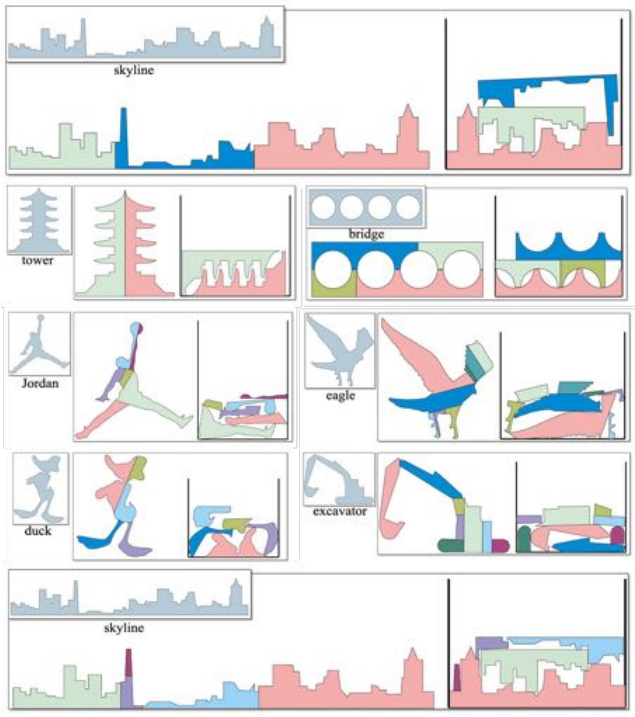


Figure 8: *A gallery of 2D results from DAP with the goal of height minimization. For each example, we show the input shape, the final decomposition, and the packing after local refinement. All results are obtained with the same default parameter setting except for the last row (a different setting $\alpha = 0.1$ and $\eta = 30$ is used).*

Unless otherwise specified, all DAP results shown were obtained by setting $\alpha = 0.3$, $\omega = 2$, $\eta = 10$, and $\lambda = 0.001$ for powder-based printing. For FDM, we make an adjustment to $\lambda = 100$. The parameter β varies between 4 and 8 to control breadth pruning.

Note that we fix the way voxelization resolution is set for an input model, by setting $R_{\text{vox}} = 0.1$. Increasing the resolution does allow the polycubes to reduce in size, hence leading to more compact packing. However, we have found the gains to be generally not sufficiently large to justify the significant increase in search time. Besides, the local refinement step, which is applied to the actual shape parts, often provides a good remedy for the packing inefficiencies caused by the loose bounds given by the polycubes.

DAP results for powder-based fabrication. Figures 8 and 9 show a gallery of 2D and 3D results produced by Dapper, all optimized for powder-based 3D printing (i.e., to minimize height of the packed configuration). The input has a mixture of organic and man-made objects, with varying complexity and geometric characteristics. Note that the skyline model in Figure 8 is already pyramidal hence no initial decomposition was applied. All results are obtained under the default parameter setting (except for the skyline result in the last row of Figure 8), with both global and local optimizations, but *without* imposing fabrication constraints. In Figure 9, we also include photographs showing the physically fabricated pieces produced by a ProJet 660-Pro powder-based printer.

With the default parameters, the skyline result in the first row of Figure 8 leaves large gaps, but boasts a small part count. By setting $\alpha = 0.1$, to lower the influence of part counts, and $\eta = 30$, to penalize more on total gap, we obtain a lower pile shown in the last row. Figure 10 shows a 2D example of how Dapper produces dif-

Figure 9: *A gallery of 3D results from Dapper optimized for powder-based 3D printing (i.e., height minimization). For each example, we show (from left to right) the input shape, the final (virtual) decomposition and packing, and a photograph of the fabricated pieces. Part counts are shown in Table 1.*

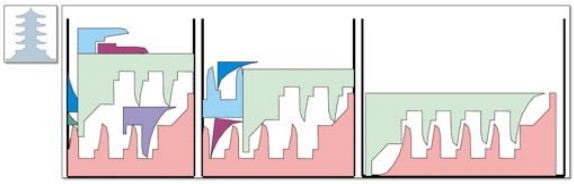
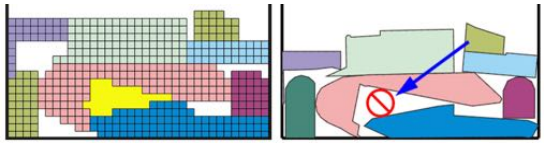


Figure 10: *DAP results with different part counts as container width varies from 25 (8 parts), 30 (5 parts), to 40 (2 parts).*

ferent solutions as the container changes its dimension. Evidently, a more liberally sized container would allow Dapper to produce fewer pieces in the final decomposition.

While many parts produced by Dapper are box-like, there are plenty of decompositions whose parts are far from box-shaped or convex, e.g., see results for the tower, bridge, chair, table, and skyline models. In these cases, a box or convex DAP scheme would have generated more parts. Working with pyramidal primitives appears to strike a good balance between part count and printing efficiency.

Note that the DAP result for the excavator examples in Figure 8 is sub-optimal. A piece at the top of the pile could have been moved down to fill a gap below, thus reducing the overall height of the pile. However, the global DAP optimization works on voxels which could represent loose bounds over the actual shape pieces. As shown in Figure 11(a), such loose bounds would not leave the kind of gaps as the actual pieces would. Moreover, the local nature



(a) (b)

Figure 11: *Sub-optimality of the global and local DAP optimizations. (a) With voxels employed by the global DAP step providing rather loose bounds of the actual shape pieces, the dark green voxelized piece on top could not fit in the yellow gap. The actual piece can (b). However, local refinement cannot move the piece either as only local moves (without interpenetration) are allowed.*

Model	$\#y$	$\#v$	$\#mv$	$\#p$	$\%h$	t_g	t_l
Skyline 1	1	534	(1 ,2)	3	78%	44s	23s
Tower	2	444	(2, 0)	2	48%	50s	20s
Bridge	2	552	(2, 2)	4	63%	54s	86s
Jordan	4	528	(4, 2)	6	-	92s	36s
Eagle	3	378	(3, 5)	8	-	35s	7s
Duck	5	456	(5, 0)	5	33%	75s	22s
Excavator	4	576	(4, 4)	8	-	98s	5s
Skyline 2	1	534	(1, 4)	5	83%	35s	34s
Candelabra	2	2651	(2, 4)	6	67%	49s	20s
Chair	2	538	(2, 3)	5	66%	57s	28s
Bar stool	3	544	(3, 2)	5	81%	33s	13s
Table	2	647	(2, 2)	4	-	27s	57s
Airplane	2	2810	(2, 4)	6	-	94s	14s

Table 1: *Printing efficiency, timing, and other statistics for DAP on examples from Figures 8 and 9. We report the number of initial pyramidal parts ($\#y$), total number of voxels ($\#v$) covering all parts, the number of (PAX, CAP) moves ($\#mv$), the final part count ($\#p$), and percentage* reduction *in height achieved ($\%h$). Note that height reduction is not reported if the input model cannot fit into the container at all. Execute time (in seconds) are reported for both the global (t_g) and local optimization (t_l) phases and accounts for all DAP operations (exincluding pyramidal decomposition). Timing is measured on an Intel(R) Core(TM) i5-4570 with 8GB RAM.*

of the refinement step dictates that it would not be able to move the top pieces to fill the gaps below. Both of these issues reflect current limitations of our DAP optimization.

Table 1 shows timing, printing efficiency improvements, and primitive counts for Dapper, when applied to models in Figures 8 and 9. Between the global and local optimization steps, height reduction is attributed mostly to the former. Local refinement does further reduce height of the final packing, but often only slightly, about 10% on average. Compared to the global DAP step, local refinement is relatively quick, taking 20-30s on average.

When measuring height *reduction*, the reference height for the input shape is the minimum height attained by fitting it, in its entirety, in the container, which has a limited base but unlimited height. While the latter is unrealistic, this setup allows for a meaningful assessment of the height reduction Dapper achieves. If the input model cannot fit into the container, we do not report height reduction.

Fabrication constraints. Figure 12 shows a few photos of the 3D printing process and assembled objects after gluing. Note that gluing for the chair model from Figure 9 was difficult due to small cut areas; the assembled object was not stable. The chair solution also contains a thin part on top. Indeed, without imposing fabrication constraints, the decomposition may result in small cut areas,

Figure 12: *A few snapshots of powder-based 3D printing of pieces produced by Dapper, along with photos of assembled 3D objects from the 3D gallery after gluing the pieces.*

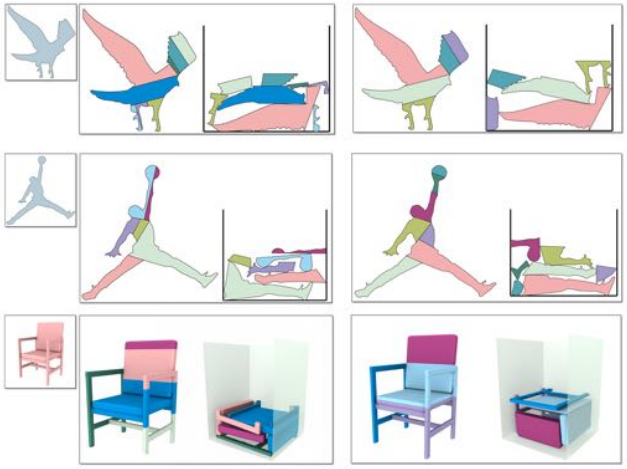


Figure 13: *Contrasting DAP results without (left) vs. with (right) fabrication constraints. As opposed to optimization results without fabrication constraints, decrease in search time and increase in height of the final packing might occur, due to the more stringent search criteria enforced by the constraints. Note the elimination of small cuts (e.g., chair, eagle leg), small parts (eagle leg), and thin structures (many on Jordan, eagle, and top piece of chair).*

tiny parts, and thin structures, e.g., see the Jordan and eagle examples in Figure 8 as well. Figure 13 shows new results obtained for the above three examples with thresholding applied to cut area, part volume, and part thickness. Since the physical properties of different models vary, the threshold values must vary accordingly to allow physically plausible fabrications and assemblies. Also, as the fabrication constraints narrow down the search, the total search time decrease but printing efficiency achieved degrades slightly.

FDM printing. Figure 14 shows several DAP and 3D printing results obtained for the global objective function (3), which is geared towards FDM-based fabrication. The fabrication results were produced by a Fortus 360mc 3D printer. These results can be contrasted to those obtained by minimizing height of the packing to benefit powder-based 3D printing. As expected, the FDM objective generally leads to larger height values for the packing but smaller amount of total vertical gap, as indicated in Figure 14; the latter corresponds to waste material for FDM-based printers.

Figure 14: *DAP results optimized for powder-based vs. FDM-based 3D printing. For each example, we show the input shape, the final (virtual) decompositions and packings with respect to the two optimization objectives, as well as a photograph of the FDM fabrication result. Part counts, heights(cm) and total volume(cm^3) of the vertical gaps are reported as well in vectors to show contrasts.*

Comparison to PackMerger. Like Dapper, the PackMerger algorithm of Vanek et al. [2014] considers both decomposition and packing, and can be toned to optimize for both powder- and FDM-based fabrication. However, the optimization objectives and search paradigms of the two methods differ significantly. We obtained the PackMerger code from the authors and compare Dapper with PackMerger on few models available from Vanek et al. [2014], as well as on models from our test set, while making an effort to evaluate their performances under comparable settings. In particular, we evaluate printing efficiency and execution time (measured on the same machine) for varying part counts, since part count plays rather different roles in the two methods. Unlike Dapper, PackMerger does not integrate part count into its optimization objectives. However, the final greedy merging step is able to produce solutions with different part counts. To allow Dapper to produce results with different part counts, we must adjust the parameter α.

Figures 15 shows several comparisons between Dapper and PackMerger in terms of printing efficiency, for both powder- and FDM-based printing. We can observe that Dapper generally outperforms PackMerger. Note that no matter how we change the parameters, we always obtain the same DAP solution P^* for the table model when optimized for FDM-based 3D printing. The reason is that this DAP solution P^* is already the optimal solution with $G_{FDM}(P^*) = 0$, thus any other solutions will be pruned once we find P^*, according to our DAP search. Analyzing the two methods in terms of their asymptotic time complexity is difficult. Execution times, on the other hand, are highly dependent on implementation choices and details. With the two available implementations, Dapper generally consumes significantly less optimization time compared to PackMerger. For example, the chair model (top row of Figure 15) requires between 1 and 80 seconds for Dapper to optimize for powder-based printing over varying part counts, while for for PackeMerger, the times vary between 55 and 410 seconds. More results are in the supplementary material.

Overall, it would appear that although the Tetris packing in PackMerger covers arbitrary rotations, its performance is inferior to Dapper in both speed and printing efficiencies achieved. Thus the consideration of arbitrary rotations by PackMerger does not seem to compensate for its decoupling of decomposition and packing or the greedy nature of its merging scheme.

Comparison to discrete scissors congruence. To our best knowledge, the only available algorithm which performs a true decompose-and-pack is the one by Zhou et al. [2012] for solving the discrete scissors congruence (DSC) problem. DSC differs from

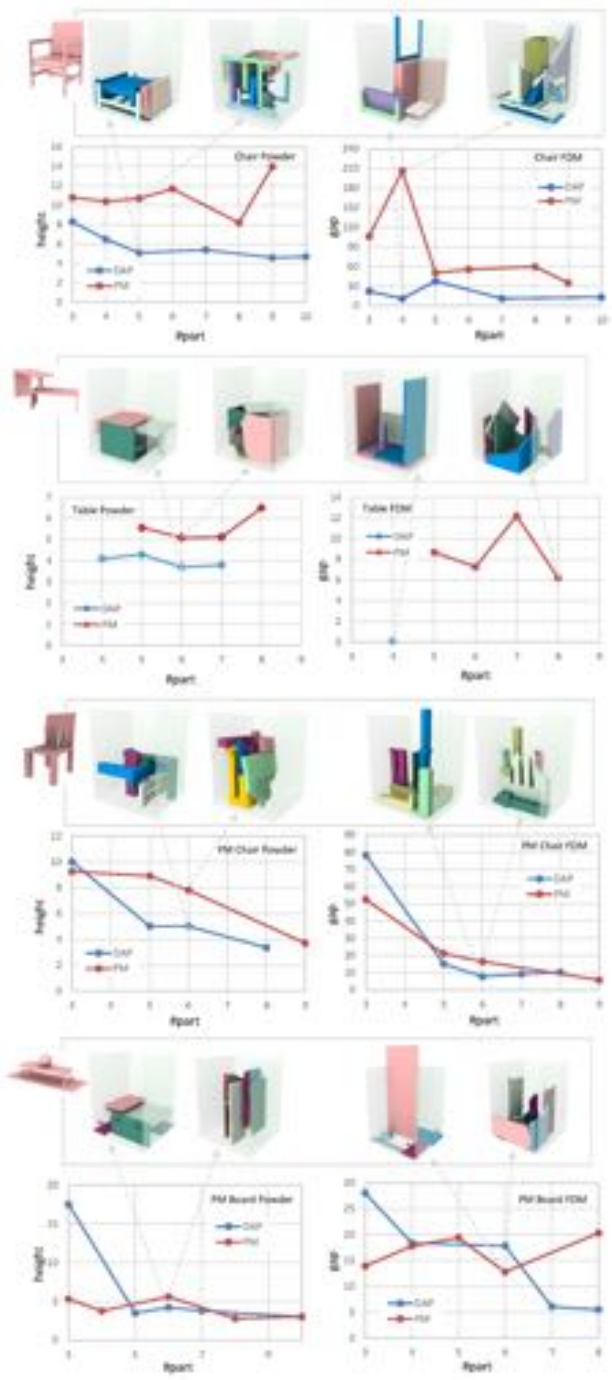


Figure 15: *Comparing Dapper with PackMerger (PM) in terms of printing efficiency (for both powder- and FDM-based printing) over varying part counts, where fabrication constraints are not imposed. The first chair model and the table model are from Figure 9, and the remaining two models were taken from PackMerger.*

DAP both in terms of objectives and constraints, hence it is difficult to fairly compare them. However, it would still be informative to ask whether their DSC solution could be viable for our problem. We obtained the DSC code from Zhou et al. [2012] and ran it on several 2D inputs. For both DAP and DSC, we provide the same voxelized input and the same container width.

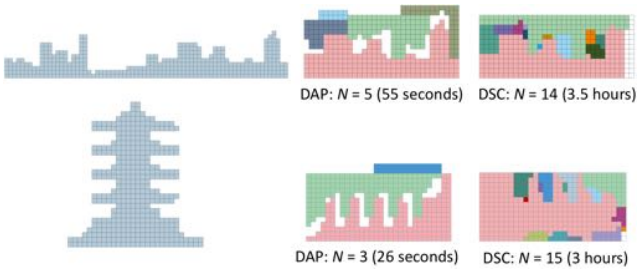


Figure 16: *Dapper vs. scissors congruence (DSC). While Dapper often does not attain the optimal height, it reaches a close enough solution with fewer parts and in a fraction of the time.*

Figure 16 compares the performance of the two methods on two examples, but the trend is representative. In general, DAP does not attain the optimal height; DSC does, since it computes exact congruence, leaving no gap at all. However, DAP typically reaches a solution that is close to optimal in terms of height of the packing, doing so with much fewer parts and in a fraction of the time compared to DSC. DSC takes hours to compute solutions in 2D and much longer on 3D inputs. Hence, it is reasonable to conclude that DSC is not a viable solution to DAP for 3D printing; it likely produces too many parts and takes too long to run.

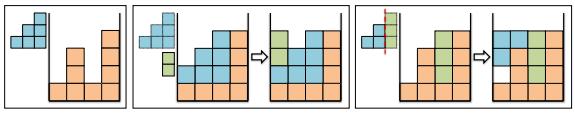
6 Discussion, limitation, and future work

Recent advances in 3D printing technologies have piqued the interests of the computer graphics community, as the desire to improve efficiency and quality of 3D fabrication is shedding new lights on a variety of geometric optimization problems. In this work, we pose the decompose-and-pack (DAP) problem for 3D printing and offer a preliminary solution to one problem instance. Indeed, DAP represents not one, but a new *class* of optimization problems. As the demand for better utilization of printing volume, print material, and build time increase, we believe that effective solutions to DAP can benefit 3D printing in more ways than one. While our coverage has so far focused on two types of printers, other printing technologies such as SLA and DLP share similarities as FDM and powder, respectively, in terms of support requirement and material consumption; they should benefit from DAP in similar ways.

In this section, we offer discussions on design choices and insights, current limitations, and possible extensions for future work.

DAP via conventional clustering. The first question we asked ourselves when developing the DAP solution was whether the problem could be solved by conventional approaches such as clustering. In a typical setting for shape decomposition [Shamir 2008], the goodness of a part is always computable from properties possessed by the part itself or its boundary. Well-known examples include convexity, pyramidality [Hu et al. 2014], and the minima rule [Hoffman and Richards 1984] which offers a means to identify boundaries between parts. However, one can hardly conclude whether a part would facilitate packing by studying the part alone. It would also be difficult to define an affinity measure between parts (to enable a clustering scheme) based on their packability. Packability is a global property inferrable only from a set of parts.

Expanding the search. Our current solution search is limited by the voxelization, which may lead to loose bounds for shape parts and disallow certain compact packing solutions. Figure 11 illustrates this, as well as a limitation of the local refinement step due to its reliance on local movements. Another limitation of Dapper is that the pile can only be piled up and not modified to facilitate pack-



(a) Input. (b) With CAX move. (c) Without CAX.

Figure 17: *Allowing the pile to be partially flattened by making a cut, a* CAX *move, leads to a better DAP solution. Compare results in the middle and right where the same number of parts are obtained. With a* CAX*, we obtain a packing without gap.*

ing. An interesting new DAP move, which we may call CAX for "Cut-and-eXchange", can be added which would allow the pile to be partially flattened by cutting off a piece and exchanging in a new piece. Figure 17 shows that CAX can improve packing. However, CAX moves are expensive to add since they invalidate the objective function bounds for depth pruning; recall that these bounds are based on the fact that the height of the pile cannot decrease. Obviously, we can also expand the search by removing the voxelization and directly decompose-and-pack the shape parts.

Ground truth and optimality. It is less than ideal that for most results shown in the paper, we could not obtain or compare to ground truth. The DAP problem is extremely difficult and its complexity dictates that only for rather trivial examples, one may manually obtain provably optimal results. The 2D tower and bridge examples in Figure 8 are simpler than others and the results we obtain are likely close to optimal. However, even in such simple cases, proving optimality would be hard. Therefore, no optimality claims we make in the paper imply closeness to the ground truth; all such claims are confined to the search space defined and explored.

Pyramidal primitives. While pyramidal primitives do offer several advantages in our solution framework, they are not suitable for all types of input geometry. For example, objects with thin structures, such as the object shells and the ball-and-stick figures featured in PackMerger [Vanek et al. 2014], do not induce small pyramidal decompositions to initialize Dapper. Hence, they are unlikely to result in efficient DAP solutions using our current algorithm.

Decomposition, assembly, and slicing directions. A common issue to all decomposition-based fabrication methods is the *warping* of cut surfaces due to the printing process, which prevents tight assembly of the parts. The work by Hildebrand et al. [2013] is an excellent reminder that choosing the appropriate slicing or cut directions in a decomposition is important. For example, cuts that are not well aligned with the direction of fabrication can be jaggy, especially when printing at a low resolution. Such cuts cannot be glued properly. While pyramidal primitives are highly desirable for DAP, more ideal would be parts that are pyramidal and have cut boundaries aligned with the orthogonal slicing directions derived from [Hildebrand et al. 2013]. Both properties are preserved by voxelization and axial cuts over the resulting polycubes.

Future work. Aside from addressing limitations and possible extensions mentioned above, performance improvements can be expected if we incorporate efficient inexact string matching algorithms into the docking scheme. Additional fabrication constraints, such as those related to stress analysis and aesthetics (e.g., symmetric cuts or cuts over concave regions for less visibility), can also be considered. We would also look into applications of DAP beyond 3D printing, e.g., for furniture disassembly and packing where container dimensions would play a critical role. Finally, it may be interesting to see whether the search paradigm developed in this paper can be adjusted to solve scissors congruence.

Acknowledgments. We thank all the reviewers for their insightful comments and valuable suggestions. We also acknowledge help from Sha He on video editing and early discussion with Ilya Baran on DAP. This work is supported in part by grants from National 973 Program (2015CB352500), NSFC (61232011, 61202147, 61332015), NSERC Canada (No. 611370), Guangdong Science and Technology Program (2015A030312015, 2014B050502009, 2014TX01X033), Shenzhen VisuCA Key Lab (CXB201104220029A), Israeli Science Foundation (No. 1790/12), and U.S.-Israel Bi-National Science Foundation (No. 2012376).

References

AHO, A. V., AND CORASICK, M. J. 1975. Efficient string matching: An aid to bibliographic search. *Communications of the ACM 18*, 6, 333–340.

CAGAN, J., SHIMADA, K., AND YIN, S. 2002. A survey of computational approaches to three-dimensional layout problems. *Computer-Aided Design 34*, 597–611.

CHEN, Y., AND MEDIONI, G. 1992. Object modelling by registration of multiple range images. *Image Vision Comput. 10*, 3, 145–155.

CRAINIC, T. G., PERBOLI, G., AND TADEI, R. 2012. *Recent Advances in Multi-Dimensional Packing Problems*. New Technologies - Trends, Innovations and Research.

DICKINSON, J. K., AND KNOPF, G. K. 2002. Packing subsets of 3D parts for layered manufacturing. *International Journal of Smart Engineering System Design 4*, 3, 147–161.

GOGATE, A. S., AND PANDE, S. S. 2008. Intelligent layout planning for rapid prototyping. *International Journal of Production Research 46*, 20, 560–563.

HILDEBRAND, K., BICKEL, B., AND ALEXA, M. 2013. Orthogonal slicing for additive manufacturing. *Computer & Graphics 37*, 6, 669–675.

HOFFMAN, D. D., AND RICHARDS, W. A. 1984. Parts of recognition. *Cognition 18*, 65–96.

HU, R., LI, H., ZHANG, H., AND COHEN-OR, D. 2014. Approximate pyramidal shape decomposition. *ACM Trans. on Graph 33*, 6, 213:1–213:12.

LOWERRE, B. T. 1976. *The harpy speech recognition system*. PhD thesis, Carnegie Mellon University.

LUO, L., BARAN, I., RUSINKIEWICZ, S., AND MATUSIK, W. 2012. Chopper: Partitioning models into 3D-printable parts. *ACM Trans. on Graph 31*, 6, 129:1–129:9.

SHAMIR, A. 2008. A survey on mesh segmentation techniques. *Computer Graphics Forum 27*, 6, 1539–1556.

VANEK, J., GARCIA, J., BENES, B., MECH, R., CARR, N., STAVA, O., AND MILLER, G. 2014. PackMerger: A 3D print volume optimizer. *Computer Graphics Forum 33*, 6, 322–332.

WIKIPEDIA, 2014. 3D printing — wikipedia, the free encyclopedia. [Online; accessed 6-November-2014].

ZHOU, Y., AND WANG, R. 2012. An algorithm for creating geometric dissection puzzles. In *Proc. of Bridges Conf.*, 49–58.

ZHOU, Y., SUEDA, S., MATUSIK, W., AND SHAMIR, A. 2014. Boxelization: Folding 3D objects into boxes. *ACM Trans. on Graph 33*, 4, 71:1–71:8.