autoAssembler: *Automatic* Reconstruction of Laser-Cut 3D Models

THIJS ROUMEN, CONRAD LEMPERT, INGO APEL, ERIK BRENDEL, MARKUS BRAND, LAURENZ SEIDEL, LUKAS RAMBOLD, AND PATRICK BAUDISCH

Hasso Plattner institute

University of Potsdam, Germany

Firstname.lastname@hpi.de

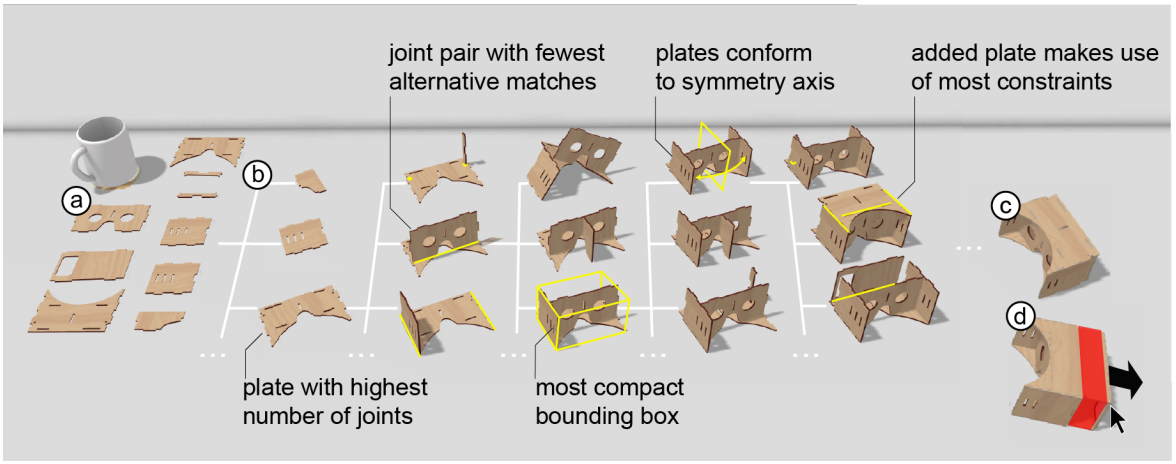

Figure 1: *AutoAssembler* converts 2D cutting plans to 3D models by (a) importing 2D cutting plans and (b) beam-searching the space of ways to assemble the plates. *AutoAssembler* prefers candidates that (1) have no intersecting plates, (2) fit into a small bounding box, (3) use plates whose joints fit together well, (4) do not add many unpaired joints, (5) make use of constraints posed by other plates, and (6) conform to symmetry axes of the plates. (c) This allows users to load the model into a 3D editor (*kyub* [4]), (d) where they can now apply parametric changes.

1 ABSTRACT

Recent research showed how to import laser cut 3D models encoded in the form of 2D cutting plans into a 3D editor (*assembler*^{3} [28]), which allows users to perform parametric manipulations on such models. In contrast to *assembler*^{3}, which requires users to perform this process *manually*, we present *autoAssembler*, which performs this process *automatically*. AutoAssembler uses a beam search algorithm to search possible ways of assembling plates. It uses joints on these plates to combine them into assembly candidates. It thereby preferably pursues candidates (1) that have no intersecting plates, (2) that fit into a small bounding box, (3) that use plates whose joints fit together well, (4) that do not add many unpaired joints, (5) that make use of constraints posed by other plates, and (6) that conform to symmetry axes of the plates. This works for models that have at least one edge joint (finger or t-joint). In our technical evaluation, we imported 66 models using autoAssembler. AutoAssembler assembled 79% of those models fully automatically; another 18% of models required on average 2.7 clicks of post-processing, for an overall success rate of 97%.

CCS CONCEPTS • Human-centered computing~Human computer interaction (HCI)**;**

Additional Keywords and Phrases: laser cutting, personal fabrication, reuse, remixing

ACM Reference Format:

Thijs Roumen, Conrad Lempert, Ingo Apel, Erik Brendel, Markus Brand, Laurenz Seidel, Lukas Rambold, and Patrick Baudisch. 2021. AutoAssembler: Automatic Reconstruction of Laser-Cut 3D Models. Proceedings of the 34th Annual ACM Symposium on User Interface Software and Technology. Association for Computing Machinery, New York, NY, USA, DOI:https://doi.org/10.1145/3411764.3445453

2 INTRODUCTION

Models for laser cutting are hard to produce, they typically go through iterations of refinement and testing. Unsurprisingly, there is a strong community of enthusiasts who share such models [39]. Researchers studying such communities identified that remixing [2] and customizing [14] models has been a great driver for the field, in fact, even a key enabler to transition towards ubiquitous personal fabrication [3] as suggested by Stemasov et al., [35]. In the context of laser cutting, typical modifications would be changing the material thickness and stretching the dimensions of the models aka "parametric changes".

The most interesting laser cut models, consist of multiple plates that connect to one another using cutouts on the edges of the plates. These patterns form a joint together with the same (yet inverted) cutouts on another plate. Such joints allow users to make 3D models using 2D laser cutting plans. This approach of constructing 3D models from 2D plates is much faster than additive fabrication processes like 3D printing.

If a model is in a 3D format (e.g., *flatFab* [9] or *kyub* [4]) applying parametric changes is easy. Most models shared online, however, are in the form of 2D cutting plans (95% of over 200,000 models [28]), which makes parametric changes demanding, slow, and error prone. A 9-plate VR headset model, for example, took study participants on average 24 min to modify, 11 out of the 13 models edited in 2D fit together in 3D [28]. The underlying problem is that the reasoning of which plates are affected and how, is governed by the underlying 3D model—which requires users to reconstruct the underlying 3D models in their minds—a complex and error-prone process.

Recent research addressed this challenge by offering a set of manual tools called *assembler*^{3} [28], that allow users to convert 2D cutting plans into a 3D format, thus allowing users to make parametric changes more efficiently on that 3D format.

Unfortunately, the manual reconstruction workflow implemented by assembler^3 is laborious. It still requires users to perform a number of manual assembly steps that are linear in the size of the model, and it requires users to have at least a basic understanding of the underlying 3D model. Yet, the vast majority of users are non-experts, who just want to fabricate/customize the model, without having to learn exactly how the model is assembled.

We present *autoAssembler*, a software tool that converts 2D cutting plans to a 3D format *automatically*, as illustrated by Figure 1. Since the search space is exponential in the number of joints, exhaustive search is impractical for any non-trivial model. AutoAssembler thus pursues only the most "promising" subset of candidates (beam search). It considers candidates as promising if they (1) contain no intersecting plates, (2) fit into a small bounding box, (3) use plates whose joints fit together well, (4) the plates do not add many unpaired joints, (5) make use of constraints posed by other plates, and (6) conform to symmetry axes of the plates. We validated autoAssembler by importing 66 laser cut models that contain at least one edge joint each (find the benchmark models in the auxiliary materials). In our technical evaluation, autoAssembler imported 79% models in fully automatic fashion, another 18% of models required 1-4 manual clicks using our disambiguation tool, achieving an overall success rate of 97%.

3 CONTRIBUTION, BENEFITS, AND LIMITATIONS

In this paper we make three main contributions:

First, we present the *autoAssembler* software tool and its underlying algorithm. AutoAssembler uses a beam search algorithm to search the exponential space of possible ways of assembling parts. Our main contribution lies in the heuristics that assesses partially assembled models in order to pick the most promising candidates for subsequent exploration; our method prefers candidates that (1) have no intersecting plates, (2) fit into a small bounding box, (3) use plates whose joints fit together well, (4) do not add many unpaired joints, (5) make use of constraints posed by other plates, and (6) conform to symmetry axes of the plates.

Second, we integrate autoAssembler with the code base of a 3D editor for laser cutting (*kyub* [4]), resulting in an integrated system that allows loading, editing, and writing 2D cutting files.

Third, we release the benchmark models we tested against as auxiliary materials.

Our approach is subject to the same limitations as assembler[3], in that it does not handle living hinges, moving parts, or joints based on stacked plates, glued connections, or bolts. AutoAssembler also does not apply to models where the structure is derived from the *shape* instead of the joints, in particular models only consisting of planar sections held together by cross joints (Figure 16a).

4 THE AUTOASSEMBLER ALGORITHM

When assembling a model, automatically or by hand, one explores a space of possible solutions that is factorial in the number of joints. Even if we reduced the search space by (1) limiting our search to joints that fit, (2) eliminating orientations that lead to intersections between the new plate and what has been assembled already, and (3) adding a method that looks up matching joints for the joint at hand in constant time, the search space remains too large for exhaustive search (see Figure 2). We achieve (3) using the *joint hash* from [28], which achieves this by storing geometric profiles of joints in a hash table—the joint and its counterpart share that profile, when looking up the profile of a joint in the hash table, it returns its counterpart as a collision in the table in constant time.

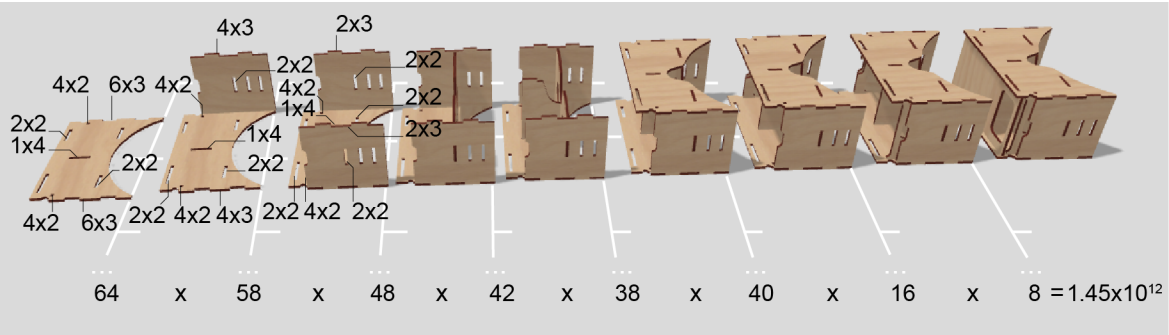

Figure 2: The search space for the simple VR headset consisting of 9 parts and 33 joints after limiting our search to joints that fit and only exploring orientations of plates that do not lead to an immediate collision. (Labels denote the number of joints that fit at a given position x the number of orientations they fit in).

AutoAssembler therefore limits the search to the more promising candidates at each stage (*beam search* [6]). To this end, autoAssembler starts with an empty model and recursively tries to add one plate each time. The time-complexity of beam search is O(dk) where depth d is the number of plates and k is the *beam width* (the number of candidates autoAssembler picks to generate children for, at each stage) multiplied by the maximum fanout at each stage. The fanout

in principle is proportional to the depth. Worst case complexity thus is quadratic, however the joint hash table mentioned before reduces the fanout to the joints with the same signature. If all joints have a unique counterpart (ideal case) the complexity is linear, in practice the complexity sits between these bounds. In our technical evaluation, a *beam width* of 4 proved sufficient for achieving the aforementioned success rate (79% + 18% = 97%), allowing for overall very fast execution (median of 0.30s).

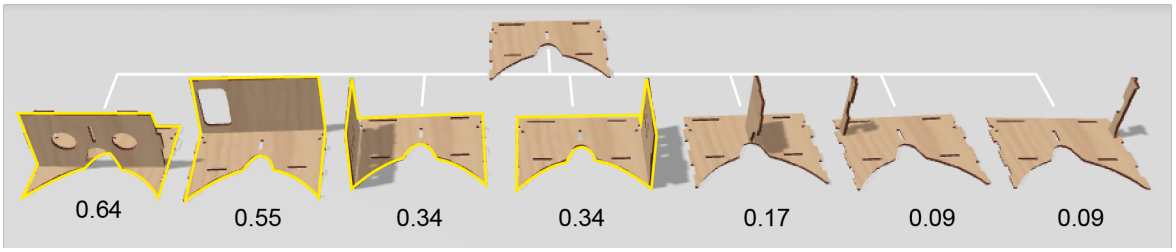

Figure 3: The first stage of search for a VR headset. AutoAssembler picks the initial plate with the most joints, uses the joint-hash to find what plates fit into the open joints, scores the candidates (labeled above) and generates new children from the best candidates.

As shown in Figure 3, performing beam search, autoAssembler selects the four highest scoring candidates to generate the candidates for the next round, detailed in pseudo code in Algorithm 1.

Algorithm 1: Find best candidate

Input: starting plate, detected plates and joints in 2D cutting plan (using the assembler[3] algorithm [28])
Output: best assembled model
Internal data structures: *candidates* are assemblies of one or more plates, their children are the same assembly with one additional plate. Empirically determined maximum beam width MAX_BEAM_WIDTH=4

```
// These candidates have two plates, (see figure 3 for examples)
currentCandidates = children of the candidate, which only contains the starting plate
// Based on heuristics in chapter 4.1, score each candidate
score(currentCandidates)
while there is at least one currentCandidate and not all plates are used {
        // AutoAssembler groups candidates that use the same plates, but not using the same joint or
        orientation, see heuristic "minimize candidates that are highly similar"
        groups = group currentCandidates together, which have the same "connection pattern"
        currentCandidates = Highest scoring candidate from each group
        // If there are more states than the maximum beam width, limit them based on score
        currentCandidates = currentCandidates limited to MAX_BEAM_WIDTH
        // Generate children by adding a plate to the current candidates in different orientations
        currentCandidates = children of currentCandidates
        // Based on heuristics in chapter 4.1, score each candidate
        score(currentCandidates)
        }
return currentCanddiate with highest score
```

As part of the search process, as illustrated by Figure 4, autoAssembler eliminates duplicate candidates. It achieves this by storing previously visited candidates (*memoization*) in a hash.

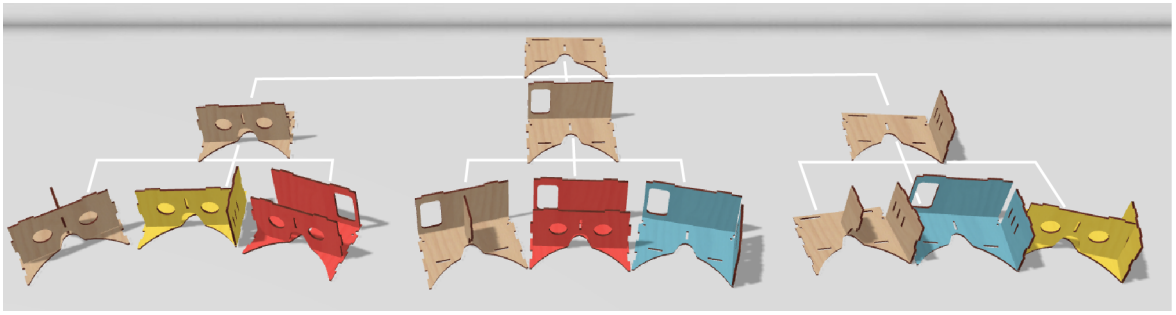

Figure 4: autoAssembler encounters a candidate model more than once (the colored candidates), autoAssembler drops the redundant states (Memoization), by hashing visited candidates. (Here shown with three candidates each for visual clarity).

4.1 How autoAssembler picks promising candidates: The heuristic function

The main contribution of autoAssembler is the specific way it selects the candidates it pursues, i.e. how it assesses the potential of each candidate (its *heuristics function* [6]). It computes a weighed sum to prefer candidates (1) that have no intersecting plates, (2) that fit into a small bounding box, (3) that use joints that can be unambiguously matched, (4) that do not add a large number of unmatched joints, (5) that make use of constraints posed by other plates, and (6) that conform to symmetry axes of the plates. We developed these heuristics based on our observation of common patterns in laser-cut models and by manually evaluating candidates in our engineering team. As autoAssembler calculates a score for all candidates at every stage, the implicit objective for these heuristics is that they are efficient to compute.

AutoAssembler aggregates six heuristics as a weighted sum. Two additional heuristics (deduplicating symmetric/similar plates and minimizing highly similar candidates) are procedural in nature as they operate on the *stage* (all "current candidates") rather than scoring individual candidates. We determined the optimal weights of the individual heuristics using hyperparameter optimization (see "technical evaluation" for details):

Table 1: parameter optimization for the heuristic function

parameter	weight
compactness of candidates	0.07
intersections between plates	0.63
ambiguity of the joints that are completed	0.88
minimizing the number of unmatched joints	0.95
make use of constraints posed by other plates	0.58
conform to symmetry axes of the plates	0.83
minimize candidates that are highly similar	n/a
deduplicating symmetric plates and similar plates	n/a

1. Give preference to compact candidates: parts that "stick out" of a model break off easily. Since designers generally prefer sturdy designs, laser-cut models tend to be "compact", i.e., fit into a comparably small encompassing volume. AutoAssembler therefore is designed to prefer candidates that fit into smaller bounding boxes. The red plate in Figure 5 for example, could be assembled as shown in (a), but that increases the spanned volume (calculated using the axis-aligned bounding box as this is the cheapest metric to compute). AutoAssembler calculates this "compactness" using the metric:

$\frac{(surface\ area)^{1.5}}{volume}$, as proposed by Parker et al. [27]. The compactness score of Figure 5b is much higher (the surface area remains the same, but the denominator is much smaller), so autoAssembler gives this candidate a higher score. Note that the weight of this heuristic is very small (0.07) and thus mostly serves as a tie-breaker for the other heuristics.

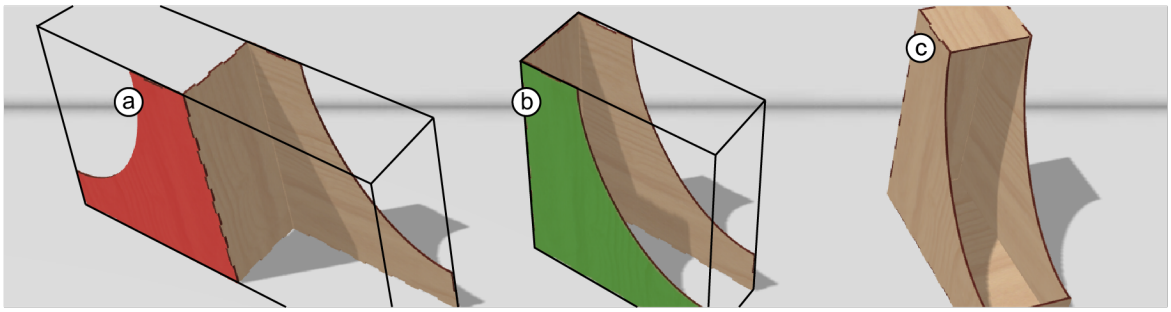

Figure 5: (a) Adding the red plate gives it a much larger bounding box, autoAssembler thus gives (b) this candidate the higher score. (c) The resulting magazine holder.

2. Avoid intersections between plates: laser-cut plates in a model must not intersect. Computing intersections between plates is an expensive operation as it requires comparing every outline feature of the plate to the already existing candidate. To achieve this efficiently, autoAssembler compares the bounding box of the newly added plate to plates already present in the current candidate. In the train wagon of Figure 6, autoAssembler initially prefers to put the wheel mount up because of the compactness metric, however that causes an intersection, which forces autoAssembler to assemble this part in another orientation. Avoiding intersections is not a hard constraint, because the relatively cheap method of computing intersections comes at a cost of accuracy.

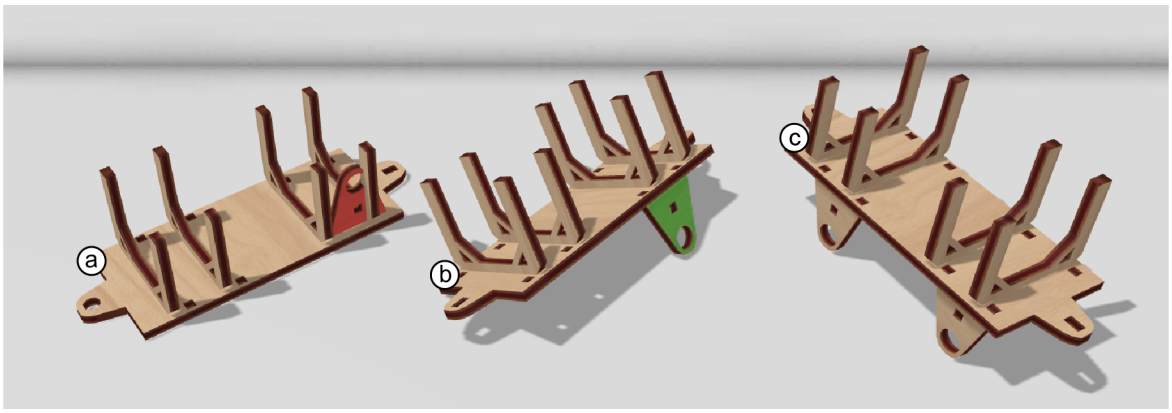

Figure 6: (a) The compactness heuristic suggested mounting this bearing (red) on top of this train wagon, but here it intersects with plates mounted on top of the wagon, causing autoAssembler (b) to flip the bearing plate to its correct position. (c) the same heuristic fixes the other bearings too.

3. Give preference to unambiguous joints: AutoAssembler delays inserting plates that can be mounted in many different ways as long as possible, so as to await additional plates to introduce additional constraints that can help make the decision. AutoAssembler achieves this by giving preference to joints that have few, or ideally only a single matching

partner joint. More specifically, autoAssembler assigns a probability to pairs of joints in the joint hash as proposed by assembler^3 [28] and it tries to maximize the ratio between the assigned probability and the sum of the probabilities of all other ways of matching up this joint. When assembling the dice tower shown in Figure 7, for example, starting with the ambiguous single finger joint of the red plate creates many different opportunities for mounting the top plate. (b) AutoAssembler instead first assembles the unambiguous and long finger joints, which then pose constraints on the red plate of Figure 7a, reducing overall ambiguity.

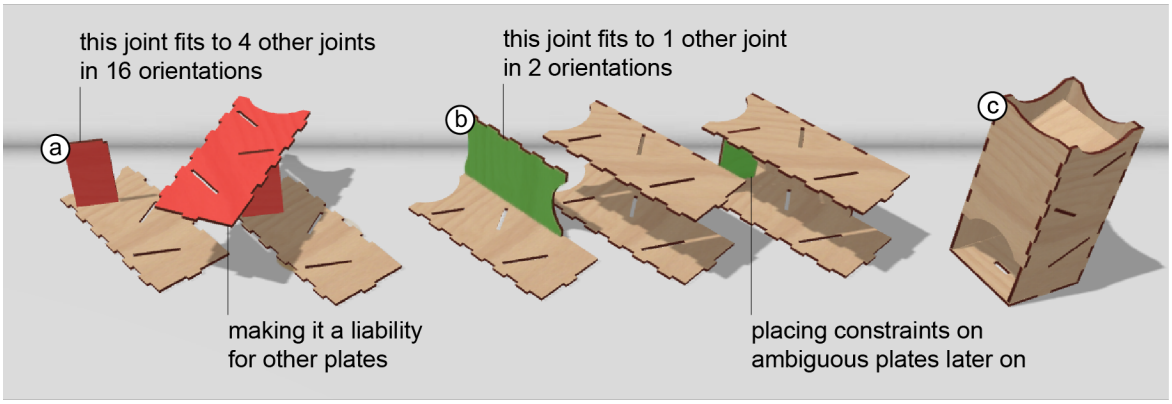

Figure 7: (a) Assembling this ambiguous joint early on forms little or no constraints on other plates, as a result the top plate here can be assembled in many different ways (b) autoAssembler prefers to greedily connect plates with high probabilities. This adds constraints for other plates, (c) to eventually make this dice tower.

4. Minimize the number of unmatched joints: the size of the search space at every candidate correlates with the amount of unmatched (unused) joints. AutoAssembler prioritizes plates that add the fewest incomplete joints. This works because it started out with the plate having the most joints, otherwise autoAssembler would paint itself into the corner (e.g., start with a plate with one joint, then close that joint without opening new ones–done). In the example shown in Figure 8, for example, autoAssembler therefore does not add (a) the side plate that brings in multiple new joints but runs with (b) the middle divider, which only adds one new joint.

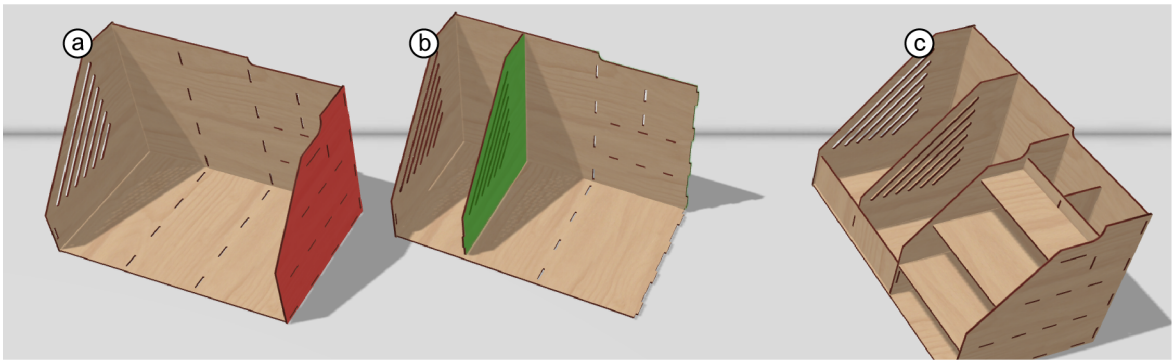

Figure 8: AutoAssembler prioritizes completing joints first as this reduces the search space: (a) inserting the side plate (red) would add four incomplete joints to the model. (b) AutoAssembler therefore rather adds this "divider" plate, which only adds one unmatched joint. (c) leading to this desktop organizer.

5. Make use of constraints posed by other plates: AutoAssembler prioritizes inserting plates whose placement is supported by multiple plates/joints already in the model. It tries to maximize the number of completed joints by adding a plate. This avoids situations as shown in Figure 9, where (a) the nose-piece of this VR headset is under-constrained: it can be assembled in different orientations that all seem equally good according to the other metrics. (b) AutoAssembler thus prioritizes assembling the front plate first, which completes three joints at once and then later (c) adds the nose-piece as the front plate imposes additional constraints on that plate.

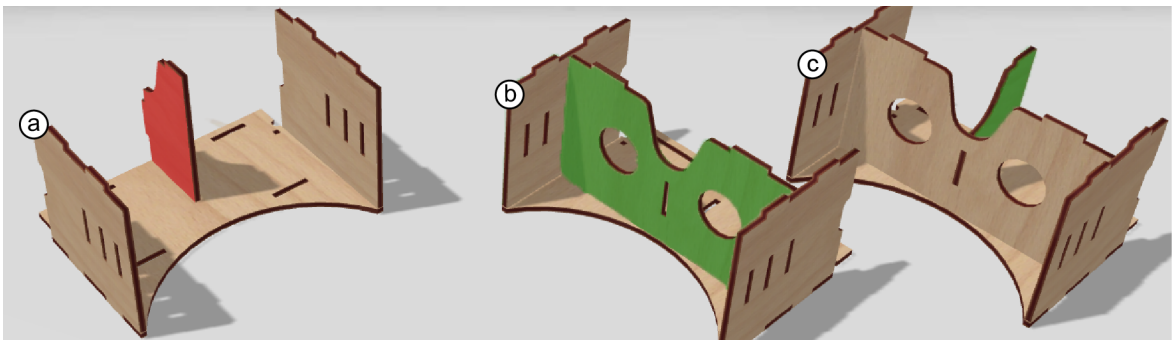

Figure 9: (a) This candidate offers too few constraints to orient the red plate correctly. (b) AutoAssembler therefore prioritizes this plate, which completes three joints. (c) This adds constraints that come in handy when eventually inserting the middle piece.

6. Minimize candidates that are highly similar: as illustrated by Figure 10, if autoAssembler encounters multiple candidates that differ only by one or more plates being flipped, it drops all but the highest scoring one, so as to make room for candidates consisting of a different subset of plates. As shown in Algorithm 1, candidates are filtered by "connection pattern". This is a high-level data structure that describes what plates are connected, using which joints. This prevents AutoAssembler from only looking at similar structures with a flipped plate.

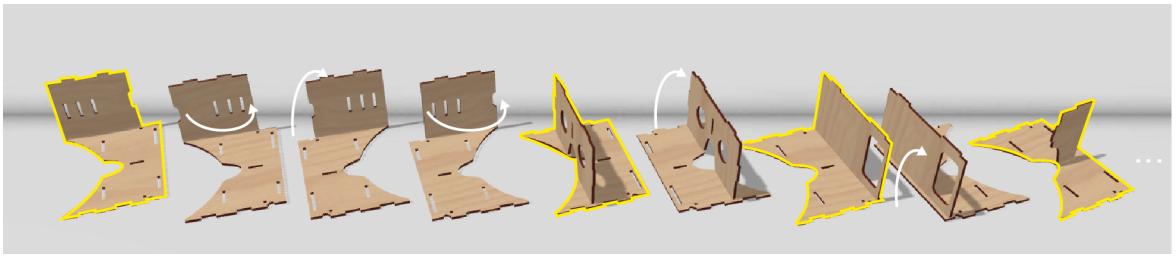

Figure 10: AutoAssembler picks the best four candidates of this stage. To avoid picking the first four candidates which are almost the same, it skips candidates that share the same connection pattern with a candidate that is already picked. Resulting in the four candidates highlighted in yellow.

The algorithm, as described above is functional, but performs poorly on models containing symmetries, such as the models shown in Figure 11. On such models the search space is cluttered with results that are the same, but contain different plate connectivity, resulting in problems similar to the ones in Figure 10. This unnecessarily blows up the search space and deprioritizes asymmetrical plates in the assembly that end up defining the structure. This is problematic, as 3D models designed for laser cutting are commonly symmetrical in nature. Out of the benchmark of assembler^3, for example, 81/100 have reflective symmetries, and 11/100 have rotational symmetries.

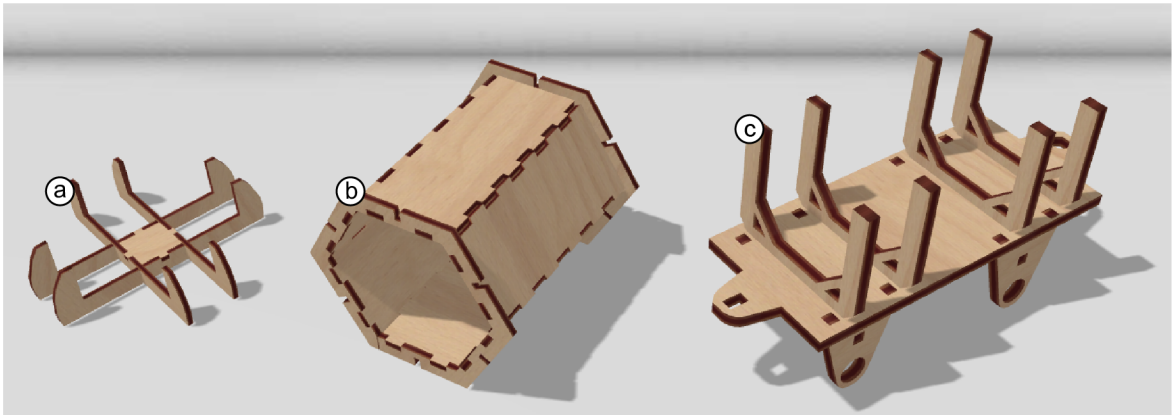

Figure 11: Examples of symmetric laser-cut models from assembler^3 [28] (a) double reflectional symmetry, (b) 6-point rotational symmetry, (c) double reflectional symmetry (and multiple uses of same plate).

We propose two extensions of the algorithm based on symmetries: prioritizing symmetric assembly of plates, and deduplicating plates and orientations when possible.

7. Favor symmetric candidates: AutoAssembler prefers symmetric assemblies over asymmetric ones. As shown in Figure 12b, when autoAssembler adds a plate to a symmetrical plate, it verifies whether there is a similar plate elsewhere in the assembly and if so, it increases the score of a placement that implements symmetry.

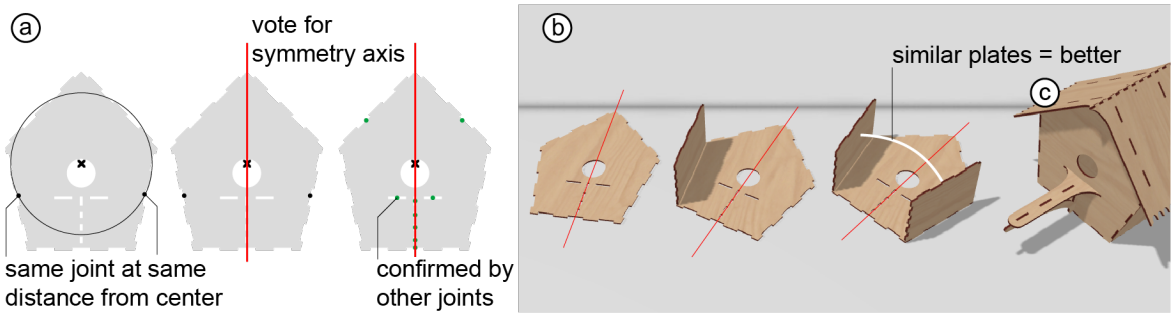

Figure 12: (a) AutoAssembler detects symmetries by having pairs of joints vote for symmetry axes (b) autoAssembler uses the information to prefer similar plates connected to joints on opposite sides of the symmetry axis. (c) resulting in this birdhouse.

As shown in Figure 12a, autoAssembler detects symmetries in three steps: (1) it starts by looking for joints with a similar profile using the joint hash table, at the same distance to the center of the plate, (2) it constructs the symmetry axis this pair of joints conforms to, and (3) then verifies that proposed symmetry axis with the other joints on the plate, similar to Mitra et al. [24].

8. Deduplicating symmetric plates and similar plates: Symmetric plates blow up the search space unnecessarily: When encountering a symmetric plate, such as the one shown in Figure 13, the basic version of autoAssembler considers inserting it in all possible orientations, leading to a much bigger search space with a lot of candidates that turn out to be geometrically identical.

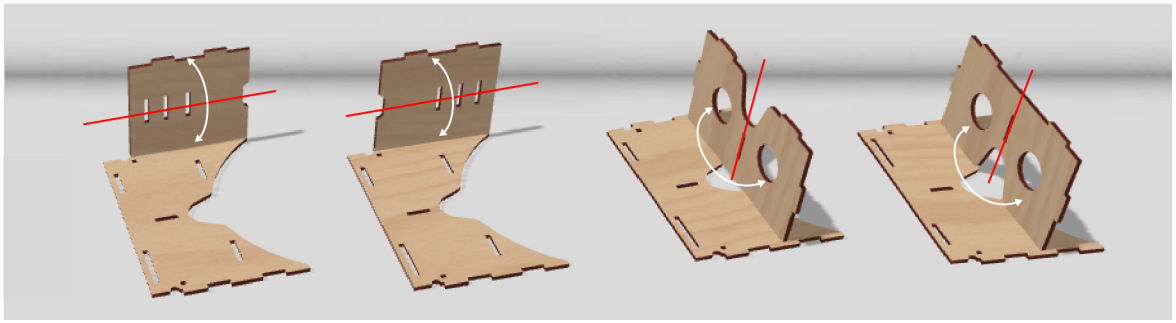

Figure 13: Because of the horizontal symmetry axis, only these two orientations of the side plate produce a unique state. The same for this in-plate symmetry in the vertical orientation.

A similar issue is caused by multiple identical plates, such as the ones shown in Figure 14. Again, the basic version of autoAssembler considers inserting each copy of that plate separately, thereby blowing up the search space and increasing the risk of beam search dropping relevant models, as the algorithm is instead pursuing multiple essentially identical models.

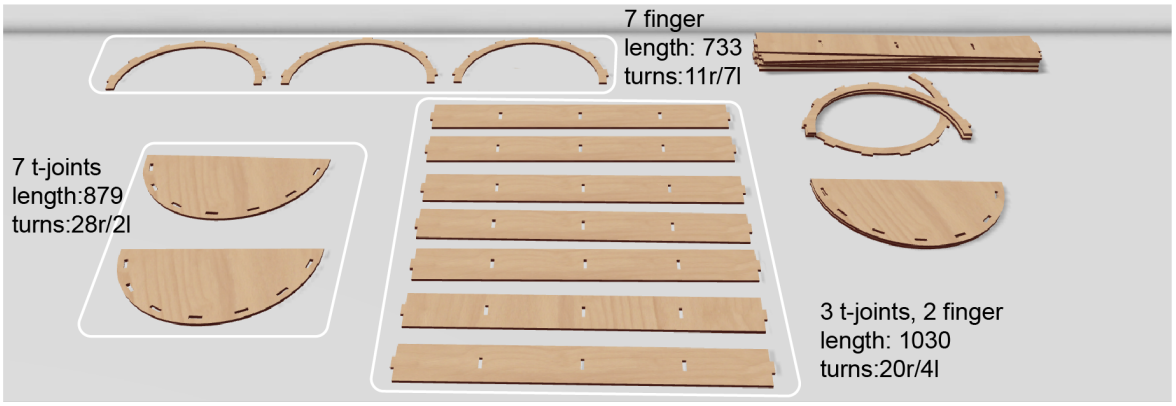

Figure 14: Clustering by similarity (a) autoAssembler characterizes each input plate by three easy-to-compute metrics (b) For this barrel model, autoAssembler considers 3 types of plates with 19 joints as opposed to 12 types of plates with 58 joints.

AutoAssembler determines that two plates are identical by comparing their outlines using efficient-to-compute characteristics: the number and the types of joints, the length of the outline, and the number of left/right turns along each outline.

For hand-drawn models or models subject to rounding errors these metrics may differ by some epsilon. To overcome these imprecisions, autoAssembler uses a density-based clustering algorithm (DBSCAN [13]), which allows autoAssembler to cluster similar plates, without knowing in advance how many clusters to look for.

For the model shown in Figure 14, for example, autoAssembler reduces this model from 12 types of plates featuring 58 joints down to 3 types of plates featuring 19 joints, which heavily reduces the search space.

Favoring symmetry and similarity detection allows autoAssembler to correctly reconstruct the six models shown in Figure 15, thereby increasing autoAssembler's success rate. It also improves the algorithm's performance by a factor of 1.5.

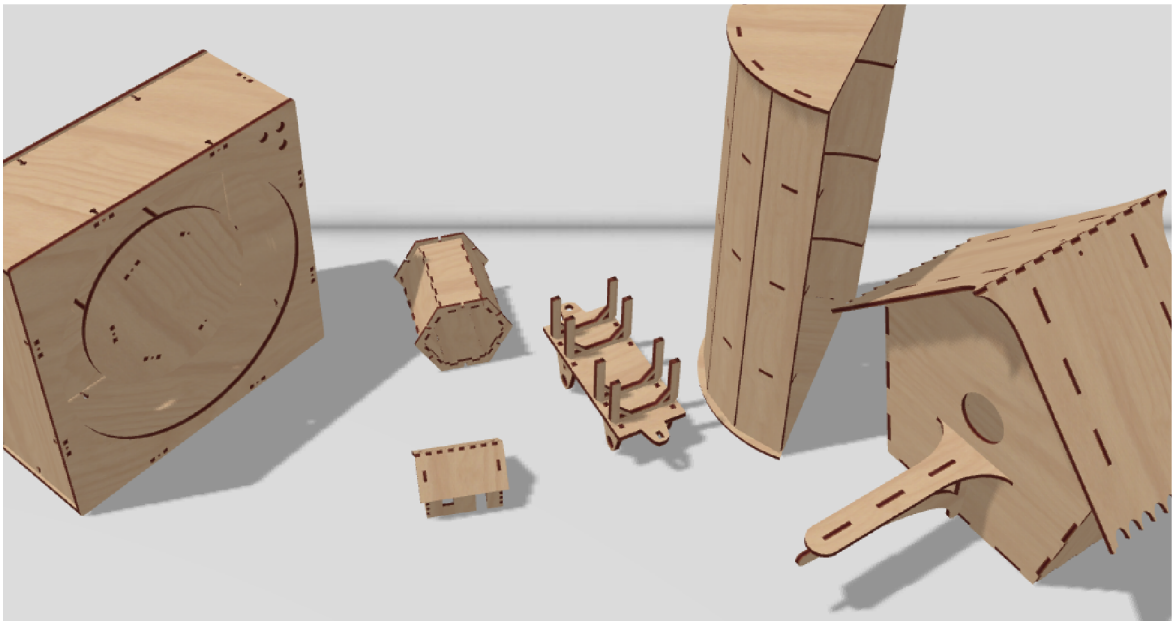

Figure 15: Six models from the test set that assemble correctly in autoAssembler because of the symmetry heuristics.

4.2 Manual disambiguation

Some models do not contain sufficient information to automatically complete the model. The triceratops shown in Figure 16, for example, would require *domain knowledge* of the anatomy of dinosaurs (as a paleontologist would have) to tell how to sort the ribs, or at best a visual reference for what to assemble (as a child may have puzzling the dinosaur together). AutoAssembler does not have this domain knowledge and consequently it precisely fails to assemble models of this type—this is a limitation of the system and the reason we exclude from our analysis this particular type of decorative models, which are based on cross joints alone.

Models that do not solely rely on cross joints, however, tend to have only a small number of such ambiguities and these generally do not derail autoAssembler. The remaining 14 models that autoAssembler did not automatically assemble, had a few plates that were not captured by the general heuristics of the algorithm. We address these by complementing autoAssembler with two manual tools that allow disambiguating these cases:

1. Clicking a plate using the "reorient plate" tool reorients the clicked plate by forcing autoAssembler to re-evaluate the plate's orientation. Users keep clicking until satisfied with the plate's orientation.

2. Clicking a plate using the "swap plate" tool swaps a plate with another selected plate if their joints match up. Users click on one of the plates, which gets highlighted and then click on the other plate to swap them out.

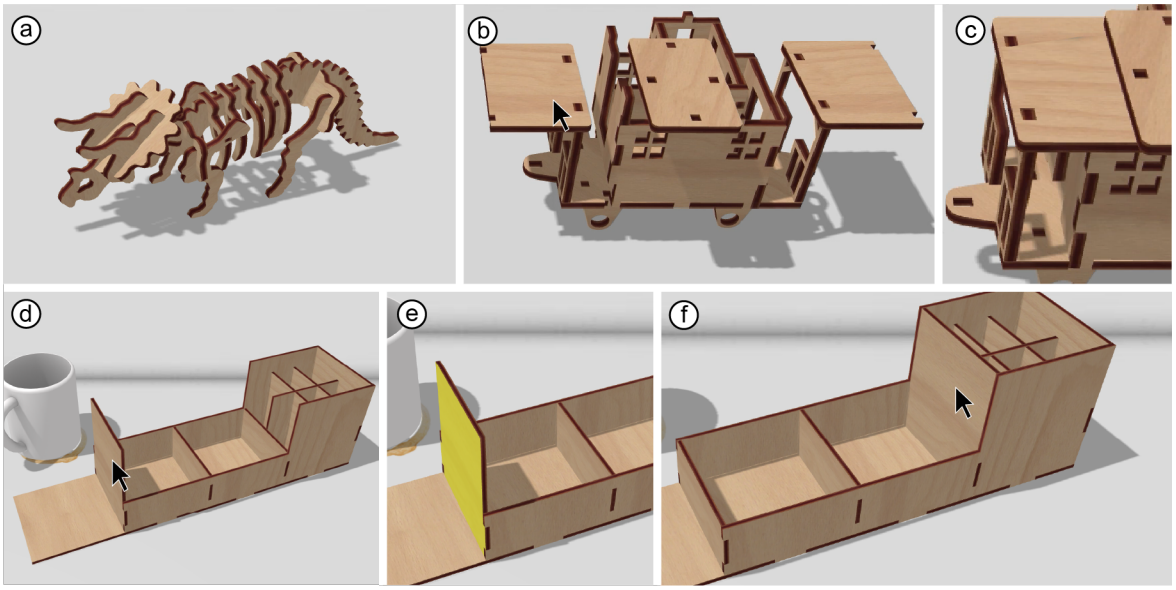

Figure 16: (a) Models such as this dinosaur require domain knowledge, placing them outside the scope of automatic assembly. (b) this train wagon has misassembled plates after automatic assembly, with the reorient tool this is quick to fix (c) users click a wrongly assembled plate, which gets re-evaluated by autoAssembler.(d) Two plates are swapped in this organizer. (e) The "swap-plate" tool lets users select one of the plates, and (f) by clicking the other one they swap if they share common joints.

In our technical evaluation, fully automated use of autoAssembler assembled 79% of all models correctly. The "re-orient" and "swap" tools allowed fixing an additional 12 of 14 models using 1-4 such clicks, resulting in a 97% success rate.

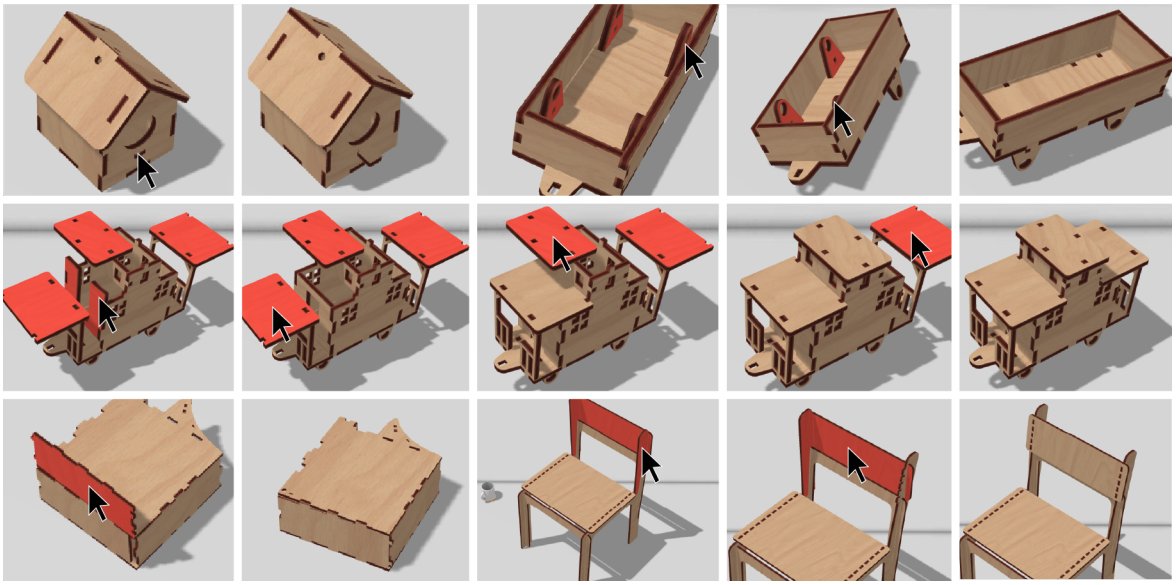

Figure 17: Click sequences of the disambiguate tool. Users click a poorly assembled plate, which autoAssembler then reconsiders. Here are five models that all were fixed by 1-4 manual disambiguation overrides (2.7 on average).

4.3 Integration of autoAssembler with kyub

Figure 18 shows the integration of autoAssembler into kyub [4]. When users import an SVG file, the dialog window shows a live preview of autoAssembler assembling the model. AutoAssembler completes the import after a median of 0.30s, (see "Technical Evaluation").

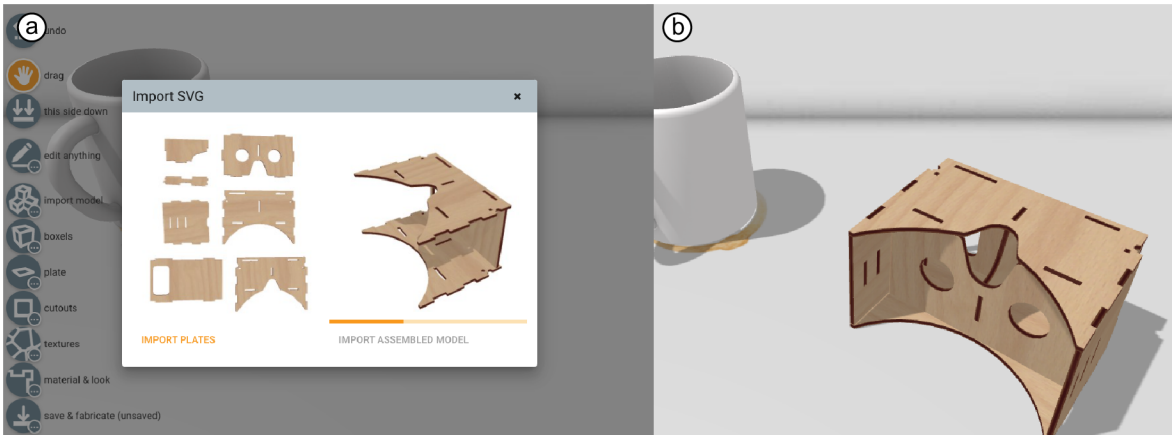

Figure 18: Importing an SVG model automatically into kyub.

5 RELATED WORK

Our work builds on research into reconstruction of 3D models and related search algorithms, combinatoric puzzle solvers, sharing and remixing of models for 3D printing, and portable formats for laser cutting.

5.1 Reconstruction of 3D assemblies

Traditionally, 3D reconstruction in computer vision has focused on recovering the surface of 3D models, however this does not reveal the underlying structures of functional 3D assemblies. One approach to reconstructing 3D models is to reconstruct based on one or more 2D images: *3-Sweep* [8] reconstructs a 3D model from a single photograph. Agrawal et al. [1] take the opposite route by reconstructing 3D models of historical monuments by aggregating massive amounts of pictures found online. More recently, the focus of this field has shifted to machine learning-based approaches that approximate the 3D shape from images, in the form of voxel-based representations [10], and for generating 3D surfaces [15].

Besides visual reconstruction of 3D models, there has been a large interest to reconstruct 3D models for fabrication as well. An interesting take from Liu et al [23] allows users to bend structures from wires, which they then scan and reconstruct into a 3D model. Lin et al [22] use raw scans of moving mechanisms and generate functional 3D printable mechanisms from that. Xu et al. [43] reconstruct 3D mechanisms from multi-view images and even reconstruct the movement of the mechanisms from a corresponding video. StructureNet [25] opens up the space to extend such approaches towards structural modeling, considering how primitives within models are connected rather than just their outside shape. Editors for folding [32] allow users to take 2D plans and turn them into 3D models as long as edges connect, similarly Berthouzoz et al., [5] do this with sewing patterns to create 3D models of garments.

Once a 3D model has been created, *InverseCSG* [12] lets users turn a non-parametric 3D model (e.g., STL) into parametric models, allowing users to modify the models using advanced 3D operations. It reconstructs what geometric primitives and CSG operations were required to get to an input 3D model. IkeaBot [20] uses analysis of geometric parts of IKEA furniture to reconstruct a 3D model and then create assembly instructions for robots to put it together.

The closest approach to our algorithm comes from Willis et al [42] who reconstruct 3D models of broken pots in an archeology context. They geometrically attempt to line up the shreds from pots using a Bayesian Maximum Likelihood Estimation algorithm. This approach works great because there is only one way the pieces fit together and in the context of archaeology, three hours of computation time to solve the problem is totally acceptable.

5.2 Solving combinatoric puzzles

When arranging 2D laser-cut plates in 3D autoAssembler solves a combinatoric problem. Combinatoric problem solving has a long history in the algorithmic solving of *2D* puzzles. Demaine & Demaine [11] show that jigsaw puzzles are in principle NP-hard. Nevertheless, Sholomon et al. [33] show that large jigsaws can be solved using a genetic algorithm, given good "compatibility" metrics describing how well the pieces fit together.

Chen et al. [7] solve puzzles based on a complete joint map between any kind of "objects". This is very similar to the joint hash autoAssembler uses to find matching candidates. Combined with knowledge of the constraints of orientations of laser-cut joints and the heuristic function that assesses the success of each candidate, autoAssembler makes that problem tractable.

On the flipside, researchers have developed various algorithms to create puzzles from 3D models, in some ways approaching the problem from the opposite side. Xin et al., [40] turn 3D models into bur puzzles, Tang et al., [36] convert to 3D dissection puzzles and Song et al., [34] create interlocking puzzles from original 3D models.

5.3 Sharing and remixing of 3D models: the road to success for ubiquitous personal fabrication

Flath et al. [14] studied sharing and remixing behavior on thingiverse and highlight the impact of the thingiverse *customizer* (a tool in the browser to modify other people's parametric models). They show how an increase in remixing and

building on the work of others lead to a massive increase in the number of models being shared on the platform. Alcock et al. [2] agree that the customizer is powerful but add that it lacks expressivity and ease-of-use. Grafter [29] is a software tool targeted to facilitate such forms of online remixing in the context of 3D printed mechanisms, and the *PARTS framework* [17] enables users to specify mechanical parametric models, which again fosters remixing and modifying other people's models.

ShapeAssembler [19] drives this one step further by developing a domain specific language that describes how geometry is connected and what is structurally sound assembly. They use this language to train a neural network on available shape repositories to then allow users to edit models by synthesizing assemblies as users modify parameters of the "program of the 3D model".

In observing the maker community, Hudson et al. [18] identified the need for better tools for remixing and customizing. And related to that, Stemasov et al., [35] argue for remixing and customization as a sweet spot between modeling and simply downloading models made by others and that this a key enabler for making personal fabrication truly ubiquitous. We have seen this play out in other fields as well such as the open-source software community [21]. We add that this applies just as much to the context of 3D models for laser cutting.

5.4 Portable formats for laser cutting

Portability in the context of laser cutting would imply that laser-cut models are encoded in a way that everyone can reproduce them and edit them in meaningful ways. It turns out that the commonly shared 2D cutting plans fail to deliver on both counts. SpringFit [30] made the case that models are not reproducible as they implicitly encode the kerf (removed material) of the laser cutter, resulting in models that fail to fabricate on a slightly different machine. SpringFit and KerfCanceler [31] combined take 2D cutting plans and rewrite joints, mounts and mechanisms (that cause problems with kerf variations) to versions of these elements that are invariant to kerf.

Editing 2D cutting plans is an even bigger struggle. A major milestone in this was the shift towards 3D modeling environments for laser cutting, specifically FlatFab [9]. CutCAD [16] follows a similar approach but uses a combination of 2D input and 3D visualization. CODA [37] is a Fusion360 plugin that provide assistance for alignment and modifications of 3D models for, among others, laser-cut models. In the context of furniture design, there is extensive work on modeling interlocking structures [38]. Kyub [4] further increases the level of abstraction by representing laser cut models as volumetric models. However, the majority of models still exist in the form of 2D cutting plans.

Assembler^3 [28] proposes an alternative workflow to take existing 2D cutting plans, reconstruct them in 3D, make the required 3D edits and then export it back to 2D cutting plans for laser cutting. The vast majority of models that are out there (on shared repositories or in industry databases) still exist in that 2D format. AutoAssembler takes that powerful contribution of assembler^3 and goes a step further by solving a second order effect it creates: the manual reconstruction in assembler^3 is like solving a combinatorial puzzle, to solve it users need to know how the plates in the model *should* be connected. This is trivial for the original modeler or the thousands of people with laser cutting experience, but a challenge for the millions of users that may be interested in using the model. The manual workflow also takes time, which places a large burden on the laser cutting experts to deal with this legacy problem. Automating this workflow makes the conversion and thus the access to the 3D models inclusive for anyone who wants to use and edit the models.

6 TUNING THE ALGORITHM

We ran a series of tests to optimize key parameters of the algorithm. (1) Weights for the heuristics (2) determine minimum beam widths, and (3) what plate to start out with.

6.1 Test set

We created the test set starting with the test set of the *assembler*[3] project [28], from which we extracted the 34 "planar section" models, as discussed above in "manual disambiguation" section. When models consisted of multiple assemblies, we split these into separate files as autoAssembler expects one assembly per model. They can be loaded into the same kyub scene though but through 2 import sessions.

We measured the success rate by taking the ground truth models that were manually assembled in assembler[3] as a reference. We automatically verify our test runs by comparing the distance and angle between plates to these ground truth models. We considered a model to be a success only when all the dimensions matched perfectly (e.g., there is no 50% successful assembly).

6.2 Procedure

We measured success rate (percentage of models that assembled correctly) and the run time and repeated every measurement 10 times to compensate for performance glitches and any potential delays confounding our measure due to background activities on the machine (MacBook Air 2020 1.2GHz Quad Core Intel Core i7).

6.3 Composition of the heuristic function

To determine the right weights for the parameters of the heuristic function, we ran a hyperparameter optimization algorithm using the tree pazen estimator called ATPE, proposed by Wen et al [41]. We trained the algorithm by feeding it the parameters of random runs on the benchmark, and the corresponding candidates. The candidates are labeled automatically by comparing it to our ground truth distance matrix. After assessing 1000 candidates the algorithm converges, we found the optimal parameters presented in Table 1.

6.4 The beam width: from 4 on, autoAssembler achieves maximum success rate

The beam width is the number of candidates autoAssembler selects at every stage after sorting the states. To achieve the optimal success rate and performance trade-off, we ran the benchmark with increasing beam widths until the success rate not increased further. We also did a run with a beam width of 1, which is equivalent to best first search to see if the heuristic function alone (without beam search) would yield sufficiently good results. Results are shown in Figure 19 below.

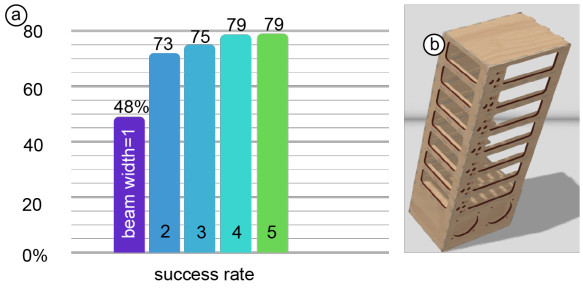

Figure 19: (a) Success rate of autoAssembler on our benchmark while varying the beam width. (b) This model still makes an improvement at a beam width of 8, but the associated performance loss is not worth it.

As shown in Figure 19, after a beam width of 4, the success rate stabilizes. To verify whether if the success rate had reached an upper bound, we ran the remaining 18% of models with a beam width of 10 as well. Apart from the raspberry-

pi rack shown in Figure 19b, which after some more testing improved at a beam width of 8, there was no more progress. We also see that with a greedy best first search, we could still achieve a success rate of 48%, which indicates that the heuristic function alone is rather good at picking the right option, but in many cases, we do benefit from searching more alternatives.

The median performance per model in each run was 0.11 (beam width =1), 0.22 (2), 0.22 (3), 0.30 (4), 0.44 (5), increasing the beam width scales the performance roughly linearly. Therefore, doubling the beam width (and thus the run time) from 4 to 8 is not worth it, only to save a single model in our test set.

6.5 Start with the plate with most joints

To know what role the starting plate plays in the success rate of the algorithm, we ran the benchmark with different starting plates: (1) the plate with the most joints, as this puts the most constraints on the assembly (2) the biggest plate as this would define most of the shape, and (3) a random plate (the plate that contains the first path in the SVG), as a baseline. Figure 20 shows these strategies at the example of a test tube rack.

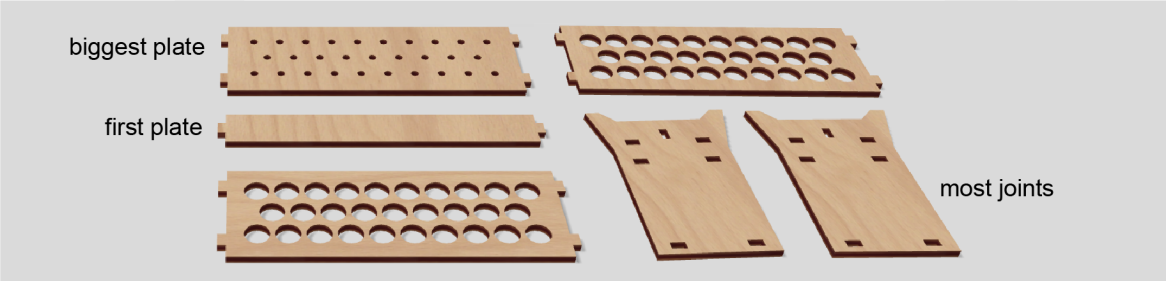

Figure 20: Different starting plate metrics for the test-tube model.

The results show that the best solution is to start with the plate with most joints. To see if there are better options for the models that do not succeed, we ran detailed tests with those models where we started out every plate. Some of the broken models get closer to success by picking a different starting plate, but none were “fixed” by doing so. We thus stick to the plate with most joints as this is cheap to compute.

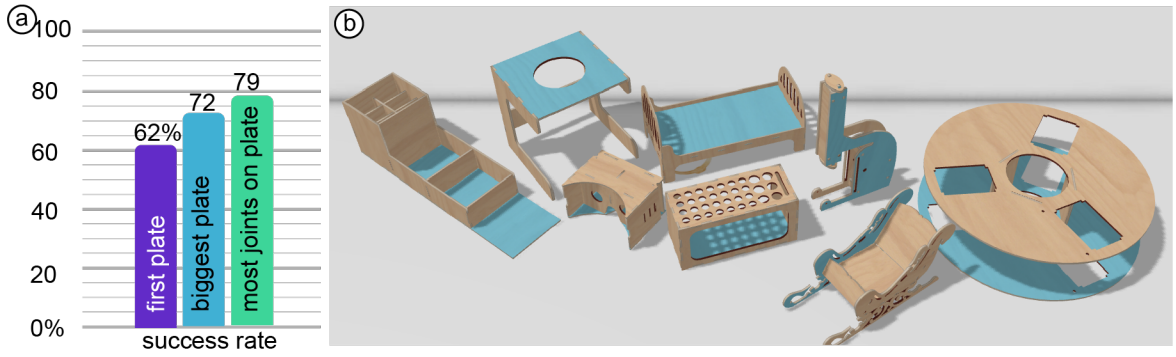

Figure 21: (a) Results of varying the starting plate. (b) some models with their ideal starting plate highlighted.

7 TECHNICAL EVALUATION: AUTOASSEMBLER ACHIEVES A 97% SUCCESS RATE

To evaluate the autoAssembler algorithm, we ran it on the benchmark of 66 models encoded as 2D cutting plans. To determine how much the extension of the algorithm contributes to the overall success rate of the algorithm, we ran the tests in four distinct conditions: (1) the basic algorithm, (2) the base with detection of similar plates, (3) the fully extended automatic algorithm with symmetry detection and similarity detection, and (4) with the manual disambiguation tools.

We used the same test set and procedure as presented in the previous section.

7.1 Results

As illustrated by Figure 22, the complete algorithm of the eight heuristics and the symmetry/similarity extensions, combined with the manual disambiguation tools resulted in a 97% success rate.

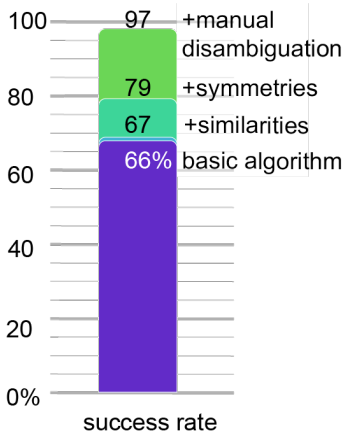

Figure 22: The overall success rate of autoAssembler is 97% based on three extensions of the algorithm: detecting similarities, symmetries and manual disambiguation.

As shown in the diagram, adding the similarity detection alone does not impact the success rate, which is unsurprising as it only reduces the search space (and thus contributes to performance). The model that did get fixed in the process failed before because the search space was overly populated with candidates that were the same.

Symmetry and similarity handling account for 13%: To assess the contribution of the symmetry and similarity detection, we ran the benchmark with each of these steps enabled and disabled. The results shown in Figure 22 show that the symmetries and similarity detection combined increase the success rate from 66 to 79%.

Symmetry and similarity detection reduced the runtime from a median of 0.44s per model to 0.30s per model (a 1.5x performance improvement).

Manual disambiguation accounts for 18%: As shown in Figure 23, 12 models required tweaking using the manual disambiguation tools, although the required user effort was minimal with 1-4 clicks required per model (avg 2.7).

Figure 23b finally shows the two failed models: a minibar model and a birdhouse the entrance of which was flipped inwards. Both of these can be fixed, but require the use of an additional tool that extracts 6 plates from the models and assembles them manually.

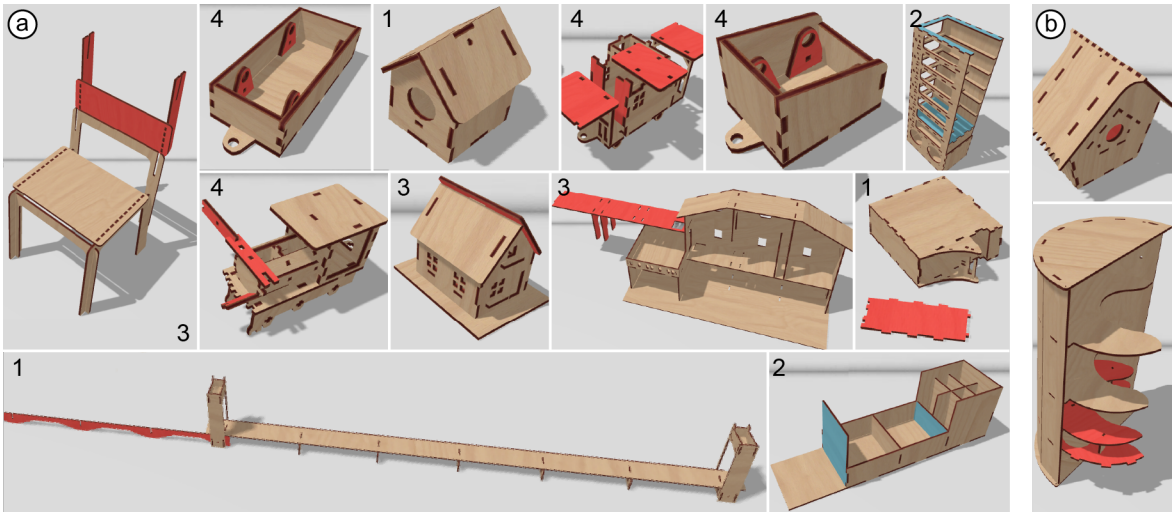

Figure 23: (a) 1-4 clicks using autoAssembler's "re-orient plate" tool fix the mis-oriented red plates in these 12 models from the test set. One pair of clicks each using the "swap tool" fixes the swapped blue plates. (b) this birdhouse and minibar would require users to extract 6 plates and re-assemble them back into the model.

8 CONCLUSION

In this paper, we presented autoAssembler, a software tool that converts 2D cutting plans to 3D models. In our technical evaluation, autoAssembler assembled 79% of models fully automatically, while 18% of models required an average of 2.7 clicks of post-processing. AutoAssembler offers a faster and easier workflow for reconstructing 3D models from 2D cutting plans than earlier manual methods [28]. Hence, autoAssembler brings the field closer to transitioning from 2D cutting plans to 3D models, thereby providing an improved format for storing and sharing laser cut models. As future work, we plan to reconstruct laser cut cross section models by using SVG files *and photographs* of the assembled model as input.

REFERENCES

[1] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M. Seitz, and Richard Szeliski. 2011. Building Rome in a day. *Commun. ACM* 54, 10 (October 2011), 105–112. DOI: https://doi.org/10.1145/2001269.2001293

[2] Celena Alcock, Nathaniel Hudson, and Parmit K. Chilana. 2016. Barriers to Using, Customizing, and Printing 3D Designs on Thingiverse. *In Proceedings of the 19th International Conference on Supporting Group Work (GROUP '16).* Association for Computing Machinery, New York, NY, USA, 195–199. DOI:https://doi.org/10.1145/2957276.2957301

[3] Patrick Baudisch and Stefanie Mueller. 2017. Personal Fabrication, Foundations and Trends in Human–Computer Interaction: Vol. 10: No. 3–4, pp 165-293. DOI: http://dx.doi.org/10.1561/1100000055

[4] Patrick Baudisch, Arthur Silber, Yannis Kommana, Milan Gruner, Ludwig Wall, Kevin Reuss, Lukas Heilman, Robert Kovacs, Daniel Rechlitz, and Thijs Roumen. 2019. Kyub: a 3D Editor for Modeling Sturdy Laser-Cut Objects. In *2019 CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019),* May 4–9, 2019, Glasgow, Scotland, UK. ACM, New York, NY, USA. DOI: https://doi.org/10.1145/3290605.3300796

[5] Floraine Berthouzoz, Akash Garg, Danny M. Kaufman, Eitan Grinspun, and Maneesh Agrawala. 2013. Parsing sewing patterns into 3D garments. *ACM Trans. Graph.* 32, 4, Article 85 (July 2013), 12 pages. DOI:https://doi.org/10.1145/2461912.2461975

[6] Roberto Bisiani, 1987. Beam search. In Shapiro, S., ed., *Encyclopedia of Artificial Intelligence.* John Wiley and Sons. 56–58.

[7] Yuxin Chen, , Leonidas J. Guibas, and Qi-Xing Huang. "Near-optimal joint object matching via convex relaxation." *arXiv preprint arXiv:1402.1473* (2014).

[8] Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. 2013. 3-Sweep: extracting editable objects from a single photo. *ACM Trans. Graph.* 32, 6, Article 195 (November 2013), 10 pages. DOI: https://doi.org/10.1145/2508363.2508378

[9] James McCrae, Nobuyuki Umetani, and Karan Singh. 2014. FlatFitFab: interactive modeling with planar sections. In *Proceedings of the 27th annual ACM symposium on User interface software and technology (UIST '14).* ACM, New York, NY, USA, 13-22. DOI: https://doi.org/10.1145/2642918.2647388

[10] Christopher B. Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction." In *European conference on computer vision*, pp. 628-644. Springer, Cham, 2016.

[11] Erik D. Demaine, and Martin L. Demaine. "Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity." *Graphs and Combinatorics* 23, no. 1 (2007): 195-208.

[12] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. 2018. InverseCSG: automatic conversion of 3D models to CSG trees. *ACM Trans. Graph.* 37, 6, Article 213 (November 2018), 16 pages. DOI: https://doi.org/10.1145/3272127.3275006

[13] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "Density-based spatial clustering of applications with noise." In *Int. Conf. Knowledge Discovery and Data Mining*, vol. 240, p. 6. 1996.

[14] Christoph M. Flath, Sascha Friesike, Marco Wirth, & Frederic Thiesse, Copy, transform, combine: exploring the remix as a form of innovation. *Journal of Information Technology*, 1-20. DOI: https://doi.org/10.1057/s41265-017-0043-9

[15] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. "A papier-mâché approach to learning 3d surface generation." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 216-224. 2018.

[16] Florian Heller, Jan Thar, Dennis Lewandowski, Mirko Hartmann, Pierre Schoonbrood, Sophy Stönner, Simon Voelker, and Jan Borchers. 2018. CutCAD - An Open-source Tool to Design 3D Objects in 2D. In *Proceedings of the 2018 Designing Interactive Systems Conference (DIS '18).* ACM, New York, NY, USA, 1135-1139. DOI: https://doi.org/10.1145/3196709.3196800

[17] Megan Hofmann, Gabriella Hann, Scott E. Hudson, and Jennifer Mankoff. 2018. Greater than the Sum of its PARTs: Expressing and Reusing Design Intent in 3D Models. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18).* ACM, New York, NY, USA, Paper 301, 12 pages. DOI: https://doi.org/10.1145/3173574.3173875

[18] Nathaniel Hudson, Celena Alcock, and Parmit K. Chilana. 2016. Understanding Newcomers to 3D Printing: Motivations, Workflows, and Barriers of Casual Makers. *In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems(CHI '16).* Association for Computing Machinery, New York, NY, USA, 384–396. DOI: https://doi.org/10.1145/2858036.2858266

[19] Kenny R. Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. 2020. ShapeAssembly: learning to generate programs for 3D shape structure synthesis. *ACM Trans. Graph. 39, 6, Article 234* (December 2020), 20 pages. DOI:https://doi.org/10.1145/3414685.3417812

[20] Ross Knepper, Todd Layton, John Romanishin, and Daniela Rus. "Ikeabot: An autonomous multi-robot coordinated furniture assembly system." In *2013 IEEE International conference on robotics and automation*, pp. 855-862. IEEE, 2013.

[21] Karim Lakhani, and Eric Von Hippel. "How open source software works:“free” user-to-user assistance." In *Produktentwicklung mit virtuellen Communities*, pp. 303-339. Gabler Verlag, 2004.

[22] Minmin Lin, Tianjia Shao, Youyi Zheng, Niloy Jyoti Mitra, and Kun Zhou. "Recovering functional mechanical assemblies from raw scans." *IEEE transactions on visualization and computer graphics* 24, no. 3 (2017): 1354-1367.

[23] Lingjie Liu, Duygu Ceylan, Cheng Lin, Wenping Wang, and Niloy J. Mitra. 2017. Image-based reconstruction of wire art. *ACM Trans. Graph.* 36, 4, Article 63 (July 2017), 11 pages. DOI: https://doi.org/10.1145/3072959.3073682

[24] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. 2006. Partial and approximate symmetry detection for 3D geometry. *ACM Trans. Graph.* 25, 3 (July 2006), 560–568. DOI: https://doi.org/10.1145/1141911.1141924

[25] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy J. Mitra, and Leonidas J. Guibas. "StructureNet: hierarchical graph networks for 3D shape generation." *ACM Transactions on Graphics (TOG)* 38, no. 6 (2019): 1-19.

[26] Georgios Papaioannou, Tobias Schreck, Anthousis Andreadis, Pavlos Mavridis, Robert Gregor, Ivan Sipiran, and Konstantinos Vardis. 2017. From Reassembly to Object Completion: A Complete Systems Pipeline. J. In *Comput. Cult. Herit.* 10, 2, Article 8 (March 2017), 22 pages. DOI: https://doi.org/10.1145/3009905

[27] Kevin J Parker, Saara Marjatta Sofia Totterman, and Jose Tamez Pe. "System and method for 4d reconstruction and visualization." U.S. Patent 6,169,817, issued January 2, 2001.

[28] Thijs Roumen, Yannis Kommana, Ingo Apel, Conrad Lempert, Markus Brand, Erik Brendel, Laurenz Seidel, Lukas Rambold, Carl Goedecken, Pascal Crenzin, Ben Hurdelhey, Muhammad Abdullah and Patrick Baudisch. 2021. Assembler^3: 3D reconstruction of laser-cut models. In *CHI Conference on Human Factors in Computing Systems* (CHI '21), May 8–13, 2021, Yokohama, Japan. ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3411764.3445453

[29] Thijs Roumen, Willi Müller, and Patrick Baudisch. 2018. Grafter: Remixing 3D-Printed Machines. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18).* ACM, New York, NY, USA, Paper 63, 12 pages. DOI: https://doi.org/10.1145/3173574.3173637

[30] Thijs Roumen, Jotaro Shigeyama, Julius Cosmo Romeo Rudolph, Felix Grzelka, and Patrick Baudisch. SpringFit Joints and Mounts that Fabricate on Any Laser Cutter. In *Proceedings of the 32nd annual ACM symposium on User interface software and technology* (UIST '19). ACM, New York, NY, USA, DOI: https://doi.org/10.1145/3332165.3347930

[31] Thijs Roumen, Ingo Apel, Jotaro Shigeyama, Abdullah Muhammad, and Patrick Baudisch. "Kerf-Canceling Mechanisms: Making Laser-Cut Mechanisms Operate Across Different Laser Cutters". *In proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (UIST'20).* ACM. New York, NY, USA, DOI: https://doi.org/10.1145/3379337.3415895

[32] Shuang Shan, Yiting Ma, Chengcheng Tang, and Xuejin Chen. "Folding cartons: Interactive manipulation of cartons from 2D layouts." *Computer Aided Geometric Design* 62 (2018): 228-238.

[33] Dror Sholomon, Omid David, and Nathan S. Netanyahu. "A genetic algorithm-based solver for very large jigsaw puzzles." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1767-1774. 2013.

[34] Peng Song, Chi-Wing Fu, and Daniel Cohen-Or. "Recursive interlocking puzzles." *ACM Transactions on Graphics (TOG)*31, no. 6 (2012): 1-10.

[35] Evgeny Stemasov, Enriko Rukzio and Jan Gugenheimer, "The Road to Ubiquitous Personal Fabrication: Modeling-Free Instead of Increasingly Simple," in *IEEE Pervasive Computing, vol. 20, no. 1, pp. 19-27,* 1 Jan.-March 2021, DOI: https://10.1109/MPRV.2020.3029650

[36] Keke Tang, Peng Song, Xiaofei Wang, Bailin Deng, Chi-Wing Fu, and Ligang Liu. "Computational design of steady 3D dissection puzzles." In *Computer Graphics Forum*, vol. 38, no. 2, pp. 291-303. 2019.

[37] Tom Veuskens Florian Heller, and Raf Ramakers. "CODA: A Design Assistant to Facilitate Specifying Constraints and Parametric Behavior in CAD Models." *In Proceedings of the 47th Graphics Interface Conference on Proceedings of Graphics Interface 2021 (GI'21)*

[38] Ziqi Wang, Peng Song, and Mark Pauly. "DESIA: A general framework for designing interlocking assemblies." *ACM Transactions on Graphics (TOG)* 37, no. 6 (2018): 1-14.

[39] Molly McLure Wasko, and Samer Faraj. "Why should I share? Examining social capital and knowledge contribution in electronic networks of practice." *MIS quarterly* (2005): 35-57. DOI: https://doi.org/10.2307/25148667

[40] Shiqing Xin, Chi-Fu Lai, Chi-Wing Fu, Tien-Tsin Wong, Ying He, and Daniel Cohen-Or. "Making burr puzzles from 3D models." *ACM Transactions on Graphics (TOG)* 30, no. 4 (2011): 1-8.

[41] Long Wen, Xingchen Ye, and Liang Gao. "A new automatic machine learning based hyperparameter optimization for workpiece quality prediction." *Measurement and Control* 53, no. 7-8 (2020): 1088-1098. https://doi.org/10.1177/0020294020932347

[42] Andrew R. Willis, and David B. Cooper. "Bayesian assembly of 3D axially symmetric shapes from fragments." In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1, pp. I-I. IEEE, 2004.

[43] Mingliang Xu, Mingyuan Li, Weiwei Xu, Zhigang Deng, Yin Yang, and Kun Zhou. 2016. Interactive mechanism modeling from multi-view images. *ACM Trans. Graph.* 35, 6, Article 236 (November 2016), 13 pages. DOI: https://doi.org/10.1145/2980179.2982425