Foldabilizing Furniture

Honghua Li[1,2*] Ruizhen Hu[1,3,4*] Ibraheem Alhashim[1] Hao Zhang[1]
[1]Simon Fraser University [2]National University of Defense Technology [3]SIAT [4]Zhejiang University

Abstract

We introduce the *foldabilization* problem for space-saving furniture design. Namely, given a 3D object representing a piece of furniture, our goal is to apply a minimum amount of modification to the object so that it can be folded to save space — the object is thus foldabilized. We focus on one instance of the problem where folding is with respect to a prescribed folding direction and allowed object modifications include hinge insertion and part shrinking.

We develop an automatic algorithm for foldabilization by formulating and solving a nested optimization problem operating at two granularity levels of the input shape. Specifically, the input shape is first partitioned into a set of integral *folding units*. For each unit, we construct a graph which encodes conflict relations, e.g., collisions, between foldings implied by various patch foldabilizations within the unit. Finding a minimum-cost foldabilization with a conflict-free folding is an instance of the maximum-weight independent set problem. In the outer loop of the optimization, we process the folding units in an optimized ordering where the units are sorted based on estimated foldabilization costs. We show numerous foldabilization results computed at interactive speed and 3D-print physical prototypes of these results to demonstrate manufacturability.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

Keywords: foldabilization, folding, furniture design, shape optimization, shape compaction

> *"Man, himself a collapsible being, physically and psychologically, needs and wants collapsible tools."*
>
> – Per Mollerup (2001)

1 Introduction

Space-saving furniture designs are ubiquitous in our daily lives and workplaces. Effective space saving does not depend on downscaling, but on smart ways of collapsing a piece of furniture or making it more collapsible. Among the many space-saving mechanisms such as stacking, implosion, and bundling [Mollerup 2001], *folding* is perhaps the most frequently observed and best practiced on furniture. Even when confined to furniture, folding can still be executed in a rich variety of ways, offering an abundant source of appealing and challenging geometry problems.

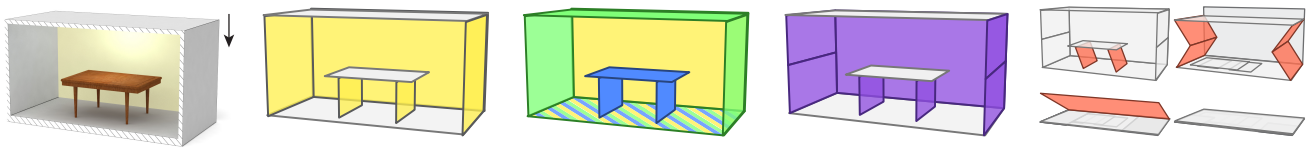
Figure 1: *Two automatic foldabilizations of a chair with respect to two folding directions. Shown on the right are fabrication results produced by a MakerBot Replicator II 3D printer. Folding is possible by adding hinges, shrinking parts (chair back in the top row), or allowing slanting or shearing, leading to less hinges and better structural soundness (bottom right). The foldable chair in the top row resembles the Stitch Chair by the designer Adam Goodrum.*

An interesting geometry question about folding is: what makes some 3D objects more amenable to folding than others? Since rigid parts cannot be folded, hinges need to be inserted to make the parts foldable. Moreover, folding involves constrained movements of object parts and such movements often require necessary clearing space to avoid collision. Hence reducing the size or extent of furniture parts to make space is beneficial to folding, e.g., the back of the chair in Figure 1 (top) is shrunk to allow folding of the seat. Taking the two factors to the extreme, we arrive at structures formed by thin frames with many hinges; famous examples of such objects are scissoring structures such as the Hoberman spheres. However, due to structural strength and functionality considerations, furniture foldabilization can hardly go to that extreme.

In this paper, we pose the novel *foldabilization* problem for space-saving furniture design. Given a 3D object O representing a piece of furniture, our goal is to apply a *minimum* amount of modification to O to allow its parts to be folded *flatly* onto themselves or each other. We focus on one particular instance of the foldabilization problem where folding is allowed only with respect to a prescribed *folding direction* and the allowed modifications include adding (*line*) *hinges* onto furniture parts and *shrinking* the parts. Figure 1 shows two automatic foldabilization results for a chair, and the resulting folding sequences on fabricated 3D prototypes.

Foldabilization is not an easy task for humans. It resembles a 3D puzzle with a large search space and a multitude of constraints. Solving the problem requires delicate spatial reasoning and a keen foresight to adapt to the dynamic changes to the shape configuration as folding sequences proceed. While humans are highly apt at pattern recognition, they are not as skilled at precise 3D manipulation while relying solely on visual guidance. In particular, in human perception, lengths in 3D are systematically distorted due to perspective viewing [Baird 1970; Norman et al. 1996]. Thus, by

*Honghua Li and Ruizhen Hu are joint first authors.
e-mail: howard.hhli@gmail.com, ruizhen.hu@gmail.com



(a) Input 3D object. (b) Patch scaffold. (c) Folding units. (d) Modifications. (e) Unit ordering and folding.

Figure 2: *Key components of foldabilization. Given an input 3D object (a) with a folding direction (arrow), we first abstract the object using a patch scaffold (b) consisting of base (white) and foldable patches (yellow). The scaffold is then decomposed into a set of folding units (c); here three units are obtained (blue, green, and yellow) and they share the bottom patch. Each unit is foldabilized to allow a folding; this step may introduce patch modifications including shrinkage and hinge insertion (d), as well as patch disconnection and slanting (e). Note that the back panel of the wall is identified as a patch where no hinge insertion is allowed, possibly due to structural soundness concerns. The algorithm automatically determines an order for folding the units: blue → green → yellow, and the folding options selected, to minimize a global cost. Some key frames for the resulting folding are shown in (e).*

only relying on manual effort, we can expect foldabilization to be a tedious and error-prone process.

In our work, we develop an automatic algorithm for foldabilization, aimed at assisting designers in creating foldable furniture and quick exploration of many design alternatives. The technical challenges are two-fold. Conceptually, formulating 3D foldabilization into a tractable optimization problem is not straightforward. Perhaps the most challenging aspect of the problem is the *dynamic* nature of the folding process: as it proceeds, the shape configurations and constraints to foldabilization change over time. It is difficult to formulate a constrained optimization with a dynamically changing constraint set. Computationally, we face an expensive search problem. The problem of finding a collision-free folding while allowing shape modifications is an instance of the maximum independent set problem (MISP), which is NP-hard, and this is only a sub-problem of foldabilization. Furthermore, adding the time dimension to the search also increases the problem complexity.

Given a 3D shape, we first convert it into a *patch scaffold*, or simply, a scaffold, representation. The scaffold consists of a connected set of planar and *zero-thickness* surface patches, serving as a shape abstraction for folding analysis. While the zero-thickness assumption is unrealistic, the algorithmic framework for foldabilization is the same with or without patch thicknesses. We make the assumption to simplify presentation of the algorithm and describe later how to reintroduce path thickness as additional constraints in the algorithm to produce manufacturable results. Starting with a scaffold and a given folding direction $\vec{d}$, we identify a set of *base* patches which shall remain rigid and onto which the scaffold can be folded. The remaining patches are *foldable* patches: they can be folded onto the bases, possibly after hinge insertion or patch shrinking, each with its associated cost. The foldabilization problem seeks the minimal-cost modification to the input scaffold which allows it to be folded along $\vec{d}$ into a flat structure. Figure 2 shows an overview of our foldabilization algorithm.

To find an optimal foldabilization, we formulate and solve a *nested* optimization problem which operates at two granularity levels of the input scaffold. Specifically, we partition the scaffold into a set of *folding units* each of which consists of a connected subset of patches whose foldings influence each other. In the outer loop, we find an optimized order in which the folding units are folded, while in the inner loop, each unit is foldabilized and then folded. Breaking down the optimization this way makes the search problem more tractable. The units are folded *independently time-wise* but *dependently space-wise*. In other words, the algorithm folds the units one after another while the choice of which unit to fold first influences how much space is left to fold subsequent units.

For each unit, we construct a *conflict graph* which encodes conflict relations (mainly collisions) between foldings implied by various patch foldabilizations within the unit. Each patch foldabilization is associated with a cost. Finding a conflict-free folding turns out to be an instance of MISP, while finding a *minimum-cost* folding is an instance of the maximum-weight independent set problem (MWISP); both problems are NP-hard. In the outer loop of the optimization, we optimize to find an order in which to process the folding units — the units are sorted based on estimated foldabilization costs.

The zero thickness assumption and the simplified patch modification model revealed so far allow us to maintain a certain purity of the foldabilization problem so that the algorithm can first focus on the hardness of the optimization and folding dynamics. However, practical issues such as manufacturability and structural soundness must also be addressed to provide a realistic solution to foldabilizing furniture. It is fairly straightforward to extend our algorithm to incorporate patch thickness, patch disconnection, and modification constraints into the optimization framework. We simply need to modify the set of space constraints, e.g., when patches have assigned thicknesses, and to enrich the set of object modifications, e.g., to allow separating patches and to disallow hinge insertion on a part which may bear heavy weight.

Figure 1 and Section 6 demonstrate numerous results for furniture foldabilization and folding, obtained in seconds. We show that our optimization framework works well with a variety of folding options and design constraints. We also 3D-print physical prototypes of foldabilized designs to demonstrate manufacturability. An interesting implication of our work is that the prototypes can be printed while folded to save material and print time.

2 Related work

In recent years, a steady stream of geometry problems motivated by design and manufacturing applications have emerged. These problems arise from architectural design [Pottmann et al. 2007], 3D printing [Luo et al. 2012] and prototyping of mechanical objects including furniture [Koo et al. 2014].

Furniture design and fabrication. Lau et al. [2011] decompose a furniture object into fabricatable pieces and determine proper placement of connectors for assembly. Umetani et al. [2012] develop an interactive system to assist exploration and design of geometrically and physically valid plank-based furniture. A growing body of recent works [Saul et al. 2011; Schmidt and Ratto 2013; Schulz et al. 2014; Koo et al. 2014] propose data- or user-driven approaches to design fabricatable furniture. The key motivation for most of these works is to automate the design process and reduce human effort. Our work is in the same spirit from an application perspective. Our core technical contribution is the introduction and

solution to foldabilization, a challenging geometry problem. Recent work of Koo et al. [2014] can also produce folding solutions while allowing part scaling. However, in their work, the user is required to directly specify folding (part) pairs, and only within-pair collision resolution is considered. In contrast, our algorithm automatically foldabilizes an entire object and resolves collisions between multiple folding pairs and units.

Space-saving designs. Efforts invested into the design of collapsible tools and furniture can be traced back thousands of years. The inspiring book by Mollerup [2001] exhibits twelve key space-saving principles and hundreds of examples drawn from our daily lives. Well-known geometry problems related to space-saving designs include cutting+packing [Lodi et al. 2002; Bennell and Oliveira 2008] and folding. Li et al. [2012] pose and solve the stackabilization problem which seeks minimal modifications to a 3D object so that it can be compactly stacked together. While stackabilization and foldabilization share similar goals and both are applicable to space-saving furniture design, the geometry problem and the optimization are completely different.

Geometric folding problems. More closely related to our work are obviously geometric folding problems. There is an amazing variety of folding problems, e.g., see [Demaine and O'Rourke 2007], and most of them are highly challenging in their general setting, hence various simplifying assumptions have been made to allow tractable solutions to be developed. We cover a sampler of folding problems below, where a few of them share some commonality to our foldabilization problem. However, the problems are all different in one way or another, in terms of objectives or constraints, leading to varied problem formulations and solutions. The key difference between folding and foldabilization is that the latter must optimize with respect to both folding and shape modification.

Linkage folding and boxelization. A linkage consists of a set of rigid line segments connected by universal joints to form a graph. The fundamental question studied in linkage folding is how to transform one folding configuration into another where folding only occurs at the given joints and the linkage does not self-cross during the folding [Demaine and O'Rourke 2007]. In a recent work called boxelization, Zhou et al. [2014] converts a voxelized 3D object into a tree linkage structure that can be folded into a box form. This problem is extremely difficult as it has a packing element to it. Foldabilization has a different goal. In terms of folding analysis, their task is to determine where and how a pre-determined set of voxels approximating the input shape should be connected, while we minimize the cost of a variety of patch modifications, allowing the input to be geometrically altered. Also, we find a conflict-free folding sequence by solving a constrained geometric optimization while boxelization employs physical simulation.

Paper folding. Paper folding [Demaine and O'Rourke 2007; Jackson 2011], e.g., origami [McArthur and Lang 2013], is perhaps the most popular form of folding. Generally speaking, paper folding, in particular, origami, allows folding and sculpting of a piece of paper to form a 2D or 3D shape. In most cases, the folds are straight but 3D shape creation via curved foldings has also been studied [Kilian et al. 2008]. Origami designs can lead to fascinating 3D shapes that are more free-form than the kinds of furniture objects we deal with in this paper. The design objective in this domain is a difficult inverse problem, seeking a crease pattern on a piece of paper and a folding sequence which results in a target shape.

Pop-up design. Li et al. [2010] design pop-up paper architectures where both folding and cutting, which resembles patch shrinkage in our work, are allowed. However, their problem is grounded on a geometric formulation of planar layout. Later, Li et al. [2011] study v-style pop-up designs where the base patches belong to four parallel groups. They derive sufficient conditions for a v-style paper structure to be pop-uppable. Recent work by Ruiz Jr. et al. [2014] allows more pop-up styles and converts a given 3D shape into a multi-style pop-up design. Their goal is to approximate the 3D shape using base and foldable patches which lead to a pop-up design. In contrast to these paper folding problems, including pop-up designs, foldabilization is a form of design optimization, which finds ways of modifying an existing 3D shape to make it foldable.

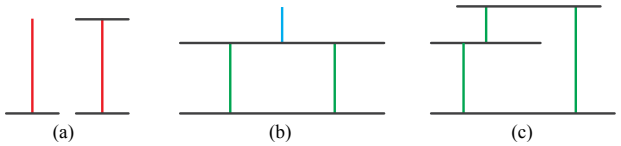


Figure 3: *Basic scaffolds and folding units. (a) T-scaffold (left) and H-scaffold (right) with foldable patches in red. (b) The T-scaffold on the top forms a unit (blue) by itself, and the two H-scaffolds at the bottom form another unit (green). (c) Three H-scaffolds form a single unit. Multiple basic scaffolds form a unit since their foldings influence each other directly (b) or indirectly (c).*

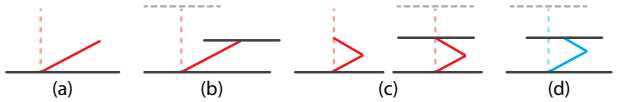


Figure 4: *Folding mechanisms. (a, b) Slanted foldings of a T- and H-scaffold. (c) In-place foldings of the same scaffolds, requiring hinge insertion. In all of these cases, movements of the top of the foldable patch are predictable by the scaffold types. (d) Hybrids between slanted and in-place foldings are not considered.*

3 Overview

The input to our algorithm is a pre-segmented 3D object O and a folding direction $\vec{d}$. The output is a foldabilized shape O' with an optimized set of modifications to allow O' to be folded with respect to $\vec{d}$. All object parts are piecewise rigid and abstracted using cuboids which are tight OBBs that bound the actual part geometry. Any part that is folded must rest flatly onto itself or another part. All hinges to realize the folding are perpendicular to $\vec{d}$.

In this section, we provide an overview of our baseline foldabilization algorithm which operates on a *scaffold abstraction* $\mathcal{I}$ of the input object O; see Figure 2. The scaffold $\mathcal{I}$ is a connected set of *planar rectangular* patches with zero thickness. All *base patches* are perpendicular to the folding direction $\vec{d}$ and during folding, they only undergo translation and maintain perpendicularity to $\vec{d}$. The remaining patches are *foldable patches* which, after foldabilization, can possibly be folded onto the base patches.

Basic scaffold and unit. The minimal scaffold for our foldabilization analysis, called a *basic scaffold*, consists of one foldable patch together with either one or two base patches. A basic scaffold with two bases is called an *H-scaffold*; if one of the bases degenerates, we have a *T-scaffold*; see Figure 3(a). Between basic scaffolds and the entire input scaffold $\mathcal{I}$, we establish a set of mid-level scaffolds called *folding units*. The folding units allow us to model and execute the search for an optimal foldabilization of $\mathcal{I}$ as a nested optimization problem. Each folding unit consists of a set of basic scaffolds whose foldings influence each other directly or indirectly; see Figure 3 for an illustration and Section 5.2 for more details.

Folding mechanisms. To enable folding, hinges have to be *enabled* at joints between base and foldable patches to allow *slanting*, or they are *inserted* in the interior of an otherwise rigid patch to make the patch foldable. Figures 4(a-c) show the four types of folding mechanisms allowed in our problem formulation, where a folding is *in-place* if the topmoat tip or patch only moves perpendicularly to the base patch. For any of these cases, movements of the (topmost) base patches are predictable by scaffold types, facilitating collision analysis during folding. On the other hand, we do not allow a hybrid between slanting and in-place foldings, where a slanted patch also has inserted hinges, as shown in Figure 4(d), since it is somewhat uncommon in practice and complicates the search.

Folding configuration, transform, and timing. Given any scaffold $\mathcal{S}$, e.g., a basic scaffold or a unit, a *folding configuration* expresses a *state* of folding for $\mathcal{S}$, which consists of the angles between patches at all the hinges in $\mathcal{S}$. A *folding transform* is a *timed* folding *"event"*, and specifically, a continuous sequence of folding configurations for $\mathcal{S}$. Such an event is associated with a *folding time interval* defined by the start and end times of the folding transform. All folding events are executed in uniform speed, hence the time interval for each basic scaffold in a unit is dictated by the positions of its base patches. More detailed coverage is in Section 4.

Folding solutions. A *folding solution* for $\mathcal{S}$ stores information about how the patches in $\mathcal{S}$ are modified and a folding transform that is obtainable by folding the modified $\mathcal{S}$. Our goal is to find an optimal folding solution for the input scaffold $\mathcal{I}$, which is obtained by *progressively* accumulating folding solutions from basic scaffolds and then from the folding units. For space saving considerations, we provide an option to constrain the folding of $\mathcal{I}$ to be confined to its tight bounding box volume.

Unit foldabilization. To foldabilize a given folding unit, we establish a conflict graph. Each node of the graph stores a folding solution for a basic scaffold in the unit; there may be multiple folding solutions (nodes) per basic scaffold. There is an edge between two nodes if their corresponding folding transforms induce a conflict, such as a collision between patches during folding. We find a set of patch modifications with minimum cost which would lead to a flat folding of the unit by solving an MWISP [Östergård 2001].

Unit ordering. We assume that the folding time intervals of the units do not overlap. However, to determine an optimal ordering of the units so as to achieve the minimum total foldabilization costs remains difficult. To this end, we resort to a *cost-driven* greedy scheme with one-step lookahead to order the folding units. At each step, a unit U which incurs the least total cost for foldabilizing itself and the remaining (unfolded) units is chosen, where we simulate the folding of a remaining unit only with respect to the configuration with U folded. Section 5.5 provides more details.

Units are folded one by one this way, until no remaining unit can be folded conflict-free. Finally, we combine all the remaining units into a single sub-scaffold and foldabilize it as a unit.

4 Definitions and problem statement

The notion of a *scaffold* as a shape abstraction has appeared in previous works such as [Li et al. 2011] in a setting similar to ours. In our work, we denote a generic scaffold by $\mathcal{S}$, which is composed of a set $P_{\mathcal{S}}$ of rigid planar rectangular patches connected by a set $H_{\mathcal{S}}$ of (line) hinges. We distinguish the input scaffold by denoting it by $\mathcal{I}$ and its patch and hinge sets by $P_{\mathcal{I}}$ and $H_{\mathcal{I}}$, respectively.

We characterize the structure of a scaffold by a *hinge graph*:

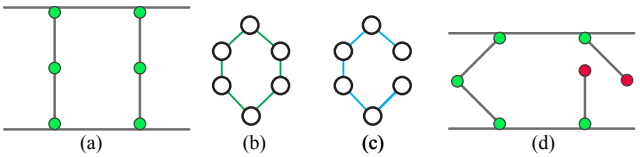


Figure 5: *Valid vs. invalid folding configurations. (a) A scaffold with a valid configuration. (b) The hinge graph for (a). (c) A spanning tree of (b). (d) An invalid folding configuration which breaks the red hinge when using only angles of the green hinges to determine patch positions.*

Definition 1. *A **hinge graph** $G_{\mathcal{S}}^{hinge}$ for scaffold $\mathcal{S}$ has nodes that are the patches in $\mathcal{S}$. There is an edge between two nodes if there is a hinge connecting the two corresponding patches.*

Definition 2. *A **folding configuration** $\Theta_{\mathcal{S}} = (\theta_1, \theta_2, \ldots, \theta_n) \in \mathbb{R}^n$ for a scaffold $\mathcal{S}$ is a hinge angle assignment, where the angle of hinge $h_k \in H_{\mathcal{S}}$ is given by θ_k.*

Due to the dynamic and sequential nature of folding, we need to assign timestamps to folding configurations. A folding configuration that emerges at time t is denoted by Θ^t.

Definition 3. *A **folding transform** on scaffold $\mathcal{S}$ is $f(t_i^0, t_i^1) = \Theta_{\mathcal{S}}^t$, which represents a continuous sequence of folding configurations of $\mathcal{S}$ defined over the folding time interval $t \in [t^0, t^1]$.*

Validity of folding configuration and transform. Given a folding configuration Θ, if there exists a collision-free arrangement of patches such that all the assigned hinge angles are satisfied, then we say that Θ is a *valid* configuration. A folding transform is valid if all of its corresponding folding configurations are valid.

To verify validity of a folding configuration Θ, we utilize the hinge graph. After finding a spanning tree of the hinge graph G^{hinge} and fixing the position of one patch, the positions of other patches in the scaffold can be uniquely determined according to the angles assigned to the edges in the spanning tree. If the angles of the hinge edges not included in the spanning tree are also satisfied and no patches collide with each other, then Θ is valid.

Refer to Figure 5 for some illustrations. Figure 5(a) shows a simple scaffold. The hinge graph (b) and spanning tree (c) are constructed to verify the validity of the folding configuration in (a). An example of an invalid folding configuration is shown in (d). Using the angle assignment only to the edges of the spanning tree breaks the red hinge and thus the angle assignment is not valid.

Foldable scaffold. A folding configuration is flat if all of its patches are co-planar. A scaffold $\mathcal{S}$ is said to be foldable if, starting with its current configuration (i.e., the hinge angles), it can be folded into a flat configuration via a valid folding transform.

Definition 4. *A **folding solution** $F_{\mathcal{S}} = \{m, f\}$ for a scaffold $\mathcal{S}$ is a pair where m is a set of patch modifications applied on $\mathcal{S}$ and f is a valid folding transform starting at the initial configuration of $m(\mathcal{S})$ and ending at a flat configuration, where $m(\mathcal{S})$ is the scaffold after applying the set of modifications.*

Allowed modifications. In our baseline algorithm, two types of patch modifications are allowed: a *split* of a foldable patch introduces a new hinge that is required to be perpendicular to the folding direction $\vec{d}$; a *shrinking* modification scales a patch down along the hinge direction on that patch; see Figure 1 for some examples.

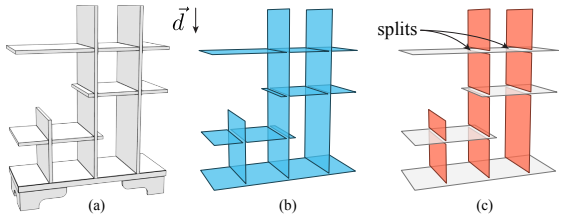


Figure 6: *Scaffold abstraction. (a) A segmented input 3D shape. (b) Each segment is abstracted by a rectangular patch. (c) Scaffold obtained by fitting base patches (gray) perpendicular to $\vec{d}$ and splitting foldable patches (orange) with the base patches.*

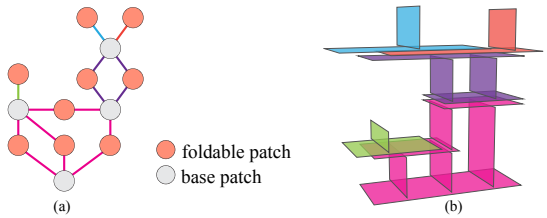


Figure 7: *Example of unit decomposition via hinge graph. (a) Hinge graph of the scaffold in Figure 6(c). Edges belonging to different units are shown in different colors. (b) Unit decomposition result with corresponding colors.*

Foldabilization. We define a cost for a folding solution $F_{\mathcal{S}}$ which is used to guide the optimization. The cost is based on the costs of the associated patch modification m_p:

$$cost(F_{\mathcal{S}}) = \sum_{p \in P_{\mathcal{S}}} \lambda_p cost(m_p), \tag{1}$$

where λ_p is the (normalized) importance of patch p. We formally define the foldabilization problem for the input scaffold $\mathcal{I}$ as,

$$F^{\star}_{\mathcal{I}} = \arg \min_{F_{\mathcal{I}} \in \mathscr{F}_{\mathcal{I}}} cost(F_{\mathcal{I}}), \tag{2}$$

where $\mathscr{F}_{\mathcal{I}}$ is the set of all possible folding solutions for scaffold $\mathcal{I}$ and $F^{\star}_{\mathcal{I}}$ is the optimal solution to the problem.

5 Algorithm

In this section, we first present in detail the baseline algorithm which foldabilizes a scaffold $\mathcal{I}$ with zero-thickness patches; see Section 5.1 on scaffold construction. The scaffold is decomposed into two levels of sub-scaffolds: basic scaffolds and units (Section 5.2). We combine the folding solutions for basic scaffolds (Section 5.3) to those of units (Section 5.4) and finally to those of the input scaffold (Section 5.5). We describe how to incorporate patch thickness and structural soundness constraints into the baseline algorithm in Sections 5.6 and 5.7, respectively.

5.1 Scaffold abstraction

We assume that the input shape has been meaningfully segmented. Segmentation of furniture objects is relatively straightforward using state-of-the-art algorithms. We approximate each segment of the input shape using a rectangular patch by OBB part fitting and medial sheet extraction. The patches are connected first by distance thresholding to ensure that the resulting scaffold $\mathcal{I}$ is connected.

To prepare for folding, we further classify patches into two types: base and foldable patches. During a folding process, base patches remain perpendicular to the folding direction $\vec{d}$, while foldable patches are folded onto base patches, possibly after inserting hinges. Figure 6 shows a 3D object and its scaffold abstraction.

Base patches. We first collect all the patches and patch edges that are perpendicular to $\vec{d}$ as a set of candidate fitting components. Then for each set of *connected* candidate components that are co-planar with the shared plane perpendicular to $\vec{d}$, we fit a rectangular patch as a base patch. A base patch obtained as described above is discarded if the corresponding connected co-planar set consists of one single patch edge; this is a degenerate case with the base patch correponding to the "tip" of a T-scaffold.

Foldable patches. After identifying base patches, we split the remaining patches that intersect with base patches to make sure they only contact base patches at their ends. Then we fit the remaining patches by rectangular patches so that each patch has one or two ends connecting to base patches; these are the foldable patches. Note that there are no connections between foldable patches. All (line) joints between a base and a foldable patch are possible hinges and can potentially enable slanted folding.

5.2 Decomposition into folding units

In this section, we describe how to decompose a scaffold into a set of sub-scaffolds, and the conditions for integrating folding solutions from sub-scaffolds to the whole scaffold. We also motivate the use of mid-level scaffolds, called folding units, in our optimization framework and present the decomposition algorithm.

Decomposition. A *decomposition* of the scaffold $\mathcal{S} = (P, H)$ is a set of sub-scaffolds $\{\mathcal{S}_i = (P_i, H_i)\}$, where $\{H_i\}$ is a partition of H and P_i is a subset of patches connected by hinges in H_i. Accordingly, $\mathcal{S}$ is said to be a combination of the $\mathcal{S}_i$'s.

Suppose that the folding solution for each sub-scaffold $\mathcal{S}_i$ is $F_i = (m_i, f_i)$, where f_i is the folding transform applied on the modified sub-scaffold $m_i(\mathcal{S}_i)$. At timestamp t, the sub-scaffold $\mathcal{S}_i$ has folding configuration $\Theta^t_{\mathcal{S}_i}$ according to the folding transform $f_i(t^0_i, t^1_i)$. In case the timestamp exceeds the scope of folding time interval, we define $\Theta^t_{\mathcal{S}} = \Theta^{t^0}_{\mathcal{S}}$ if $t < t^0$ and $\Theta^t_{\mathcal{S}} = \Theta^{t^1}_{\mathcal{S}}$ if $t > t^1$.

In order to integrate folding solutions F_i's into a folding solution $F = (m, f)$ for the whole scaffold $\mathcal{S}$, there are three necessary conditions that have to be satisfied:

(i) Patches shared between two sub-scaffolds must have the same modification.

(ii) All hinge angles in the hinge graph of the combination of $\Theta^t_{\mathcal{S}_i}$'s must be satisfied; see Figure 5 for an illustration.

(iii) There is no patch collision in the combination of $\Theta^t_{\mathcal{S}_i}$'s.

In regards to Condition (i), we simply assume that all base patches remain unaltered during foldabilization and only foldable patches can be modified. It follows that the set of modifications to the $\mathcal{S}_i$'s are non-overlapping and the cost of F is simply a sum of costs from the individual sub-scaffold modifications. Conditions (ii) and (iii) together state that the combination of $\mathcal{S}_i$'s must be a valid folding configuration at any timestamp.

Folding units. Decomposing the input scaffold $\mathcal{I}$ into a set of basic scaffolds is one possible decomposition and it appears to be straightforward, however combining their folding solutions remains problematic, especially since Condition (ii) is not easy to satisfy.

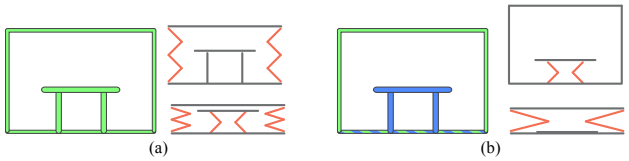


Figure 8: *Benefit of unit decomposition. (a) A scaffold is treated as a single unit. In this solution, the table would remain static until the ceiling patch touches it and forces it to move. More hinges are needed on the wall patches to avoid collision with the table. (b) The scaffold is decomposed into two units (blue and green). The table is folded first, thus fewer hinges are needed on the walls.*

Recall that not all folding configurations are valid (Figure 5). The angle assignment for those hinges, whose corresponding edges in the hinge graph are not selected in the spanning tree, may not be valid. In our work, we identify a type of mid-level sub-scaffolds, which we call folding units, so that hinges that influence each other are considered together to guarantee that Condition (ii) is satisfied.

Unit decomposition. We utilize the hinge graph $G_{\mathcal{I}}^{hinge}$ for scaffold $\mathcal{I}$, as defined in Section 4, for unit decomposition. We find all the simple cycles and cluster edges in all cycles which share hinge edges. For hinge edges that are not contained in cycles, we merge edges in each basic scaffold as a cluster. This way we obtain an edge partition of the hinge graph. Each set of partitioned edges with the connected patch nodes corresponds to a sub-scaffold of $\mathcal{I}$, which becomes the folding unit, or simply a unit, that we seek. Figure 7 shows an example of unit decomposition, where the bottom unit obtained, shown in red color in Figure 7(b), resembles the example shown in Figure 3(c). Figure 8 shows a 2D example where decomposition into units helps us find a better solution than if the whole scaffold is treated as a single unit.

5.3 Folding solutions for basic scaffolds

In this section, we describe all possible folding solutions for a basic scaffold in isolation. These basic folding solutions are employed for foldabilizing units in Section 5.4.

A folding solution for a basic scaffold is a combination of a set of patch modifications and a valid folding transform. The modifications on the foldable patch of the basic scaffold are derived from our assumption on the movement of the base patches (see Figure 4), and result in a valid folding transform for the modified basic scaffold. For convenience, in the following coverage, we assume that the folding direction $\vec{d}$ is vertical and pointing downward. Accordingly, the foldable patch in the basic scaffold is *not* horizontal.

Split and folding mechanisms. A T-scaffold can be folded onto the base patch after adding hinges onto its foldable patch. We always insert hinges at *equal intervals*, leading to a zig-zag folding pattern as shown in Figure 9(a). An H-scaffold has two base patches: one is designated as the upper patch and the other as the lower patch. While keeping the lower patch fixed, the upper patch should translate to its lower counterpart and remain perpendicular to the folding direction $\vec{d}$. The upper patch can translate along any trajectory, which requires careful positioning of hinges. In our work, we consider two common folding options in practice:

- *In-place folding*: This option forces the upper base to translate along $\vec{d}$ *straight down* to the lower base. This is desirable for space saving, as it incurs no horizontal space expansion. Again, all the hinges are inserted from one end of the foldable patch at equal intervals. When the foldable patch is parallel to $\vec{d}$, it is easy to compute hinge locations. However, if the foldable patch is in a slanted position to start with, as shown in Figure 9(b), the computation is more involved and we refer the reader to the supplementary material for details.

- *Slanted folding*: In this option, no hinges are added to the foldable patch, but a hinge is enabled between the base and foldable patch. When folding occurs, the upper base translates such that the foldable patch swings onto the lower base patch, as shown in Figures 4(a) and (b), where the former shows slanted folding of a T-scaffold.

  Since the foldable patch is not split, this type of folding incurs *zero* modification cost; it may also be desirable in terms of maintaining structural strength (of the foldable patch). However, slanting incurs possible horizontal space expansion. In our solution search, slanting is implemented as an option. For T-scaffolds though, slanting is turned on by default.

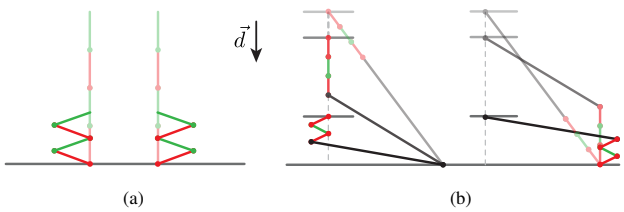


Figure 9: *Split foldable patches and folding mechanisms. (a) Three splits on a T-scaffold. (b) In-place folding with three splits on an H-scaffold. Folding to different sides leads to different splits.*

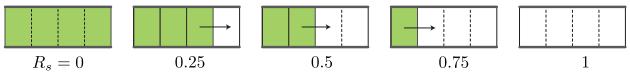


Figure 10: *Patch shrinking with discretized ratios. The foldable patch is cut into $N_s = 4$ pieces. A shrunk patch only takes a portion shown as green windows, which can slide to different locations.*

We set the maximum number of new hinges per patch as a tunable parameter $\hat{N}_h$. A hinge number $N_h < \hat{N}_h$ corresponds to two folding solutions, one to the right and the other to the left, see Figure 9. We denote a split modification $m^{split} = (N_h, s)$, where $s \in \{l, r\}$ indicates to which side the lowest part of the foldable patch folds. Note that only an *odd* number of hinges on the foldable patch can imply in-place foldings in our setting.

Patch shrinking. A foldable patch can shrink only along a direction parallel to the hinge direction on that patch; both the shrinking and hinge directions are perpendicular to the folding direction $\vec{d}$. Note that patch shrinking does not influence the folding mechanisms; it however changes the space occupied by folding the foldable patch, which may benefit other basic scaffolds when multiple basic scaffolds fold together, e.g., in a unit; see Section 5.4 for further details.

Along the hinge direction, we uniformly cut the foldable patch into N_s pieces, where N_s is another tunable parameter. The shrinking ratio R_s is discretized such that the shrunk patch is bounded by two cutting points a_0 and a_1, where $[a_0, a_1] \subseteq [0, 1]$. We denote a shrinking modification by $m^{shrink} = (a_0, a_1)$, where $R_s = 1 - (a_1 - a_0)$. Figure 10 demonstrates all possible shrunk patches with $N_s = 4$. Given a shrinking factor N_s, we can shrink the foldable patch in $n_2 = N_s(1+N_s)/2$ ways, as well as delete it when setting $R_s = 1$. However, patch deletion is only allowed if it does not cause disconnection of the input scaffold.

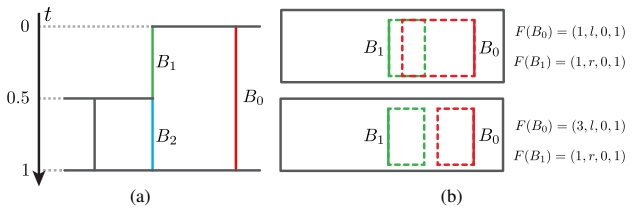


Figure 11: *Folding time intervals and collision relationships between in-place folding solutions. (a) Folding time interval for each basic scaffold inside a unit (side view). (b) Collision relationships between basic folding solutions (top view). The folding time intervals for B_0 (red) and B_1 (green) overlap, thus they collide if their folding regions (dashed boxes) intersect. The top two folding solutions collide while the bottom two do not.*

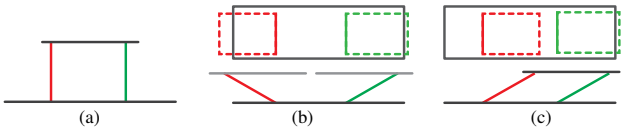


Figure 12: *Conflict relationships between slanted folding solutions. (a) A unit consisting of two H-scaffolds sharing an upper patch (side view). The two slanted folding solutions in (b) conflict since they drag the upper base in different directions, while the two slanted foldings in (c) do not. The top rows in (b) and (c) show the corresponding folding regions (dashed boxes) from the top view.*

Cost of basic folding solution. Since a basic scaffold B contains one unique foldable patch p, we define its folding solution cost based on the patch modification $m_p = (N_h, s, a_0, a_1)$,

$$cost(F_B) = cost(m_p) = \alpha N_h / \hat{N}_h + (1 - \alpha) R_s^2, \tag{3}$$

where $\alpha \in [0, 1]$ is a preference weight parameter between two types of patch modifications.

5.4 Unit foldabilization

Each unit U can be decomposed into a set of basic scaffolds $\{B_i\}, i = 1, \ldots, N$, thus a folding solution F_U for U can be represented by a combination of folding solutions $\{F^*_{B_i}\}$ for B_i's with corresponding folding time interval $[t_i^0, t_i^1]$:

$$F_U = \{(F^*_{B_i}, t_i^0, t_i^1)\}. \tag{4}$$

We assume that each unit U is folded in a way that the folding time interval for each B_i is dictated by the positions of its base patches. As well, we need to find a folding solution $F^*_{B_i}$ for each B_i with the given folding time interval so that the combined folding solution produces a valid folding transform for U. With these criteria, we are able to formulate the unit foldabilization problem as an instance of MWISP (Maximum Weight Independent Set Problem).

Folding time intervals for basic scaffolds. Suppose that the time interval for folding a unit is $[0, 1]$. Without loss of generality, we divide this time interval based on the position of base patches. Figure 11(a) shows how we map the base patches to the time axis. The higher base patches always start translating before the lower base patches do. The folding time interval for an H-scaffold is determined by the timestamps of its two base patches; while a T-scaffold has $[t_i^0, t_i^1] = [0, 1]$ since it forms a unit by itself.

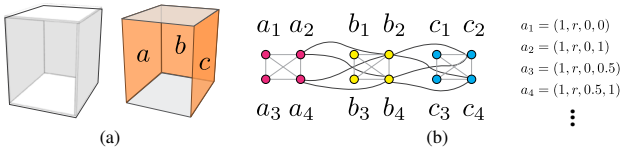


Figure 13: *Conflict graph. (a) A 3D shape and its corresponding scaffold consisting of a single unit. (b) Conflict graph with corresponding folding solutions. $\{a_3, b_3, c_3\}$ is a maximum independent set corresponding to a folding solution for the unit.*

Conflict between folding solutions. For each basic scaffold, we have designed folding solutions that always produce valid folding transforms, and assigned the folding time interval based on its base patches. Now we can show that finding a folding solution for U boils down to finding a folding solution $F^*_{B_i}$ for each basic scaffold such that $\{F^*_{B_i}\}$ do not conflict each other. Two folding solutions *conflict* if they together cannot generate a valid folding solution for the unit, violating Conditions (ii) or (iii) in Section 5.2.

Note that we enforce the base patches to translate along the folding direction $\vec{d}$. Based on our split and folding mechanisms, the hinge angles are guaranteed to be satisfied (Condition (ii)), and two fold solutions conflict only if they cause a patch collision (Condition (iii)). To simplify collision detection, we define the *folding region* for each basic folding solution as the rectangular region occupied by the foldable patch when fully folded. Two folding solutions $F^*_{B_i}$ and $F^*_{B_j}$ *collide* if their folding time intervals overlap: $(t_i^0, t_i^1) \cap (t_j^0, t_j^1) \neq \emptyset$, and their corresponding folding regions intersect: $R_i \cap R_j \neq \emptyset$, see Figure 11(b). Note that by definition, two folding solutions for the same basic scaffold always collide.

In the case where all basic scaffolds in a unit share the same pair of base patches, we also allow slanted folding solutions for basic scaffolds. To meet Condition (ii), any two basic folding solutions must drag the upper base patch through the same trajectory; this can be verified by checking the final folded position of the upper base. Figure 12 shows two different situations for slanted folding.

Conflict graph. We construct a *conflict graph* to capture the conflict relationships among all possible folding solutions for basic scaffolds with their folding time intervals; see Figure 13.

Definition 5. *Given a unit U and the basic scaffolds $\{B_i\}$ it contains, the **conflict graph** is an undirected graph $G^{conf} = \{V, E\}$, where $V = \{v_j\} = \cup_i \mathscr{F}_{B_i}$ and $E = \{e_{j,k} | v_j$ and v_k conflict$\}$.*

We can show that a folding solution F_U for unit U corresponds to a maximum independent set (MIS) on the graph G^{conf}. First, we prove that $|\text{MIS}| = N$ by noting: 1) All folding solutions for a basic scaffold form a clique, thus $|\text{MIS}| \leq N$; and 2) each basic scaffold has a folding solution of deleting the foldable patch, which does not conflict with solutions from other basic scaffolds, thus $|\text{MIS}| \geq N$. Next, we prove that $F_U \Leftrightarrow$ MIS by noting: 1) Since $F_U = \{F^*_{B_i}\}_{1,\ldots,N}$ is an independent set with size N, $F_U \Rightarrow$ MIS; and 2) From the proof for $|\text{MIS}| = N$, we know that MIS must include one and only one folding solution per basic scaffold, hence MIS $\Rightarrow F_U$.

MWISP formulation. Our goal is to find a unit folding solution F_U, namely an MIS, with minimum cost. To this end, we define $cost(v_j) = \lambda_i cost(F_{B_i})$, where v_j corresponds to F_{B_i}, and λ_i is the normalized foldable patch area indicating the importance of B_i. We formulate the problem as an instance of MWISP.

The input to MWISP is an undirected graph with weights on its nodes. The output is an independent set of nodes with maximum total weight. To convert our problem to MWISP, we assign node

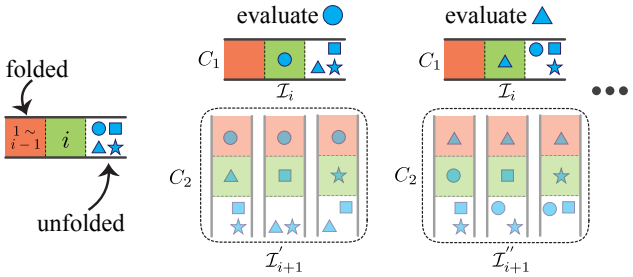


Figure 14: *Unit ordering via one-step lookahead. At the i-th step, we select a unit which leads to the least total cost $C_1 + C_2$ among all unfolded units. In the figure, each shaped mark, e.g., a star or rectangle, represents a unit. The set of folded units is colored in orange, and the set of unfolded units in white, which together constitute the shape configuration $\mathcal{I}_i$ or $\mathcal{I}'_i$. Units in the green blocks are foldabilized given the shape configurations. Both C_1 and C_2 are computed on those units in the green blocks.*

weights as $w(v_j) = \max_v cost(v) - cost(v_j) + 1$. Note that the output $\mathcal{M}$ of the MWISP is an MIS, thus a unit folding solution. Otherwise there is $\mathcal{M} \subset \mathcal{M}'$, which is an MIS with larger total weight because $w(v_j) > 0$. We use the C source code of Cliquer published in [Niskanen and Östergård 2003] to solve MWISP.

5.5 Unit ordering

As the input scaffold $\mathcal{I}$ is decomposed into a set of folding units $\{U_i\}$, we can represent a folding solution $F_{\mathcal{I}}$ as a combination of folding solutions $\{F^*_{U_i}\}$ for the units U_i's, with corresponding folding time interval $[t_i^0, t_i^1]$:

$$F_{\mathcal{I}} = \{(F^*_{U_i}, t_i^0, t_i^1)\}. \tag{5}$$

Since the unit decomposition has already resolved all the loops in the hinge graph, Condition (ii) from Section 5.2 is always satisfied. To avoid patch collision (Condition (iii)), we fold the units one by one in an order such that each unit is foldabilized with respect to the current configuration that is composed of all other units, either folded or unfolded. Suppose that the folding order is indicated by the subscript i in Equation 5, then $t_i^1 = t_{i+1}^0$. Our optimization runs in two nested loops. In the inner loop, we ensure that the folding of a unit does not cause collisions with all other units, in their folded or unfolded configurations. In the outer loop, we find an optimal order for folding the units.

Inner loop. The folding solution for each unit U is confined by the free space left by the current folding configuration of the entire shape. In the context of all other units, either folded or unfolded, the folding of U must not introduce patch collisions with other units. In the conflict graph, we prune basic folding solutions that collide with other units. By solving the MWISP on the remaining conflict graph, the best folding solution can be obtained.

Outer loop. The order in which the units are folded influences the available free space a particular unit can use, and in turn, the available space dictates the amount of patch modification possible for folding that unit. The optimization at the outer loops aims to find an optimal folding order that leads to the least amount of modifications overall. This is a difficult global optimization problem. To this end, we insist on a *cost-driven* model to order the units but resort to a greedy scheme with one-step lookahead for efficiency.

Suppose that up to now there are $(i-1)$ units that are already folded and they define a folding configuration of the input scaffold, denoted by $\mathcal{I}_i$. Pick one remaining unfolded unit U. Denote by C_1 the cost of optimally foldabilizing U given the current scaffold $\mathcal{I}_i$ and the resulting updated scaffold (after folding U) by $\mathcal{I}'_{i+1}$. Denote by C_2 the sum of costs associated with foldabilizing each of the remaining units given the scaffold $\mathcal{I}'_{i+1}$. The greedy choice, with one-step lookahead, for the i-th unit to fold is defined to be the unit, among the unfolded units up to now, which minimizes $C_1 + C_2$. Figure 14 provides a schematic diagram to explain this step. In case no remaining unit can be folded under the current context, all remaining units are merged into one unit and in-place folding is enforced to guarantee the existence of a folding solution.

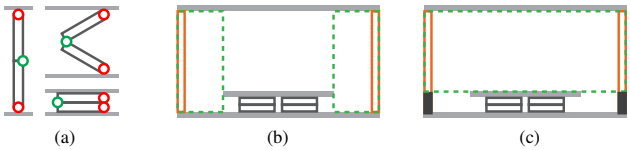


Figure 15: *Incorporating patch thickness. (a) Cylindrical hinges (side view) whose radii are half the patch thickness enable folding with thick patches. (b) With patch thickness, the folded unit (black) serves to add space constraints to the unfolded unit (orange). The orange unit must be foldabilized with respect to the available folding space shown in dashed green boxes, either in (b) or in (c).*

5.6 Patch thickness

Our baseline foldabilization algorithm described so far operates on a scaffold with zero-thickness patches. In reality, furniture parts do have thickness and folding computations depend on thickness. Thickness influences the foldabilization algorithm in two aspects: hinge design and available folding space.

Designing printable hinges for folding has been studied before, e.g., see [Zhou et al. 2014]. Figure 15(a) illustrates the hinges we use in this work. Given the thickness K of a foldable patch, we replace line hinges with cylindrical hinges with radius $r = K/2$. The hinge between the foldable patch and its base is placed at the end of the foldable patch, just making contact with the base; see hinge markers in red. Hinges introduced by splitting a foldable patch are placed on the surface, shown as green markers. Note that these two types of hinges have different rotating angle spans.

The algorithmic aspect of incorporating thickness into our foldabilization framework simply involves adding additional space constraints as thick patches occupy space even when folded. As a unit with patch thickness is folded, we use the space the folded unit occupies to crop off available space for folding subsequent units. We make the design decision that the cropping is formed by cuts that are aligned with the AABB of the folded unit. Figures 15(b) and (c) illustrate these situations, in 2D space. In (b), vertical cropping leaves two dashed green boxes into which the orange unit can be folded. In (c), horizontal cropping essentially raises the base patch, which forces the lower (gray) portion of the orange unit to be taken out of consideration during subsequent foldabilization.

5.7 Structural soundness

Our main mechanism for improving the structural soundness of a foldabilization solution is to reduce hinging on patches. Slanted folding is one option as it prevents a patch from having hinges inserted in its interior. Another option is to explicitly disallow a patch to be foldable. The determination of which parts need to remain rigid in a piece of furniture involves structural analyses of physical

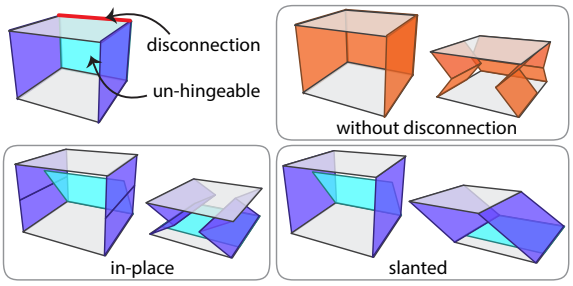


Figure 16: *Effects of enforcing structural soundness constraints. In the box model, the back panel was identified as un-hingeable. Our algorithm automatically computes an in-place foldabilization solution involving part disconnections and two interior hinges. If in-place folding is not enforced, a slanting solution is chosen by the algorithm where no interior hinges are inserted. For an in-place folding where all patches are hingeable, three interior hinges and patch shrinking are needed.*

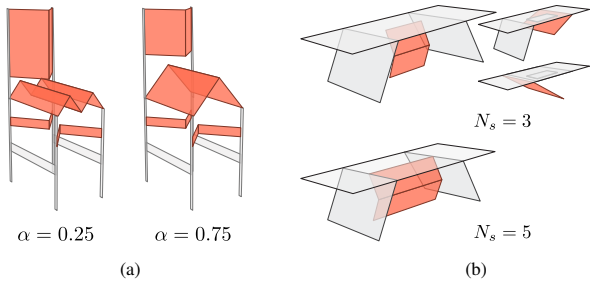


Figure 17: *Influence of parameters. (a) By penalizing splits less ($\alpha = 0.25$ vs. $\alpha = 0.75$), the foldabilized chair incurs less shrinkage on the back but an extra hinge. (b) By increasing the discretization level for patch shrinking ($N_s = 5$ vs. $N_s = 3$), the search space is enlarged, allowing a lower-cost solution to be found.*

objects and is beyond the scope of this paper. We simply let users mark such parts as input to our foldabilization algorithm.

For each un-hingeable patch marked, we prune all basic folding solutions that split it. Thus we only allow slanted folding of the patch in a basic scaffold. In case the basic scaffold is an H-scaffold and its foldable patch cannot be slanted, we disconnect the top end of the H-scaffold and turn it into a T-scaffold with only slanted folding allowed. Such a disconnection prevents hinging in the interior of the foldable patch. Like slanting, disconnections do not incur costs during foldabilization.

Figure 16 shows a few examples of enforcing structural soundness by patch disconnection and slanting. In contrast, disallowing these two folding options would necessarily lead to extra hinges and part shrinking. All solutions are computed automatically, only with the user marking which part may be un-hingeable. Note that if the back panel (shown in cyan color) were not marked as un-hingeable, it would need to be deleted to obtain a slanted solution.

6 Results

In this section, we show foldabilization results, with physical fabrication, and evaluate our algorithm. Experiments are conducted on both synthetic furniture, as well as 3D shapes obtained from existing repositories or modeled from objects captured in on-line images or photographs. In total, we have tested our algorithm on 36 objects representing different types of furniture.

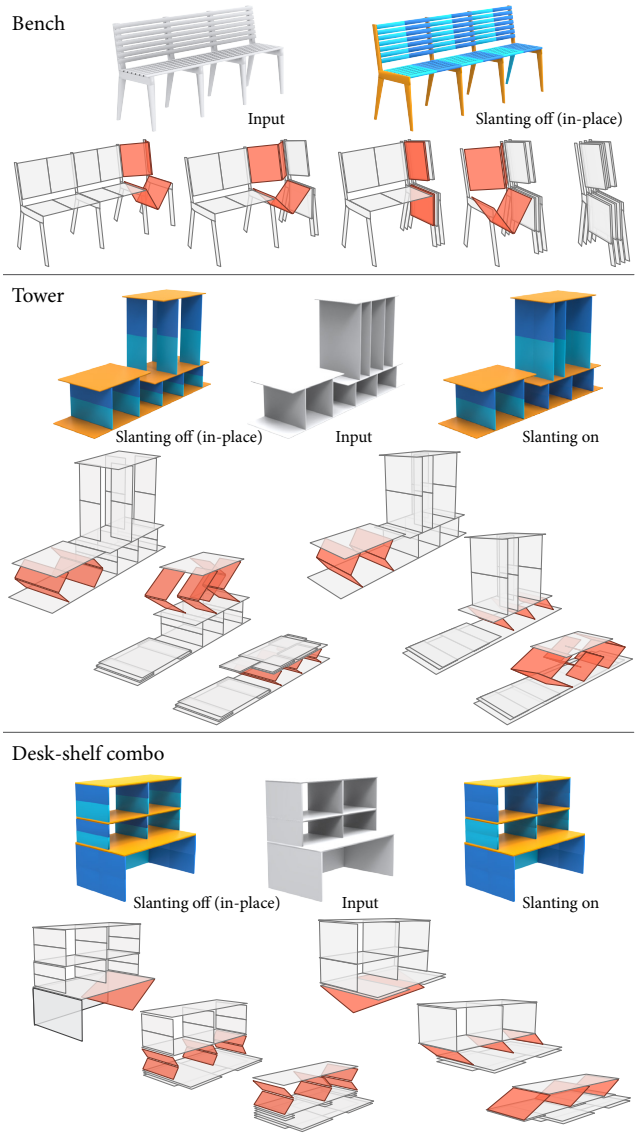


Figure 18: *A gallery of foldabilization results and folding sequences of the foldabilized furniture, leading to final flat configurations. Foldable patches in the folding sequence are indicated in orange color. Newly added hinges are shown as boundary lines between blue and cyan sub-patches.*

Our foldabilization algorithm is quite efficient. Most results (aside from those in a specially designed scalability test) can be obtained instantly using our tool: 100-300 ms on an Intel(R) Xeon(R) 2.40GHz with 16GB RAM). This allows the tool to be used to rapidly explore many foldabilization and folding options.

Parameters. Our algorithm has three tunable parameters: the preference parameter α, which controls the trade-off between the two types of modifications: splitting (adding hinges) and shrinking patches; $\hat{N}_h$, the maximum number of hinges that can be added to a foldable patch; and N_s, the shrinking factor. The last two parameters define levels of discretization for our search and serve to control the size of the search space. All the results shown in the paper have been obtained with the default parameter setting: $\alpha = 0.5$, $\hat{N}_h = 3$, and $N_s = 5$. Figure 17 shows two simple examples of the influence of changing parameters.

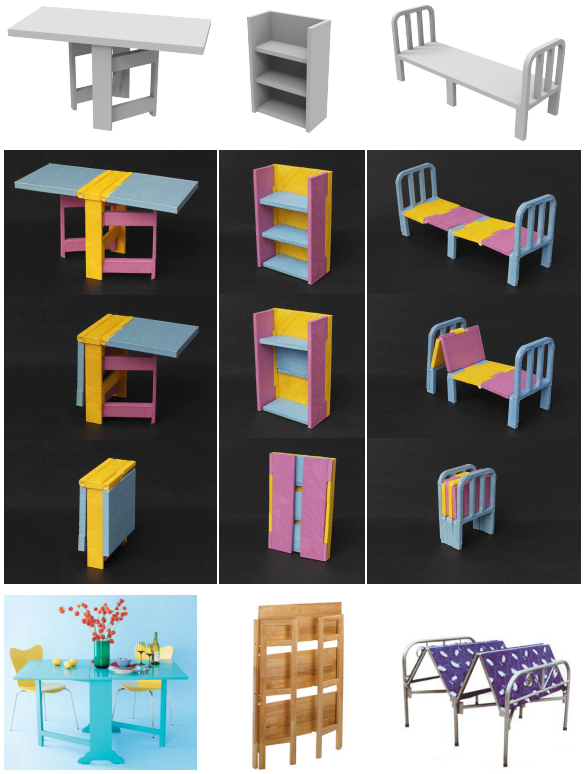


Figure 19: *Photographs showing fabricated prototypes of three foldabilized furniture designs. The space saving ratios are* 88.9%*,* 78.9%*, and* 75%*, respectively. Foldable patches, with varying thickness, are painted in color. The 3D printer used was a MakerBot Replicator II. Last row shows real-world foldable furniture designs that match our solutions.*

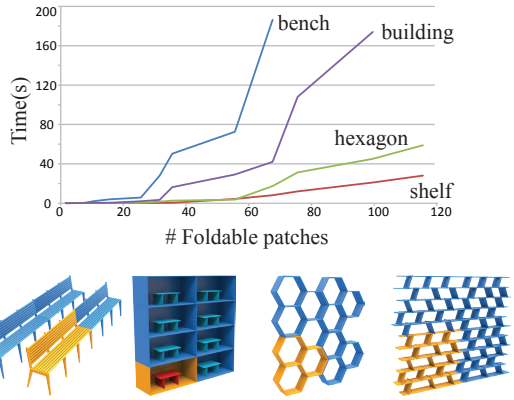


Figure 20: *Scalability of our foldabilization algorithm tested on several synthesized inputs with increasing complexity.*

Model \ Grid	1 × 1		2 × 2		3 × 3		4 × 4	
	u	p	u	p	u	p	u	p
Bench	3	6	12	24	27	54	48	96
Building	2	5	6	18	12	39	20	68
Hexagon	1	10	1	36	1	78	1	136
Shelf	5	15	10	60	15	135	20	240

Table 1: *The shape complexity of synthesized input for scalability test: the number of units (u) and foldable patches (p).*

Foldabilization and folding results. Figure 18 shows a gallery of foldabilization results and folding sequences of the resulting scaffolds leading to flat, zero-thickness, configurations. The folding sequences displayed reveal the folding units obtained by our decomposition and the order in which the units are folded, with foldable patches indicated in orange color. Newly added hinges show up as boundary lines of colored regions over foldable patches.

To provide more insight and exhibit the versatility of folding characteristics our tool can afford, we show results both with slanting turned on and off (see annotations in Figure 18). While slanting is a good option for structural soundness (due to fewer hinges), it may lead to space expansion along directions perpendicular to the folding direction. As we can see, folding of the top shelves in the desk-shelf combo (Figure 18, bottom) extends beyond the horizontal extent of the input object. With slanting on, but by constraining the folding of the tower furniture (Figure 18, middle) to be within the bounding box of the input, an option in our search, in-place folding is chosen for the top shelf panels; only the bottom panels are slanted. Turning off slanting ensures that the foldings produced are in-place, resulting in better space saving overall. Since slanting is always on for T-scaffolds, the bottom of the desk-shelf combo is still slanted to reach a lower cost.

Observe the automatic shrinking of patches, as shown with the in-place folding of the top shelf panels in the tower object when slanting is turned off. In this case, each such panel was shrunk to about half the original width and the four panels became non-overlapping when folded left-to-right. Finally, compare the foldabilization of the bench object in the top row of Figure 18 to that of the chair in Figure 1. Without the bars between the legs to conflict with the folding of the seat of the bench, it can be folded downward so as to not cause a shrinkage on the back; these are all automatically computed by our algorithm.

Fabrication. We 3D-print prototypes of several foldablized furniture designs, using an FDM-based MakerBot Replicator II printer. Figure 19 shows photographs of the fabrication results. Note the varying thicknesses of the different furniture parts, as the result of our foldabilization algorithm. The Replicator II uses PLA material, requiring removal of the waste by hand. We print the furniture parts one by one and then assemble them, using a few hinge types we have designed to realize the several folding options. Note that the reported space saving ratio has been measured using volumes of the tightest bounding box of the input shape and the folded shape.

Scalability. To scale up the problem size, we arrange a varying number of available furniture objects into larger and larger grids and apply our algorithm on the composite models. This way, we are able to test the robustness as well as performance of our algorithm on larger input models. Figure 20 (top) plots the foldabilization time against the total number of patches in the input scaffolds. Some of the sample composite models are shown in Figure 20 (bottom), with the original furniture object shown in yellow.

Our algorithm succeeded in foldabilizing all the composites. However, the execution times scaled differently, which can be attributed to the variation in the number of units and the size of the collision graph within each unit. Bench and building composites are more expensive to process since they possess a larger number of units and our greedy ordering of units has a quadratic time complexity. Note that the number of foldable patches within a unit does not necessarily increase the complexity of the collision graph. The reason is that a sparsely connected graph usually is composed of a set of components and this can lead to an efficient MWISP solution.

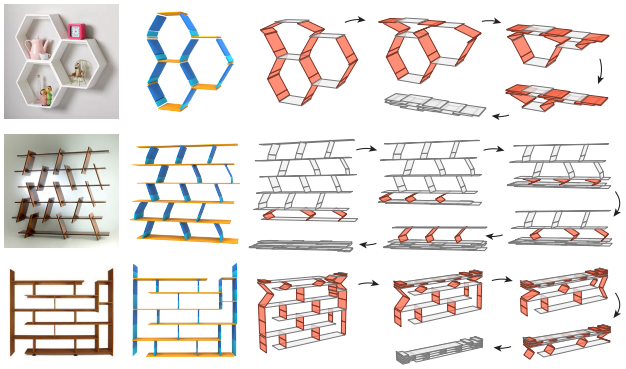


Figure 21: *In-place foldabilization of a few more complex and non-orthogonal bookshelves. The 3D objects were modeled as inspired by images (left column) returned from a Google search on "shelf".*

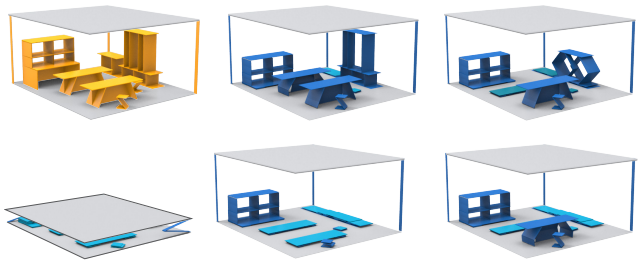


Figure 22: *"We can fold up your whole room!"*

Figure 21 shows a few additional results on more complex shelves, with slanting turned off to enforce in-place folding. It is worth noting that new hinges are not always inserted in the middle; their locations are geometry-dependent, e.g., see the hexagonal shelf. The 3D objects in the figure were modeled from images or photos (left column) returned by a Google image search for "shelf". These objects were selected to demonstrate our algorithm's ability to deal with *non-orthogonal* and more complex patch structures.

At last, we show a fun example demonstrating that our algorithm can scale up to foldabilize a room of furnitures; see Figure 22.

7 Discussion, limitation, and future work

In this paper, we pose and solve one particular instance of the foldabilization problem and apply our method to foldabilize 3D objects representing various kinds of furniture, including chairs, tables, and various forms of shelf and plank structures. A designer can utilize our tool to quickly find out the space-saving potential of a furniture design. Multiple folding directions can be tested to seek the best. Discoveries can made about effective folding orders/sequences, the least costly way of altering an existing design to achieve maximum space saving, as well as ratings of multiple designs in terms of space saving. The applicability of our method is certainly not restricted to furniture; it is suitable for any 3D object for which a patch scaffold abstraction is appropriate, e.g., certain types of trays and other containers or compartments, portable shacks, among others.

On the technical front, we have made several assumptions to arrive at the current problem instance. These assumptions include a meaningful pre-segmentation of the input shape, a given folding direction, planar rectangular patch in the scaffold abstraction, and restricting base patches and hinge directions to be perpendicular to the folding direction. Despite of these assumptions, the particular instance of the foldabilization problem we have posed and studied still has strong appeals from a geometric analysis and optimization standpoint and it remains quite challenging to formulate and solve. From a practical point of view, the results produced by our work allow a variety of space-saving design options to be realized; these include hinging, slanting, part disconnection, and patch shrinking. We take measures, e.g. slanting over hinging, to address the structural soundness issue. With varying thicknesses possible for the furniture parts, the designs are manufacturable and we show physical fabrications of design prototypes produced by our algorithm.

Applicability. To get a rough idea of the extent of applicability of our foldabilization technique in processing "interesting" furniture objects, we performed Google image search on a few object queries and examined the top returns to select the ones that qualify as "interesting". We judged individually whether each captured furniture object can be effectively abstracted using a patch scaffold and then foldabilized using our technique in a meaningful way to save space. We repeated this exercise for five object queries and examined the top 100 returns. The percentage of "foldabilizable" objects among the 100 returns are as follows: "desks" = 35%; "bookshelf" = 71%; "table" = 65%; "bench" = 63%; and finally "chair" = 38%. It is important to note that our assessment on applicability is subjective.

That said, our folding analysis and foldabilization algorithm only represent a preliminary attempt. There exists a variety of furniture folding mechanisms which are not addressed by our formulation. Figure 23 shows only a small sampler of such designs, which reflect the fundamental limitation of our approach arising from the assumption that objects are folded with respect to a single direction.

Printing folded configuration. With a furniture piece folded and keeping a low profile, there is the potential of fabricating the piece while it is folded instead of printing its parts one by one. A low profile is desirable for powder-based printing while a compact configuration is preferable by FDM printing as the support (thus waste) material is expected to be small. In the latter case, an FDM printer with dissolvable support material is more likely to succeed since with folding, we expect to have thin layers of support material residing in the interior of the folded configuration. Figure 24 shows photographs of the two teaser examples in their unfolded configurations; they were fabricated in the folded form.

Problem decomposition. An important step in our method is the decomposition of the input into a number of folding units. In retrospect, we coarsify the time scale by sorting only the units and not all the folding transforms arising from all possible patch modifications. Currently, we have not yet found a way to integrate patch modification, collision relation, and folding time all into a *one-shot* constrained optimization problem. Decomposition into folding units and solving a nested optimization is an approach that decouples the factors, making the problem more tractable. At the same time, our method is not guaranteed to find the least-cost foldabilization.

Structural strength of design. The physical strength of a piece of furniture is expected to be reduced by adding new hinges and

Figure 23: *Some types of furniture folding that we do not address.*

Figure 24: *Photographs showing two chairs in their unfolded configurations (left); each of them is fabricated in the folded state (right) using an FDM printer with dissolvable support material.*

thinning of furniture parts. We currently offer a few options, e.g., slanting (over hinging) and disallowing hinges on certain parts, but these certainly do not address all structural issues. General physical and structural analysis is beyond the scope of our work, while other works in graphics had gone into that direction, e.g., for building design and 3D printing. Modeling of functionality and usage scenarios are also possible extensions in these types of analyses.

Future work. There are many ways to expand our foldabilization framework. Incorporating timing into within-unit collision detection is a short-term task, as well as incorporating functional or aesthetic constraints such as symmetry into the optimization. Next, it would be interesting to allow different folding directions to be applied to different parts of an object. A practical yet unexplored instance of foldabilization is to constrain the folded structure to be within a given container, which resembles the boxelization problem to some extent. Finally, we would like to incorporate the options of sliding parts along hinges and allowing parts to be inserted into other (hollowed) parts, such as for drawers.

Acknowledgements

We thank the anonymous reviewers for their valuable comments and suggestions. We are grateful to Lu Lin for printing the models in Figure 24. Thanks also go to Daniel Cohen-Or, Tao Ju, and Ariel Shamir for some initial discussions. This work is supported in part by grants from Natural Sciences and Engineering Research Council of Canada (No. 611370 and No. 611649), GRAND NCE, NSFC (61232011), National 973 Program (2014CB360503), and Shenzhen Key Lab (CXB201104220029A).

References

BAIRD, J. C. 1970. *Psychophysical analysis of visual space*. New York: Pergamon Press.

BENNELL, J. A., AND OLIVEIRA, J. F. 2008. The geometry of nesting problems: A tutorial. *European Journal of Operational Research 184*, 2, 397–415.

DEMAINE, E. D., AND O'ROURKE, J. 2007. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press.

JACKSON, P. 2011. *Folding Techniques for Designers: From Sheet to Form*. Laurence King Publishing.

KILIAN, M., FLÖRY, S., CHEN, Z., MITRA, N. J., SHEFFER, A., AND POTTMANN, H. 2008. Curved folding. *ACM Trans. on Graph 27*, 3, 75:1–75:9.

KOO, B., LI, W., YAO, J., AGRAWALA, M., AND MITRA, N. J. 2014. Creating works-like prototypes of mechanical objects. *ACM Trans. on Graph 33*, 6 (Nov.), 217:1–217:9.

LAU, M., OHGAWARA, A., MITANI, J., AND IGARASHI, T. 2011. Converting 3D furniture models to fabricatable parts and connectors. *ACM Trans. on Graph 30*, 4, 85:1–85:6.

LI, X.-Y., SHEN, C.-H., HUANG, S.-S., JU, T., AND HU, S.-M. 2010. Popup: automatic paper architectures from 3d models. *ACM Transactions on Graphics 29*, 4, 111:1–9.

LI, X.-Y., JU, T., GU, Y., AND HU, S.-M. 2011. A geometric study of v-style pop-ups: Theories and algorithms. *ACM Transactions on Graphics 30*, 4, 98:1–10.

LI, H., ALHASHIM, I., ZHANG, H., SHAMIR, A., AND COHEN-OR, D. 2012. Stackabilization. *ACM Trans. on Graph 31*, 6, 158:1–158:9.

LODI, A., MARTELLO, S., AND MONACI, M. 2002. Two-dimensional packing problems: A survey. *European Journal of Operational Research 141*, 2, 241–252.

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.

MCARTHUR, M., AND LANG, R. J. 2013. *Folding Paper: The Infinite Possibilities of Origami*. Turtle Publishing.

MOLLERUP, P. 2001. *Collapsible: The genius of space-saving design*. Chronicle, San Francisco, Calif.

NISKANEN, S., AND ÖSTERGÅRD, P. R. J. 2003. Cliquer user's guide, version 1.0. Tech. rep., Helsinki University of Technology, Espoo, Finland.

NORMAN, J. F., TODD, J. T., PEROTTI, V. J., AND TITTLE, J. S. 1996. The visual perception of three-dimensional length. *Journal of Experimental Psychology: Human Perception and Performance 22*, 1, 173–186.

ÖSTERGÅRD, P. R. J. 2001. A new algorithm for the maximum-weight clique problem. *Nordic J. of Computing 8*, 4, 424–436.

POTTMANN, H., ASPERL, A., HOFER, M., AND KILIAN, A. 2007. *Architectural Geometry*. Bentley Institute Press.

RUIZ JR., C. R., LE, S. N., YU, J., AND LOW, K.-L. 2014. Multi-style paper pop-up designs from 3d models. *Computer Graphics Forum (Special Issue of Eurographics)*.

SAUL, G., LAU, M., MITANI, J., AND IGARASHI, T. 2011. Sketchchair: An all-in-one chair design system for end users. In *Proc. of Int. Conf. on Tangible, Embedded, and Embodied Interaction*, 73–80.

SCHMIDT, R., AND RATTO, M. 2013. Design-to-fabricate: Maker hardware requires maker software. *IEEE Computer Graphics and Applications 33*, 6, 26–34.

SCHULZ, A., SHAMIR, A., LEVIN, D., SITTHI-AMORN, P., AND MATUSIK, W. 2014. Design and fabrication by example. *ACM Trans. on Graph 33*, 4, to appear.

UMETANI, N., IGARASHI, T., AND MITRA, N. J. 2012. Guided exploration of physically valid shapes for furniture design. *ACM Trans. on Graph 31*, 4, 86:1–86:11.

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