Computational LEGO® Technic Design

HAO XU, KA-HEI HUI, and CHI-WING FU, The Chinese University of Hong Kong
HAO ZHANG, Simon Fraser University

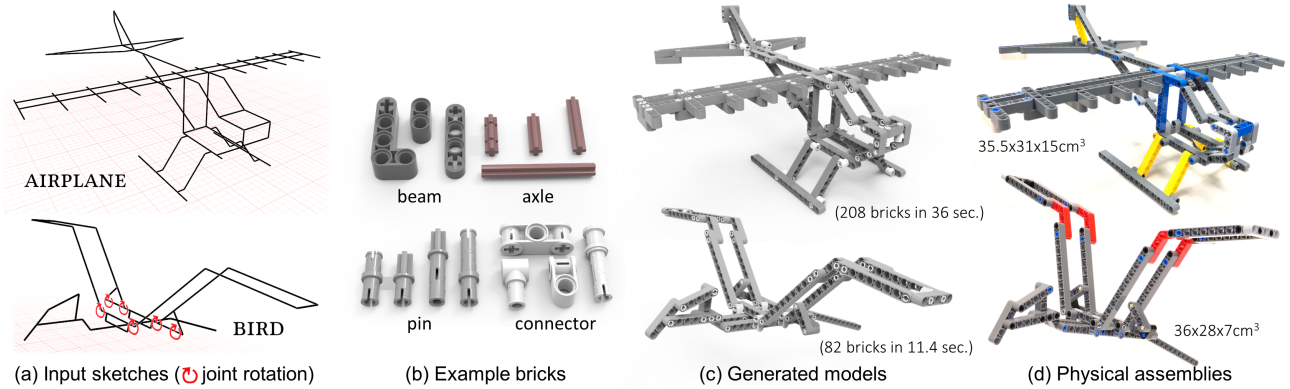


Fig. 1. We introduce a fully automatic method to compute LEGO®Technic designs (c & d) that resemble user input sketches, which may be annotated with joint rotations (bottom-left). The designs are built by coherently-connected LEGO®Technic bricks (b) that respect the symmetry and structural integrity of the models, as well as the dynamics implied at the specified joints, while aiming for minimalistic layouts.

We introduce a method to automatically compute LEGO®[1] Technic models from user input sketches, optionally with motion annotations. The generated models resemble the input sketches with coherently-connected bricks and simple layouts, while respecting the intended symmetry and mechanical properties expressed in the inputs. This complex computational assembly problem involves an immense search space, and a much richer brick set and connection mechanisms than regular LEGO®. To address it, we first comprehensively model the brick properties and connection mechanisms, then formulate the construction requirements into an objective function, accounting for faithfulness to input sketch, model simplicity, and structural integrity. Next, we model the problem as a sketch cover, where we iteratively refine a random initial layout to cover the input sketch, while guided by the objective. At last, we provide a working system to analyze the balance, stress, and assemblability of the generated model. To evaluate our method, we compared it with four baselines and professional designs by a LEGO®expert, demonstrating the superiority of our automatic designs. Also, we recruited several users to try our system, employed it to create models of varying forms and complexities, and physically built most of them.

CCS Concepts: • **Computing methodologies** → **Shape modeling**.

Additional Key Words and Phrases: LEGO®, Technic series, computational design, fabrication, assembly

[1]LEGO®is a trademark of the LEGO®Group, which does not sponsor, authorize or endorse this work.

Authors' addresses: Hao Xu; Ka-Hei Hui; Chi-Wing Fu, The Chinese University of Hong Kong; Hao Zhang, Simon Fraser University.

0730-0301/2019/11-ART196 $15.00
https://doi.org/10.1145/3355089.3356504

ACM Reference Format:
Hao Xu, Ka-Hei Hui, Chi-Wing Fu, and Hao Zhang. 2019. Computational LEGO® Technic Design. *ACM Trans. Graph.* 38, 6, Article 196 (November 2019), 14 pages. https://doi.org/10.1145/3355089.3356504

1 INTRODUCTION

The LEGO®Technic system [The LEGO®Group 2018b] was introduced as an expert series for building advanced 3D models in 1977. With brick pieces well beyond those in regular LEGO®, e.g., beams, axles, pins, and connectors, as shown in Figure 1(b), one can build 3D frame-based structures like those commonly-seen in architecture, as well as mechanical assemblies that exhibit dynamic behavior such as joint rotations. LEGO®Technic designs have resulted in a variety of customizable robotics and mechanical models; see Figure 2 for some elaborative designs by enthusiasts.

Designing LEGO®Technic models is considerably more challenging than regular LEGO®models, even without adding complex mechanical elements such as gears and pulleys. Compared with regular LEGO®bricks, which are mostly connected through studs on top of the bricks, LEGO®Technic bricks are connected in a variety of ways, also via different kinds of pins and connectors, some of which allow joint rotations, as shown in Figures 1(b) and 5. The sheer number of assembly varieties leads to an immense search space. For instance, there are over six billion ways of assembling a simple square with nine-unit long sides. Also, as a result of the connection mechanisms, LEGO®Technic models are often built with stricter and more intricate assembly order than regular LEGO®models. Yet, the Technic system can better adapt to form non-blocky, frame-based, and even articulable shapes in 3D, since the LEGO®Technic beams can be arranged flexibly in different orientations for building different parts in the models; see the green arrows in Figure 2(b) that indicate the orientations of the associated beams.

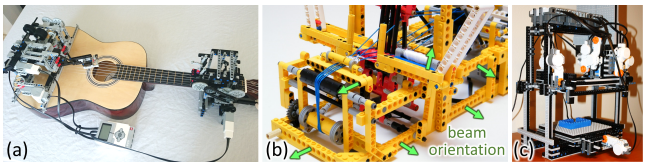


Fig. 2. LEGO®Technic models designed by enthusiasts: (a) a guitar robot by YouTuber TECHNICally Possible (see "https://www.youtube.com/watch?v=cXgB3lIvPHI" with over "20.6M" views); (b) a mechanical loom by N. Lespour (see "https://www.youtube.com/watch?v=TKdUPbtE_xk"); and (c) a 3D LEGO®printer coined MakerLegoBot by W. Gorman (see "http://www.battlebricks.com/makerlegobot/").

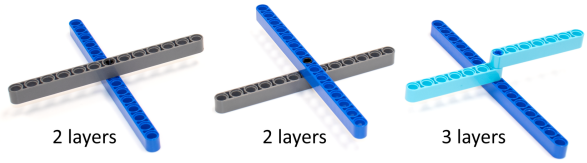


Fig. 3. A cross can be built with different beam layering options.

Due to the connection mechanisms using pins and connectors, Technic models have an entirely different and far more complex building style compared with the simple bottom-up style of regular LEGO®brick assemblies. LEGO®Technic builders have to mindfully plan the *beam placements*, *orientations*, *connections*, and *layering* (see Figure 3). Typically, they need to think several steps ahead and create small assemblies to test the feasibility via trial-and-error [Kmieć 2016; The LEGO®Group 2018b]. Given such complexity, manual designs of Technic models are tedious and challenging, requiring substantial expertise and time to realize a working design.

In this work, we aim to develop a *fully automatic* computational method for LEGO®Technic model construction. Specifically, given a user-drawn line sketch of a frame-based 3D model, optionally with annotations of joint rotations, e.g., see Figure 1(a), our method *automatically selects and arranges LEGO® Technic bricks to form Technic assemblies that resemble and cover the input sketches, while respecting the structural integrity and joint motions of the designs and striving for simplicity and cost-effectiveness of the assemblies.*

The computational challenges of this problem stem from the immense search space and the multitude of different connection mechanisms, beam orientations, and layering options. A viable construction often involves much more than aligning the Technic pieces with the sketched lines. To achieve a coherent, structurally plausible, and functional assembly, the final LEGO®Technic models may deviate in nontrivial ways from the input sketches, e.g., see the computed assembly at the tail of the AIRPLANE in Figure 1.

To approach the problem, we first comprehensively model Technic constructions by enumerating the brick properties and connection mechanisms, conceptualizing the input as a guiding graph (see Figures 4 (a,b)), and modeling the construction requirements into an objective function. Further, we formulate the brick layout problem as a *sketch cover* and solve it by first estimating the local beam orientations. Then, we further iteratively refine a random initial layout into a coherent model to cover the input sketch, while guided by the objective function; see Figures 4 (c,d).

Overall, our work makes the following contributions:

- the first method that automatically constructs LEGO®Technic models that are coherently-connected, assemblable, and functional, based on the user-provided model specifications;
- a computational model for Technic constructions, considering various brick properties and types, connection mechanisms, coherency, and construction requirements, including support for joint rotations, a natural and fundamental mechanical functionality provided by LEGO®Technic designs; and
- finally, a working system, which enables user input designs, provides structural integrity analysis, and produces assembly instructions and visualizations of the assembly process to facilitate physical constructions (see Figures 4 (e,f,g)).

We demonstrate our method by generating Technic models of varying forms and complexities, and building physical assemblies for most of them. We evaluate our method in various aspects, including a comparison with four baselines and with designs created by a LEGO®expert. Our method took only 36 sec. to compute the AIRPLANE model shown in Figure 1, while more complex models can be realized in just minutes. In contrast, it took the human expert close to 1.5 hours to design a comparable model for AIRPLANE using existing software. Finally, we show how our method consumes motion annotations and produces dynamic Technic models that exhibit hinge-style rotations and embed gear systems; see Section 6.

2 RELATED WORKS

LEGO® construction. The first work aimed at automatic construction of LEGO®models using regular bricks was by Gower et al. [1998]. The problem was formulated as a combinatorial optimization to maximize a goodness measure for LEGO®structures. Some follow-up works include Petrovič [2001] using evolutionary algorithms, Winkler [2005] using beam search, Testuz et al. [2013] using a graph-based algorithm, Stephenson [2016] using a multi-phase approach, and Lee et al. [2018] using a genetic algorithm. Considering not only the target shape, Luo et al. [2015] developed a comprehensive method for buildable LEGO®structures in larger scales, considering the brick colors in the assemblies and the structural stability. Later, Yun et al. [2017] improved LEGO®constructions by silhouette fitting. See [Kim et al. 2014] for a survey on LEGO®layout methods.

Besides regular brick models, Lambrecht [2006] computed LEGO®assemblies with oriented thin plates. Waßmann and Weicker [2012] devised a two-phase approach for stability analysis by solving a max-flow network. Kuo et al. [2015] computed brick sculptures from pixel arts by considering visual quality and stability.

So far, existing computational works have focused on regular LEGO®bricks. In comparison, LEGO®Technic designs involve a significantly more complex brick set and entirely different assembly mechanisms using a variety of pins and connectors; they even support dynamic functionalities, such as joint rotations, that our current method is able to realize fully automatically. On the other hand, while there are a number of commodity software tools to aid users to design LEGO®models, e.g., LEGO®Digital designer [The LEGO®Group 2018a], MLCad[2018], and LDview [2018], such tools only provide basic modeling and rendering for users to create LEGO®designs via simple drag-and-drop. We are not aware of any advanced computational support for designing LEGO®Technic models.

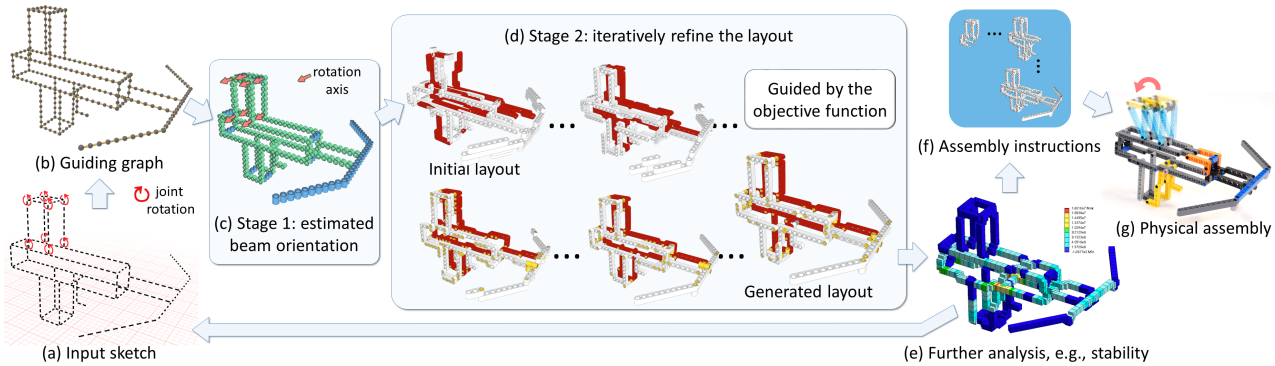


Fig. 4. Overview of our approach. First, we construct a guiding graph (b) to abstract the user-input sketch (a), which has optional annotations for joint rotations. Then, we estimate the local orientation of beams (c), and iteratively refine a random initial layout to cover the input sketch, guided by the objective function and constrained by the specified joint rotations (d). Further, we analyze the stability, balance, and assemblability of the generated model (e), and produce assembly instructions (f) to facilitate physical construction (g). Upon the analysis, we may go back and revise the input sketch: (e) → (a).

Assembly-based fabrication. Lau et al. [2011] took a 3D man-made object as inputs, and generated parts and connectors for building the object. Thomaszewski et al. [2014] designed motorized assemblies of linkage-based characters. Schulz et al. [2014] proposed a data-driven method to design 3D models. Cignoni et al. [2014] generated ribbon-shaped interlocking planar slices for assembling complex 3D shapes. Skouras et al. [2015] designed assemblies made up of interlocking quadrilateral elements. Song et al. [2016] built 3D-printed objects by collectively using 3D-printed and laser-cut parts. Recently, Yao et al. [2017] created an interactive tool for designing decorative joints, while Geilinger et al. [2018] presented a design tool for robots that move using arbitrary arrangements of legs and wheels.

In architecture modeling, Deuss et al. [2014] minimized the assembly work for building masonry structures; Yoshida et al. [2015] devised a method for building architecture-scale models formed by glued chopsticks; Pietroni et al. [2017] designed a computational framework for tensegrity structures; and Desai et al. [2018] developed a computational design system for electromechanical devices.

Most previous works from this domain decompose target shapes into customized parts to facilitate 3D fabrication. In contrast, our goal is to reconstruct a 3D shape using a diverse but fixed (thus not customized) brick set with a rich variety of brick connections, to approximate the target shape specifications geometrically, structurally, and in terms of dynamics, while ensuring coherence of the brick connections. Our problem is that of a *multi-tiling* rather than shape decomposition. Instead of accounting for various constraints related to 3D constructions, we must meet complex and multifaceted objectives, including simplicity, symmetry, rigidity, etc., to support the Technic constructions. We are not aware of any previous computational assembly works that consider an assembly of multiple brick types with diverse brick connections in 3D.

Application of LEGO® bricks. Lastly, we discuss applications that use LEGO® bricks as off-the-shelf building elements. In earlier works, Mitra and Pauly [2009] used LEGO® bricks to build shadow art sculptures and Baronti et al. [2010] considered LEGO® structures composed of regular bricks as markers for camera calibration. Song et al. [2012] used LEGO® bricks with flat tiles to build polycube-shaped interlocking puzzle pieces, while Mueller et al. [2014] developed the faBrickator system for rapidly prototyping functional objects by substituting parts of the 3D objects with LEGO® brick assemblies. Most recently, Chen et al. [2018] proposed to fabricate 3D objects with 3D-printable pyramidal parts to form the outer shells and universal blocks (or LEGO® bricks) to build the internal cores.

3 OVERVIEW

Input and output. We provide a GUI tool for sketching line segments to craft 3D LEGO® Technic designs, where the line segments are constrained to have integer lengths to match Technic bricks and nearby line segment endpoints are automatically snapped to join. Also, our tool shows semi-transparent guiding planes aligned with the principal axes for sketching coplanar lines, since Technic bricks mostly lay on the principal (xy, yz, and zx) planes; see Figure 2(b). Optionally, the user may provide motion annotations to specify *hinge-style rotations* at joints and to indicate embedded dynamic parts in the sketch designs. The output of our tool includes a LEGO® Technic design composed of Technic bricks, and assembly instructions and visualizations for the assembly process.

Objectives. Our method aims to optimize a combination of the following three objectives for the generated LEGO® Technic designs:

- *Faithfulness to input line sketch.* First, the output designs should resemble the input sketches and respect the intended symmetry and motion specifications provided by the user.
- *Structural integrity.* Second, the output should be coherently-connected and assemblable, while striving for rigidity.
- *Simplicity and efficiency.* Third, we strive for an output design that is both simple and cost-effective for assembly.

Challenges. The automatic generation of LEGO® Technic models involves four sub-problems: (i) which bricks to use and where to put them; this is analogous to the set cover problem, since the bricks should cover the input design; (ii) beam/brick orientation in the layout; (iii) layering (see Figure 3); and (iv) beam/brick connections. These four sub-problems are closely coupled, thus complicating the algorithm design. Also, we have to attentively consider the objectives during the construction process, while efficiently exploring

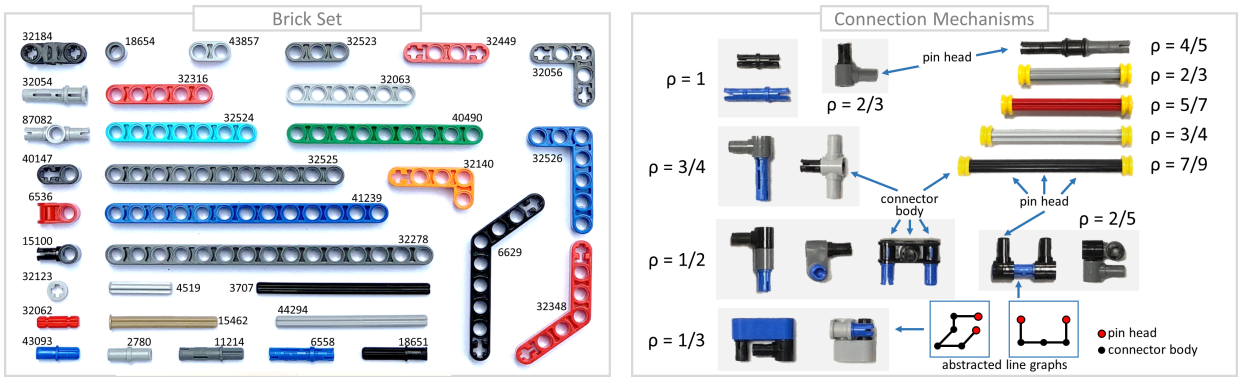


Fig. 5. LEGO®Technic bricks with their unique brick IDs (left) and connection mechanisms (right) supported in our system. The connection mechanisms are grouped by the pin head ratio (ρ), which is defined for quantifying the simplicity of connection mechanisms. In addition, we show as examples the abstracted line graphs for two of the connection mechanisms. See Supplementary material part A for more details on the brick set and connection mechanisms.

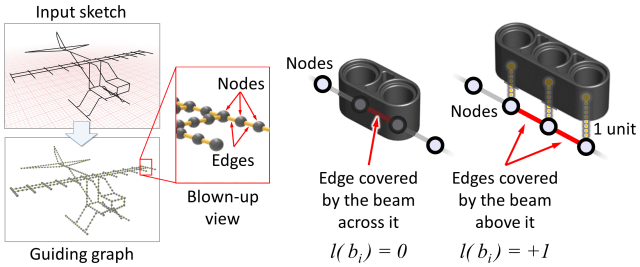


Fig. 6. An example guiding graph (left) and edge cover (right).

the immense search space for finding the optimal solution, i.e., a layout of bricks that form the input design.

Our approach. First, we study existing Technic models [Isogawa 2010; Kmieć 2016], and enumerate the brick properties and connection mechanisms. Then, we abstract each input as a *guiding graph* and each brick as a line graph to facilitate computation (see Figures 4 (a) & (b)). Next, we formulate an objective to meet the various construction requirements, develop the layout modification operator to locally update brick layouts, and design a beam connection procedure to join adjacent beams. Further, we formulate the layout search as a sketch cover to automate Technic constructions:

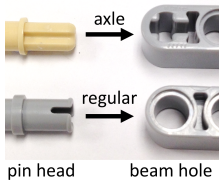
- For simple connections and structural integrity, Technic beams in local structures often have the same orientation. Hence, we first estimate the local orientation of beams over the design for sub-problem (ii) in the first stage; see Figure 4(c).
- Next, in the second stage, we optimize mainly for sub-problems (i), (iii) and (iv) altogether by first initializing a random layout, then formulating an iterative procedure to refine the layout and connect beams, guided by the objectives; see Figure 4(d).

4 MODELING LEGO®TECHNIC CONSTRUCTION

4.1 Technic Bricks

Brick set. Figure 5 (left) shows the bricks supported by our system. For each brick, our system stores its 3D mesh model, physical properties such as weight, and connecting locations on the brick, i.e., *holes* on beams and connectors, and *pin heads* on pins and axles; see Figure 5 (right) for examples. Also, we abstract the structure of each brick as a simple *line graph* in 3D. For example, the green beam in Figure 5 (left) is abstracted as a graph of nine nodes (holes) and eight one-unit-long edges between adjacent nodes.

Brick connections. Next, we enumerate the beam connection mechanisms by studying the connection mechanisms in existing Technic models, where pin heads are locations that connect to beam holes, and connector bodies are non-pin-head locations in the connection mechanisms; see the labels in Figure 5 (right). Also, we abstract each mechanism as a line graph in 3D; see Figure 5 (right) for two examples. We consider two types of pin heads, i.e., *axle* and *regular*, for connecting respective types of beam holes; see the right inset figure. The two types tradeoff between connection rigidity and flexibility: the axle pin heads enforce rigid non-rotatable connections that must be perpendicular/parallel, while the regular pin heads allow rotatable connections.



Pin-head ratio, ρ. In LEGO®Technic, simpler connection mechanisms dominated by pin heads (not connector bodies) are preferred, since they help enhance the structural integrity. To quantify the simplicity of connection mechanisms, we define ρ as the pin head count over the total node count in a mechanism's line graph, where simpler mechanisms have larger ρ values; see Figure 5 (right).

Layer number, $l(b_i)$. Technic beams of the same orientation are connected in layers, above or below one another; see Figure 3. Given a beam, say b_i, we define its *layer number* in a LEGO®Technic construction as $l(b_i)$, where $l(b_i)$ is zero, if b_i exactly goes through the associated sketch line in the input, and $l(b_i)$ is positive/negative, if b_i is above/below the associated sketch line; see Figure 6 (right).

4.2 The Sketch Cover problem

Guiding graph. For efficient computation, we abstract the input sketch as a *guiding graph* (denoted as $\mathcal{G}$), where nodes are distributed along the sketch line segments with adjacent nodes being one unit apart, like holes on Technic beams. Moreover, edges in $\mathcal{G}$ are all one unit long for connecting the adjacent nodes; see Figure 6 (left).

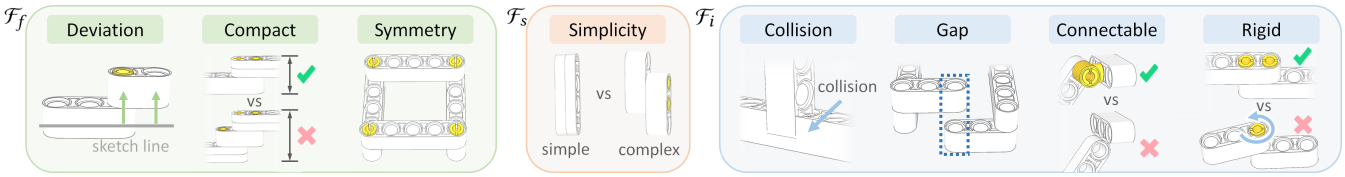


Fig. 7. Component terms in the objective function, from left to right: faithfulness to input sketch ($\mathcal{F}_f$), model simplicity ($\mathcal{F}_s$), and structural integrity ($\mathcal{F}_i$).

Sketch cover. An edge in $\mathcal{G}$ is said to be *covered* by a brick (e.g., beam) if part of the brick is parallel to it, either crossing it or locating at a small distance from the edge; see Figure 6 (right). If all edges in $\mathcal{G}$ are covered by some bricks in a generated Technic model, the model is said to *fully cover* the input sketch. The *sketch cover problem* is to find a brick layout that fully covers the given input design.

Symmetric groups. Our system considers reflection and translational symmetry. We group line segments (i.e., subgraphs in guiding graph) and mark symmetric groups in the input sketch. In our current implementation, this is done manually, but in the future, we plan to incorporate symmetry detection methods for the task.

4.3 Objective function

We formulate and minimize the following objective function to guide the finding of a beam layout (say $\mathcal{B} = \{b_i\}$) that covers the input sketch, while conforming to the various construction requirements:

$$\mathcal{F} \;=\; \mathcal{F}_f + \mathcal{F}_s + \mathcal{F}_i \,,$$

where $\mathcal{F}_f$, $\mathcal{F}_s$, and $\mathcal{F}_i$ are component terms in the objective (see the corresponding illustrations shown in Figure 7):

(i) Faithfulness to input sketch, $\mathcal{F}_f$. We define

$$\mathcal{F}_f \;=\; w_{\text{dev}}\mathcal{F}_{\text{dev}} + w_{\text{cpt}}\mathcal{F}_{\text{cpt}} + w_{\text{sym}}\mathcal{F}_{\text{sym}} \,,$$

where w_{dev}, w_{cpt}, and w_{sym} are weights, and we have:

- $\mathcal{F}_{\text{dev}}$ minimizes the distance deviation (i.e., layer number $l(b_i)$) of the beams in the layout from the input sketch:

$$\mathcal{F}_{\text{dev}} \;=\; \sqrt{\frac{1}{\sum_{b_i\in\mathcal{B}} L(b_i)} \sum_{b_i\in\mathcal{B}} L(b_i)\, l(b_i)^2} \,,$$

  where $L(b_i)$ denotes the length (number of holes) of beam b_i.
- $\mathcal{F}_{\text{cpt}}$ compacts the layering by minimizing the range of $l(b_i)$ in each coplanar component (see Section 5.1) in the layout:

$$\mathcal{F}_{\text{cpt}} \;=\; \max_{C_j\in C} \Big[\max_{b_i\in C_j} l(b_i) \;-\; \min_{b_i\in C_j} l(b_i) \Big]^2 \,,$$

  where $C = \{C_j\}$ is a set of coplanar components extracted from the input design at the end of the stage one of our framework, and each C_j is a subset of beams in $\mathcal{B}$; see Section 5.1 for how we extract C, the set of coplanar components.
- $\mathcal{F}_{\text{sym}}$ minimizes the deviation from symmetry for each pair of symmetric beam groups:

$$\mathcal{F}_{\text{sym}} \;=\; \frac{1}{|\mathcal{S}|} \sum_{(\mathcal{B}_i,\mathcal{B}_j,T_{ji})\in\mathcal{S}} \frac{1}{\sum_{b\in\mathcal{B}_i} L(b)}\, d_H(\mathcal{B}_i, T_{ji}(\mathcal{B}_j))^2 \,,$$

  where $\mathcal{B}_i \subset \mathcal{B}$ and $\mathcal{B}_j \subset \mathcal{B}$ are symmetric groups; T_{ji} is the 3D symmetric transformation to bring $\mathcal{B}_j$ to $\mathcal{B}_i$; $\mathcal{S}$ is a set of symmetric groups; and d_H denotes the Hausdorff distance. In the case of self (reflection) symmetry, we can have $\mathcal{B}_i = \mathcal{B}_j$.

(ii) Model simplicity, $\mathcal{F}_s$. We encourage *layout simplicity* by minimizing the total beams length and maximizing the total pin head ratios of the connection mechanisms in the layout;

$$\mathcal{F}_s \;=\; w_{\text{tbl}} \frac{\sum_{b_i\in\mathcal{B}} L(b_i)}{|\mathcal{V}|} + w_{\text{phr}}(1-\bar{\rho}) \,,$$

where $\mathcal{V}$ is the set of nodes in guiding graph $\mathcal{G}$; $\bar{\rho}$ is the *average pin-head ratio* (phr) over all the connections in the LEGO®Technic model; and w_{tbl} and w_{phr} are weights.

(iii) Structural integrity, $\mathcal{F}_i$. We define:

$$\mathcal{F}_i \;=\; w_{\text{col}}\mathcal{F}_{\text{col}} + w_{\text{gap}}\mathcal{F}_{\text{gap}} + w_{\text{coh}}\mathcal{F}_{\text{coh}} + w_{\text{rgd}}\mathcal{F}_{\text{rgd}} \,,$$

where w_{col}, w_{gap}, w_{coh}, and w_{rgd} are weights, and we have:

- $\mathcal{F}_{\text{col}}$ measures the total number of collisions (N_{col}) between beams in layout $\mathcal{B}$ and normalizes it by the total beam length:

$$\mathcal{F}_{\text{col}} \;=\; \frac{1}{\sum_{b_i\in\mathcal{B}} L(b_i)}\, N_{\text{col}}(\mathcal{B}) \,.$$

- $\mathcal{F}_{\text{gap}}$ measures the total number of gaps between beams. We denote $\mathcal{B}_v$ as the set of beams that associate with vertex $v \in \mathcal{V}$. Then, we detect the gap at v by

$$\text{gap}(v) \;=\; \Big[\max_{b_i\in\mathcal{B}_v} l(b_i) - \min_{b_i\in\mathcal{B}_v} l(b_i) \Big] + 1 - |\mathcal{B}_v| \,,$$

  and define $\mathcal{F}_{\text{gap}} \;=\; \frac{1}{|\mathcal{V}|} \sum_{v\in\mathcal{V}} \text{gap}(v)$. Although we may fill a gap using a single hole brick (ID 18654 in Figure 5), but we still need to reduce gaps for structural integrity.
- $\mathcal{F}_{\text{coh}}$ measures the connection coherence (connect-ability) between beams in $\mathcal{B}$. To start, we first connect adjacent beams in $\mathcal{B}$ by trying various connection mechanisms (see Section 5.2), and count the number of failure connections (N_{cfail}). We define $\mathcal{F}_{\text{coh}} \;=\; \frac{N_{\text{cfail}}}{|\mathcal{E}|}$, where $\mathcal{E}$ is the edge set in guiding graph. Note that if we fail to connect two adjacent beams, the layering and/or orientation of the beam(s) will be modified in the iterative refinement process; see Section 5.3 for detail.
- $\mathcal{F}_{\text{rgd}}$ measures the number of rigid beam subsets in the layout. In general, a local beam connection can be rigid or non-rigid; see Figure 7 (right). Here, we perform a depth first traversal over the guiding graph to examine the connections between adjacent beams, and stop the traversal at non-rigid connections. As a result, we can decompose $\mathcal{B}$ into "rigid" subsets, such that all the beams inside are rigidly connected transitively. In this way, we can evaluate

$$\mathcal{F}_{\text{rgd}}(\mathcal{B}) \;=\; \frac{\text{number of rigid subsets in } \mathcal{B}}{\sum_{v\in\mathcal{V}} (\deg(v)-1)^2} \,,$$

  where $\deg(v)$ is the valence of vertex v in $\mathcal{G}$. Note that we also tried to normalize the term by using the number of beams in $\mathcal{B}$ instead, but we find the above formulation to be more

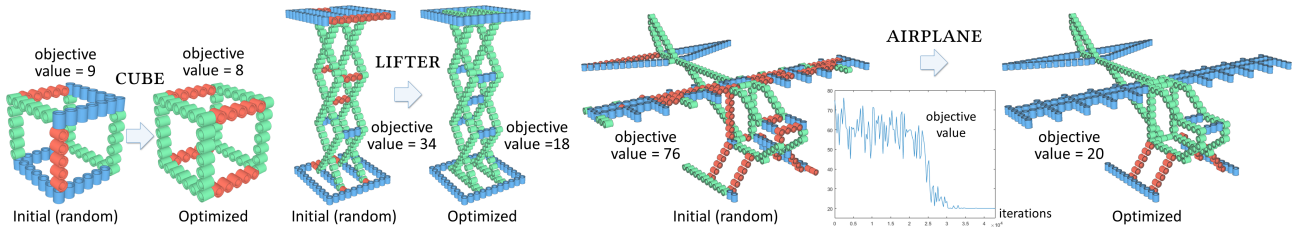


Fig. 8. Optimizing the "beam hole" orientations, such that most adjacent beam holes have the same orientation for simpler beam connections.

scalable to various input designs, since it measures not only the scale but also the topological complexity.

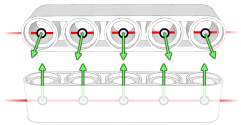
To balance the components in the objective, we empirically set the associated weights as follows: w_{cpt}, w_{sym}, w_{phr}, w_{col}, w_{gap}, and w_{coh} are set as 1, 30, 10, 30, 100, and 50, respectively. On the other hand, w_{dev}, w_{tbl}, and w_{rgd} are set based on user preference. In practice, we set $\{w_{\text{dev}}, w_{\text{tbl}}, w_{\text{rgd}}\}$ as $\{100, 0, 0\}$ to aim for high faithfulness to the input sketch, as $\{0, 100, 0\}$ to aim for model simplicity, and as $\{100, 0, 100\}$ or $\{0, 100, 100\}$ to additionally aim for rigid connections. Figure 22 and Table 2 show experiments on their effectiveness.

5 LEGO®TECHNIC CONSTRUCTION SEARCH

In our initial attempts, we tried a greedy approach that progressively arranges locally-optimum beams to cover the sketch. Further, we tried several other approaches (see method comparisons in Section 6) to improve the search, but the results produced from these approaches are poorly optimized, especially for nontrivial inputs. To address the immense search space (combinations of beam placements, orientations, connections, and layering) in Technic constructions, the search has to be efficient and allow updates that iteratively propagate over the layout. Hence, we design a two-stage approach (see Figures 4(c) & (d)) that first estimates the beam orientation then iteratively refines the layout to optimize the objectives.

5.1 Stage one: Estimate Beam Orientation

In LEGO®Technic models, adjacent beams of same orientation can be steadily connected by pins. However, to build 3D models, we generally need to arrange beams in different orientations for building different parts of the models. Since the beam orientation strongly affects the overall structure, connections and joint rotations, we first estimate the local beam orientation over the guiding graph.



Mathematically, we represent a beam's orientation as a 3D vector that passes through the medial axis of the beam's holes; see the inset figure. In general, we can reorient a beam, as long as its orientation vector is perpendicular to either its corresponding line segment in the sketch, or the edges that it covers in the guiding graph. Here, we model the problem of finding the beam orientation as an assignment problem. Since most nodes in guiding graph $\mathcal{G}$ will eventually be covered by beams in the generated LEGO®Technic model (the rest will be covered by the connection mechanisms), we create a *hole orientation variable* for each node in $\mathcal{G}$. Since LEGO®Technic bricks mostly lay on the principal (xy, yz and zx) planes, we should assign

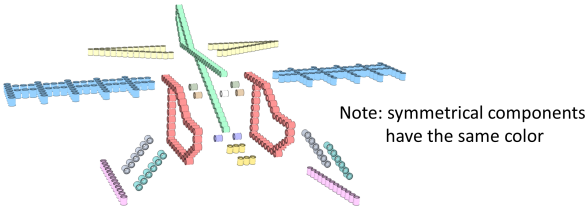


Fig. 9. The optimized AIRPLANE in Figure 8 has 22 components.

to each variable a principal direction (X, Y or Z), unless purposely specified in the user interface.

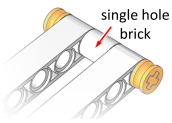
Goal. To *minimize the number of adjacent node pairs with different hole orientations*, such that we can encourage the use of pin and axle connections for model simplicity and structural integrity.

Constraints. (i) For non-junction nodes along line segments in the input sketch, their hole orientations should be perpendicular to the associated line segment [1]; (ii) orientation of adjacent holes should be the same or perpendicular to each other, so that we may use a single beam to cover the two holes or connect them using a connection mechanism shown in Figure 5 (right); (iii) at each joint annotated to allow rotation, the orientation of the associated hole must align with the joint's rotational axis; and (iv) hole orientation variables at symmetry locations are constrained to be the same (for translational symmetry) or mirrored (for reflection symmetry).

Method. We solve this combinatorial optimization problem using a simulated annealing model [Kirkpatrick et al. 1983]. Initially, we randomize all hole orientation variables (see Figure 8 for examples) but following the listed constraints. Then, we iteratively choose a random line segment, change the orientation of all the non-junction nodes in the segment, and update the orientation at each associated junction node, if the objective is minimized. Figure 8 shows the initial state, optimized result, and objective values for the three examples. In our implementation, we set the initial temperature as 2×10^3, stopping temperature as 0.01, and cooling rate as $1 - \frac{10}{|\mathcal{V}|}$, where $\mathcal{V}$ is the node set in the guiding graph. The optimization completes in only a few seconds for most models; see Section 6.

Find "components" and "beam placements". Next, we decompose nodes in guiding graph into *connected* and *coplanar* components, where the nodes in each component have the *same orientation*. For

[1] Except for short line segments with just a single interior hole (see the right inset figure), since we observe that in such a situation, existing LEGO®models may fill/cover the node using a single hole brick (near Figure 5 (top-left)) whose orientation aligns with the associated line segment.



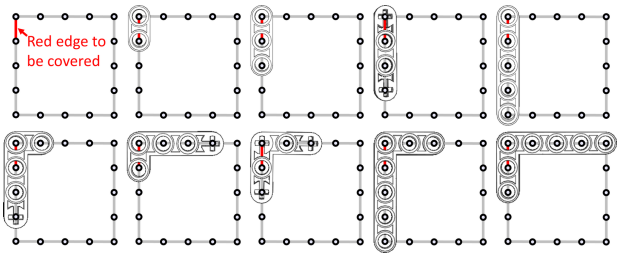


Fig. 10. Nine possible beam placements to cover the red edge (see top left).

example, the optimized CUBE and LIFTER shown in Figure 8 have six and nine components, respectively, while the optimized AIRPLANE has 22 components; see Figure 9. Also, we store symmetry information within and between components to facilitate later computation.

Furthermore, on each component, we find all feasible beam placements on the component, but ignoring the beam placements that pass through the annotated rotating joints. Hence, for each edge in a component, we keep *a list of feasible beam placements* that can cover the edge (see Figure 10 for an example), and also, a list of beam placements that stop at or pass through each node. Using this data structure, we can efficiently arrange beams to cover any edge in the guiding graph, and accelerate the layout generation in the second stage (layout modification operator) of our method.

5.2 Key components in Stage Two

Layout modification operator. There are two key components in the second stage of our method. The first one is an operator to modify a given beam layout. We design this operator with the following considerations: (i) the operator must be *efficient*, due to its heavy usage in the search process; (ii) even if the operator is local, successively applying it should produce *diverse beam layouts*; and (iii) it should *avoid obviously bad beam placements*.

Figure 11 shows the operator procedure. After randomly picking a covered edge in the guiding graph, we remove all the beams that stop at or pass through the edge, and locate all possible beam placements that can cover the resulting "uncovered" edges in the guiding graph. To promote layout diversity, we next calculate a selection probability for each possible beam placement, where each beam candidate is selected based on the number of uncovered edges that it can cover; see examples in Figure 11. Next, based on the probabilities, we randomly select a candidate beam placement to add to the layout, at a layer that produces more compact layering. Hence, we may try both long and short beams in the search, while avoiding meaningless beams that cannot cover any edge and encouraging model simplicity and faithfulness to the input sketch. We repeat this select-and-add process (usually a few times) until we cover all the uncovered edges. This procedure was carefully designed after experimenting with several alternatives; see Supplementary material part C.

Beam connection procedure. Another key component is the procedure to connect adjacent beams, such that we can join the beams and form a connected assembly in the end. Procedure-wise, given a layout of beams, we first identify edges in the guiding graph that are not covered by any beam, and locally group the adjacent uncovered edges; see the examples shown in Figure 13 (leftmost column). Then,

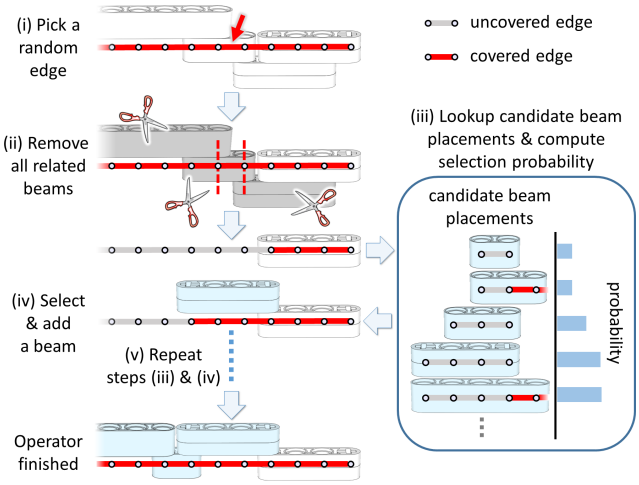


Fig. 11. The layout modification operator efficiently modifies a layout by locally removing beams around a random edge and adding new beams.

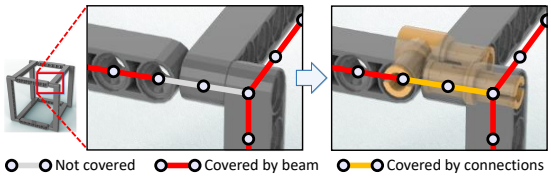


Fig. 12. After we arrange the beams (left), we need to further arrange connection mechanisms (in orange) between adjacent beams (right).

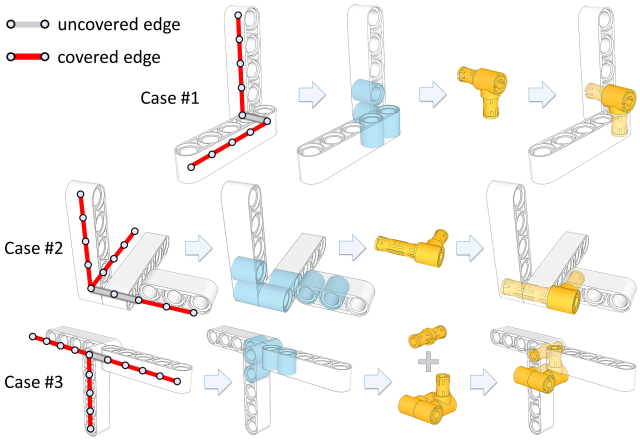


Fig. 13. Procedure: find connection mechanisms to join adjacent beams.

we identify the beam holes around each group based on a distance threshold of $\sqrt{3}$ units; see the highlighted holes shown in Figure 13 (middle column). To connect the beams in each group, we first find *all feasible connection mechanisms* that can join holes of different beams, then find *subsets of them* (see Figure 13 (rightmost column)) that satisfy the following considerations: (i) the subset of mechanisms should together connect all the different beams around the uncovered edges; (ii) the chosen mechanisms should not collide with the existing beams and also one another; (iii) we aim for maximal

pin-head ratio for model simplicity; (iv) connecting bricks that extend outward should not go below the ground plane or collide with any embedded dynamic element; and (v) the chosen mechanisms should not interfere any annotated joint rotations.

In general, beam connections are not always one-to-one, i.e., see the case in Figure 13 (top). First, some mechanisms can join three or more beams together, e.g., the L-shaped mechanism in Figure 13 (middle) and the T-shaped one with $\rho = 3/4$ in Figure 5 (right). Second, we sometimes need more than one connection mechanisms to join the beams around a group of uncovered edges, e.g., for the cases shown in Figure 13 (bottom) and Figure 12, we need two mechanisms to connect the beams. Furthermore, in case feasible connection mechanisms cannot be found, the connectable term in the objective (see Figure 7) will reflect the result, so that the search framework will be guided to modify the layout accordingly.

5.3 Stage two: Iterative Layout Refinement

Overall, our solution search in Stage two starts by initializing a random layout of beams, then iteratively modifies it to improve the objective function; see Figure 4(d). Particularly, the layout starts without beam connections, so we have to find appropriate connection mechanisms to join the beams during the search. We have tried various optimization frameworks to optimize the objective function (see Figure 19), and in the end, adopted a simulated annealing model proposed by Cagan et al. [1998] to regulate the solution search.

Layout initialization. We create the initial layout by repeating the first two steps in the layout modification operator, i.e., randomly pick an uncovered edge in the guiding graph and select a feasible beam placement to add into the initial layout. However, we deliberately select beam placements with equal probability to generate a more random initial layout (see Figure 4(d) for an example), since a more random layout helps the annealing process avoid local minima.

Overall procedure. Algorithm 1 outlines the search procedure. There are four input parameters, $T_{\max}$, T_{mid}, $T_{\min}$ and r, which denote the starting temperature, middle cutoff temperature, ending temperature, and cooling rate, respectively. We empirically set $T_{\max}$ as $2{\times}10^3$, T_{mid} as 10, $T_{\min}$ as 10^{-4}, and r as 0.999 for simple models and as 0.99997 for large complex models to trade off model quality and running time. In the early annealing process, the layout is highly random and not stable, so we guide the layout refinement by minimizing a simplified version of the objective function (denoted as $\mathcal{F}^0$) without evaluating the collision and connectable terms for computational efficiency, then switching to the full version objective ($\mathcal{F}$) when the layout becomes stable. Also, since the layout modification operator is local, we actually update the objective function value based on the local changes in the layout. This can boost the computational efficiency for evaluating the objective function.

Discussion. The beam orientations estimated in Stage one may not always be perfect; we further allow the layout modification operator to try different valid beam orientations (see the constraints in Section 5.1), when the layout becomes stable. See Figure 8 (right) for the estimated orientation of the pontoon beams on the bottom of AIRPLANE; re-orientating them allows simpler connections (see Figure 1) that minimize the objective. Besides the exponential annealing schedule we adopted in the search, we have tried other schedules:

ALGORITHM 1: Overall procedure for iterative layout refinement

Data: $T_{\max}$, $T_{\min}$, T_{mid}, r, and Guiding graph $\mathcal{G}$
$\mathcal{B}_{\text{current}}$ = initialize_layout($\mathcal{G}$) // initialize the beam layout
T = $T_{\max}$ // initialize temperature T
$\mathcal{B}_{\text{best}}$ = $\mathcal{B}_{\text{current}}$ // initialize the best layout
$\mathcal{F}_{\text{obj}}$ = $\mathcal{F}^o$ // use approx. obj. func.
while $T > T_{min}$ **do**
  if $T < T_{mid}$ **then**
    $\mathcal{F}_{\text{obj}}$ = $\mathcal{F}$ // switch to full obj. func.
  end
  $\mathcal{B}_{\text{new}}$ = modify($\mathcal{B}_{\text{current}}$) // layout modification op.
  Δ = $\mathcal{F}_{\text{obj}}(\mathcal{B}_{\text{new}}) - \mathcal{F}_{\text{obj}}(\mathcal{B}_{\text{current}})$
  if $\exp(-\Delta/T) > rand(0,1)$ **then**
    $\mathcal{B}_{\text{current}}$ = $\mathcal{B}_{\text{new}}$ // accept the change
    if $\mathcal{F}_{obj}(\mathcal{B}_{current}) > \mathcal{F}_{obj}(\mathcal{B}_{best})$ **then**
      $\mathcal{B}_{\text{best}}$ = $\mathcal{B}_{\text{current}}$ // update the best layout
    end
  end
  $T = T * r$ // update temperature T
end
return $\mathcal{B}_{\text{best}}$

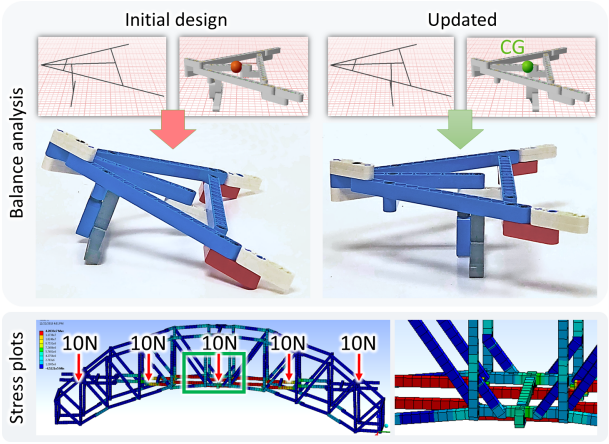


Fig. 14. Example balance analysis (top) and stress plots (bottom).

linear, logarithmic, optimum [Nourani and Andresen 1998], and thermodynamic [De Vicente et al. 2003]. However, we found no obvious improvements in results and running time. This is likely because the search space is discrete rather than continuous, where the feasible solutions are far from one another. Also, we have tried to extend Algorithm 1 to be a population-based search [Van Hentenryck and Vergados 2007] by finding N instead of one solutions in each iteration and keeping the best K for generating candidates in the next iteration (we set $N = 20$ and $K = 4$). However, the solution quality improves only slightly but the running time increases substantially, so we kept $N = K = 1$ when producing our results.

5.4 Model Analysis and Assembly

Our tool provides further analysis on the generated Technic model: (i) self-balancing — check if the model's center of gravity is well-supported [McGhee and Frank 1968; Prévost et al. 2013; Schneider

Fig. 15. A photograph showing the physical assemblies of most LEGO®Technic models generated by our method. From left to right, we have PRINT_BOX, AIRPLANE, LONG_BOW, CUBE, HOUSE, CROSSBOW, BIRD, BRIDGE, FLYING_KITE, CLAW, PICKER, TABLE_LAMP, LIFTER, CHAIR_FRONT, CHAIR_SIDE, LITTLE_FERRIS, GLASSES, SEASAW, BICYCLE, and ROBOT; see Table 1 for the statistics (number of input sketch lines, bricks, etc.) about these generated models.

and Eberly 2002] (see Figure 14 (top)); (ii) a visualization of the stress distribution — script the model as input to the ANSYS R19.0 (Academic) software (see Figure 14 (bottom)); and (iii) assemblability — test if all bricks can be iteratively removed from the assembly without collision. Based on the analysis, we can also generate an assembly sequence that respects the model symmetry, produce scripts to render a model assembly video, and generate LEGO®-style assembly instructions with the help of the LPub3D software [Sandy 2018]. See Supplementary material part D for implementation details.

6 RESULTS

We employed our method to design and generate a rich variety of LEGO®Technic models, as listed in Table 1. All models are automatically generated from input sketches on a MacBook Pro with a dual-core Intel i5 CPU and 8GB RAM. Figure 15 photographs the physical assemblies of twenty of the computed models, while Figure 17 shows the renderings of four remaining larger ones. Nine of them can perform motion dynamics, e.g., BIRD, CLAW, and LONG_BOW. These results demonstrate that our method is able to generate LEGO®Technic models of varying size, shape, structure, and functionality, from small models, such as PICKER, FLYING_KITE and LIFTER, with less than 100 bricks, to medium-sized models, such as LONG_BOW, CLAW, and AIRPLANE, as well as to large models with over 400 bricks, such as CASTLE and PRINT_BOX. Particularly, the results show coherent connections between bricks and the preserved symmetry. In terms of shape and structure, our method can generate large planar and nearly-planar models like TOWER_2D and CASTLE, as well as 3D structures of varying complexity. Please refer to the supplementary video for the input sketches and animated results.

Timing performance. The rightmost part in Table 1 reports the running times of our method. Stage one takes only a few seconds to complete, except for a few very large models like SPACE_STATION and TOWER_3D. As expected, Stage two takes up most of the processing time, since it needs to iteratively refine the layout and brick

Table 1. Statistics of our results: (i) the input sketch complexity shows the total number and total length of sketch lines, and the number of extracted coplanar components (see Section 5.1)); (ii) statistics of the generated models include the number of beams, total number of bricks (beams & connecting bricks), and model's physical size; and (iii) our method's running times.

	Input sketch complexity			Statistics of generated models			Running time (seconds)		
Model	#sketch lines	total length	#comp	#beams	#total bricks	phy. size (cm^3)	stage 1	stage 2	total time
CUBE9x9x9	12	96	6	20	44	7x7x7	0.09	2.78	2.87
PICKER	37	138	19	36	60	18x6x18	0.28	4.22	4.50
LITTLE_FERRIS	31	214	3	24	45	12x10x6	0.29	5.42	5.71
CHAIR_FRONT	17	82	6	11	33	6x6x12	0.11	5.71	5.82
CHAIR_SIDE	17	82	6	11	41	6x6x12	0.13	6.32	6.45
SEASAW	26	112	6	22	60	22x4x6	0.18	5.83	6.01
FLYING_KITE	16	91	2	21	47	14x20x5	0.13	6.33	6.46
BICYCLE	31	180	2	29	51	30x6x15	0.25	6.37	6.62
TABLE_LAMP	29	108	7	17	36	4x9x26	0.21	6.58	6.79
HOUSE	24	200	8	20	60	14x9x17	0.43	6.49	6.92
LONG_BOW	43	277	7	85	175	34x4x38	0.09	7.18	7.27
CLAW	33	116	6	29	70	22x4x10	0.32	7.64	7.96
LIFTER	57	260	9	28	90	9x9x20	0.55	7.50	8.05
CROSSBOW	51	268	18	52	119	23x25x17	0.34	8.79	9.13
GLASSES	49	242	3	45	93	17x7x23	0.48	9.76	10.24
BIRD	44	179	7	28	82	36x28x7	0.23	11.15	11.38
AIRPLANE	113	401	22	91	208	36x31x15	1.35	34.99	36.34
ROBOT	180	780	34	150	378	20x18x23	4.12	53.46	57.58
CASTLE	145	1579	9	267	495	72x75x14	6.78	102.03	108.81
SPACE_STATION	305	2102	45	477	1030	70x53x22	19.02	210.09	229.11
PRINT_BOX	166	1468	16	310	841	33x33x34	8.56	268.70	277.26
TOWER_3D	382	2524	77	482	1080	25x25x68	27.04	298.34	325.38
TOWER_2D	168	1402	1	223	458	4x87x150	3.16	345.97	349.13
BRIDGE	124	1812	3	196	514	14x107x33	8.34	360.00	368.34

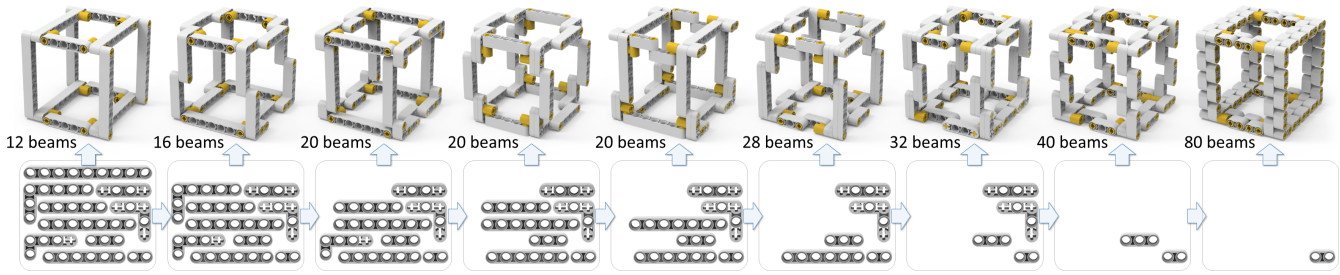


Fig. 16. Robustness of our method to brick sets: cubes generated by our method using different brick sets, from full to a single-beam set.

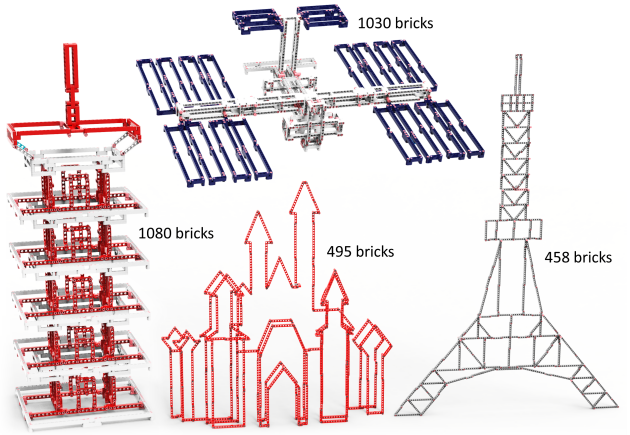


Fig. 17. Larger LEGO®Technic models generated by our method. From left to right, we have TOWER_3D, SPACE_STATION, CASTLE, and TOWER_2D.

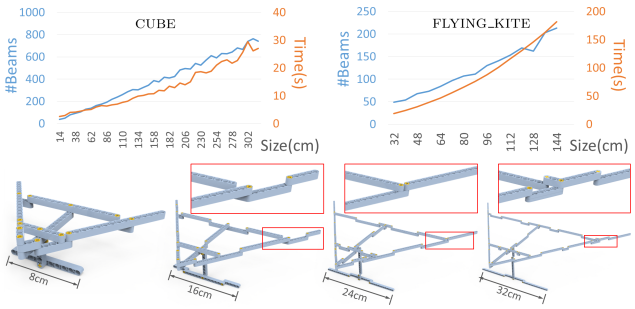


Fig. 18. Scalability test: CUBE and FLYING_KITE in increasing sizes.

connections. Overall, the whole method completes in only a few minutes, even for very large models with a thousand bricks. Note also that we sort the rows (models) in Table 1 by the total running time to reveal that the running time depends not only on the number of bricks but also on the complexity of the input models.

Scalability. We examine the scalability of our method by generating models for CUBE and FLYING_KITE in increasing sizes. Figure 18 shows the statistics of the results, where our method can efficiently generate models in varying scales within minutes, and the number of beams and running time increase roughly linearly.

Robustness to brick set. To explore our method's robustness to variations in brick set, we start with a full set of beams to generate CUBE with the goal of simple layouts. Then, we gradually take out the most frequently-used beam from the set and re-generate CUBE, until the set is empty. From the results shown in Figure 16, we can see that our method can produce coherent structures for all different brick sets. Importantly, beam arrangement is not a simple decision that greedily chooses the longest beam, but a global optimization that considers the overall structural coherency, simplicity, and symmetry. See the second cube in Figure 16, our method can skillfully arrange L-shaped beams to minimize the brick consumption; see Supplementary material part E for more results.

Efficiency comparison to alternative methods. To evaluate the efficiency of our method, we compare it with four different methods that are built upon our framework, and use them to generate Technic models, specifically for minimal gap and minimal beam counts: (i) a *random* method, which starts with a random layout and applies the layout modification operator to iteratively refine the layout for 50000n times (n is the input problem size to be described later); (ii) a *greedy* method, which progressively adds locally-optimum beams that cover the most portions of the uncovered sketch; (iii) a *beam search* method [Medress et al. 1977], which builds a three-layered search tree of partial layouts as internal nodes and iteratively updates the layout with the local best choice using a beam search width of 75; and (iv) an *ant colony* optimization method, which generalizes the set cover model in [Ren et al. 2010] to handle beam layering.

In the comparison, the input sketches we employed are 2D uniform grids of n-by-n cells (n=1..10); each cell is 4×4 sq. units in size. The difficulty in generating LEGO®Technic models for these grids, especially the larger ones, is that we have to minimize *both* gaps and beam counts, while considering beam *layering*; hence, we cannot simply use the longest beams, which will easily lead to gaps.

Figure 19 shows the comparison results: the number of gaps and beams, as well as the running times, for different grid sizes (n). Except for greedy and random, most methods can generate gap-free layouts for $n \leq 4$, where simply using the longest beams can effectively produce good solutions. Interestingly, the random method fails to find gap-free layouts even for $n = 2, 3$ grids; this reveals the immenseness of the search space. When n gets larger, random and greedy start to produce layouts with lots of gaps, while ant colony, beam search, and our method can still produce gap-free layouts for $n = 4, 6, 8$ grids, respectively. Particularly, ant colony can generate layouts with small number of beams for $n = 5, 6, 7$, but it fails to avoid gaps, since the search space is not only immense but also discrete rather than continuous. On the other hand, beam search can make good local decisions and avoids gaps better than

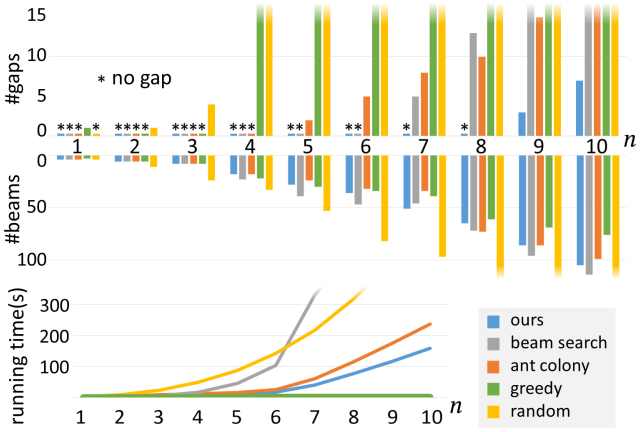


Fig. 19. Compare our method with four alternative methods (see legend) in terms of the number of gaps and beams in use, as well as the running times. The input models are uniform square grids with increasing sizes (n).

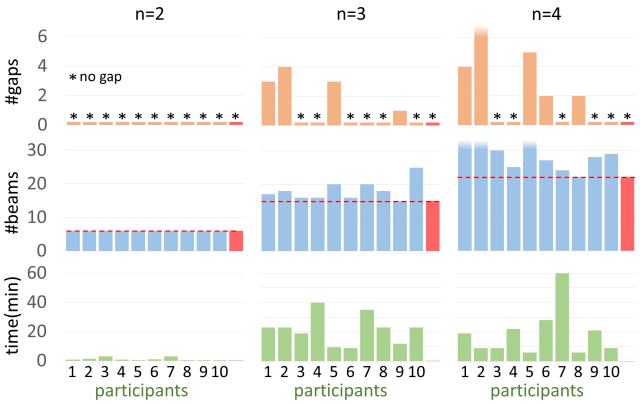


Fig. 20. Comparing manual designs with our method on generating LEGO® Technic grids of 2×2, 3×3, and 4×4 cells. The red bars show the performance of our method, but note that the timing bars are barely visible, since our method was able to compute the solutions in 2.5s, 5.8s, and 8.6s, respectively.

ant colony; however, it takes much longer time to run and fails to minimize beam counts for large n. In contrast, our method is able to produce gap-free layouts even for large grids (n=8), while effectively minimizing the beam count using much less computing time.

Manual designs. To obtain a sense of how difficult it is for humans to design LEGO®Technic models, we recruited ten participants (6 males & 4 females, aged 22 to 25). Among them, four had experience in building Technic models. Here, we followed the above comparison experiment and asked the participants to design Technic models for 2D grids with minimal gaps and beam counts. However, considering human building effort, we considered only $n = 2, 3, 4$ grids, and limited the brick set to contain only beams of up to nine units long. Before the experiment starts, we printed the grids on A4 papers in the same physical scale as the real bricks, showed the model for $n = 1$, and taught the participants the meaning of gaps. Then, we gave each participant at most an hour to work on each model, and recorded the resulting gap count, beam count, and time spent.

	CUBE		LIFTER		AIRPLANE	
Designed by	Human	Our	Human	Our	Human	Our
Num. of bricks	84	76	128	90	196	208
Beam deviation	0.35	0.35	0.27	0.46	0.15	0.24
Design time	4m51s	3s	35m13s	8s	1h23m37s	36s

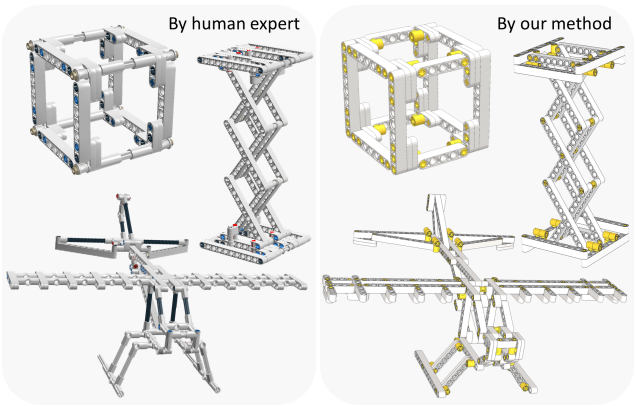


Fig. 21. Models designed by the LEGO®Technic expert and by our method.

Figure 20 shows the results; see Supplementary material part F for the models. For the smallest grid, all participants could find the optimal solution in a few minutes, while the two larger grids are more challenging: no one was able to find better or even the same solutions as our method, which are gap-free with the minimal beam count. For the largest grid, $n = 4$, only five participants found gap-free solutions, while one of them managed to find a solution with 24 beams, but it was still not as good as the solution produced by our method. Note that before this experiment on 2D grids, we did a pilot study asking the participants to build LEGO®Technic models of 3D cubes. However, those who did not have prior experience failed to connect bricks into cubes; some of them tried it for 50+ minutes. We concluded that it would be too challenging for non-expert participants to arrange beams in 3D without gaps.

Compare with human expert. Further, we recruited an expert who had over five-year full-time working experience specialized in LEGO®Technic design. In this experiment, we first showed to him some input sketches without showing him the models generated by our method, then asked him to design models that are faithful to these inputs with the least number of bricks. His first comment was that such requirement is the same as what he did in his daily designs. Also, we learnt that he preferred to create his designs using the LEGO®Digital Designer software [2018a] instead of manual assembly, since the software tool provides fast brick retrieval, more brick choices, and can replicate symmetric sub-structures. Here, we recorded his design time and the number of bricks in use, and computed the beam deviations in the results. From the results shown in Figure 21, we can see that the LEGO®Technic models produced by our method are similar to those designed by the expert, in terms of beam count, beam deviation, and visual appearance. However, designing the models using conventional software took the expert minutes to more than an hour, while our method is able to design comparable solutions in less than a minute.

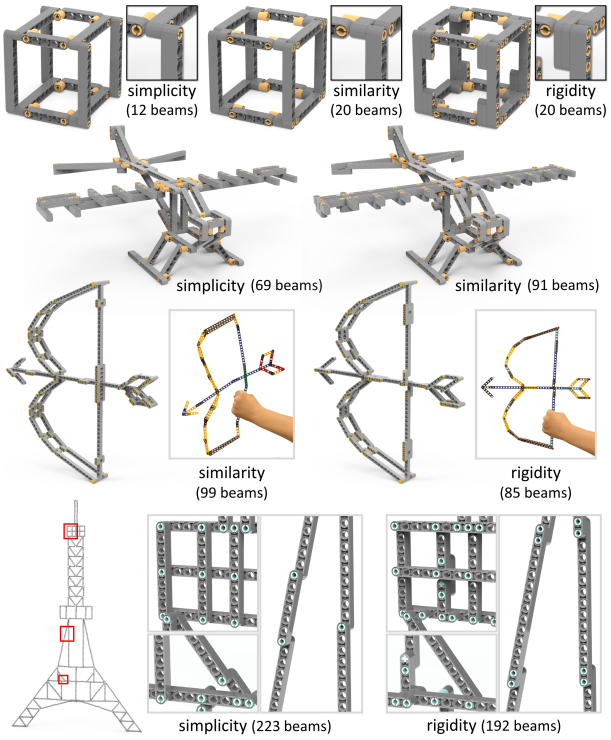


Fig. 22. LEGO®Technic models generated with emphasis on different terms in our objective, e.g., simplicity, rigidity, and similarity. Each row shows different results from the same input sketch.

Designs created by participants using our tool. We recruited seven participants (5 males & 2 females, aged 21 to 24) to try our tool. Among them, two had experience in building LEGO®Technic models and one had drawing background. When they came to the lab, we taught them our GUI and the requirements on the input sketch, then showed them a few example input sketches and the output Technic models. Then, each participant used around 15 min. to design what they wanted to create, and used around 37 min. on average to sketch their designs on our GUI. Figure 15 (bottom right) shows four of the models designed and assembled by the participants, featuring movie characters, everyday objects, and abstract models: LITTLE_FERRIS (design: 12 min., generation: 6 sec.), BICYCLE (design: 17 min., generation: 7 sec.), GLASSES (design: 27 min., generation: 10 sec.), and ROBOT (design: 87 min., generation: 58 sec.).

Adapting to different objectives. Our method can generate models from the same input sketch by emphasizing $\mathcal{F}_{\text{dev}}$, $\mathcal{F}_{\text{tbl}}$, and $\mathcal{F}_{\text{rgd}}$ in our objective to different extent; see Section 4.3 for the specific weight setting. For example, we can aim for simple layouts with minimized brick count or aim for high input similarity at the expense of using more bricks; see results for CUBE and AIRPLANE in Figure 22. Additionally, we can aim for connection rigidity and encourage our method to connect adjacent beams with multiple holes; see CUBE (rightmost), LONG_BOW, and TOWER_2D in Figure 22.

Ablation study on objective terms. Further, we conduct an experiment to show how other terms in the objective function affect the results. Here, we fix $\{w_{\text{dev}}, w_{\text{tbl}}, w_{\text{rgd}}\}$ as $\{0, 100, 0\}$ to ensure model simplicity, then test each of the other objective terms by independently adjusting its associated weight. We set the weight to zero to assess the necessity of an objective term, or double its value to increase its impact. Table 2 summarizes the ablation study results on two input models. Without changing the annealing temperatures and cooling rate, we found that doubling any particular weight usually breaks the balance among the objective terms and could deteriorate the results. On the other hand, neglecting a particular term could deteriorate the corresponding aspect in the resulting model, without improving much on the other aspects. Figure 23 shows some of the models generated in this ablation study.

Table 2. Effects of adjusting the weights of objective terms on number of beams ($|\mathcal{B}|$), layer compactness ($\mathcal{F}_{\text{cpt}}$), model symmetry ($\mathcal{F}_{\text{sym}}$), average pin-head ratio ($\bar{\rho}$), number of collisions (N_{col}), number of gaps (N_{gap}), and number of failure connections (N_{cfail}). We generate each result by independently modifying each weight, w_{cpt}, w_{sym}, w_{phr}, w_{col}, w_{gap}, or w_{coh}, while fixing the others. The adjustment either nullifies (null) the effect of the associated term by setting the weight to zero or emphasizes the effect by doubling the weight (x2). Compared with the result (last column) generated under the default weights, we highlight the improved aspects in light green and worsened aspects in light pink, showing that deviations from the default settings may improve certain aspects but could worsen others.

		w_{cpt}		w_{sym}		w_{phr}		w_{col}		w_{gap}		w_{coh}		default
		null	x2	null	x2	null	x2	null	x2	null	x2	null	x2	
FLYING_KITE	$\lvert\mathcal{B}\rvert$	15	14	15	20	15	20	14	14	14	16	14	16	14
	$\mathcal{F}_{\text{cpt}}$	9	4	9	4	4	4	4	9	4	4	4	4	4
	$\mathcal{F}_{\text{sym}}$	0.13	0.57	1.07	0.10	0.17	0.10	0.14	0.14	0.14	0.25	0.13	0.14	0.14
	$\bar{\rho}$	0.97	0.97	0.96	0.97	0.96	0.97	0.96	0.96	0.96	0.96	0.96	0.96	0.96
	N_{col}	0	1	2	0	0	0	1	0	0	0	0	0	0
	N_{gap}	1	1	0	1	1	1	1	0	3	0	1	2	1
	N_{cfail}	0	0	0	0	0	1	1	1	0	0	1	0	0
LONG_BOW	$\lvert\mathcal{B}\rvert$	98	90	82	86	84	87	88	104	82	80	83	89	85
	$\mathcal{F}_{\text{cpt}}$	16	9	9	9	9	9	9	9	9	16	9	9	9
	$\mathcal{F}_{\text{sym}}$	0.14	0.45	0.43	0.09	0.27	0.20	0.37	0.41	0.49	0.26	0.22	0.30	0.25
	$\bar{\rho}$	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
	N_{col}	0	0	0	0	0	0	2	0	1	2	0	2	0
	N_{gap}	0	5	0	3	0	0	0	7	4	0	0	0	0
	N_{cfail}	0	0	0	0	0	0	2	0	0	2	3	0	0

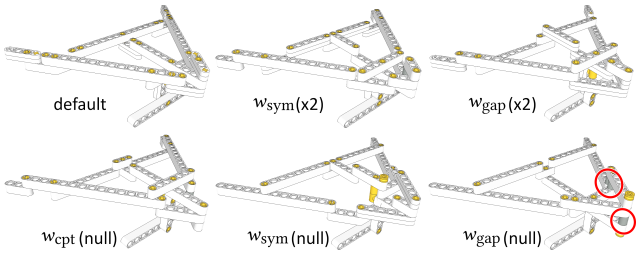


Fig. 23. Some of the generated models for FLYING_KITE in Table 2.

Dynamic models. Figure 24 shows physical assemblies designed with annotated joint rotations. Comparing the two CHAIRS, we can see that by annotating joints with different rotation axes, our method can constrain the connecting beams to rotate at specific orientations at the joints, thus leading to the production of different models. Besides, given a mechanical system (see the two examples in Figure 25), we can force the generated beams in the layout to exactly pass through some annotated sketch lines, such that we can then embed a mechanical system into the generated model. Please see the supplementary video for these models in action.

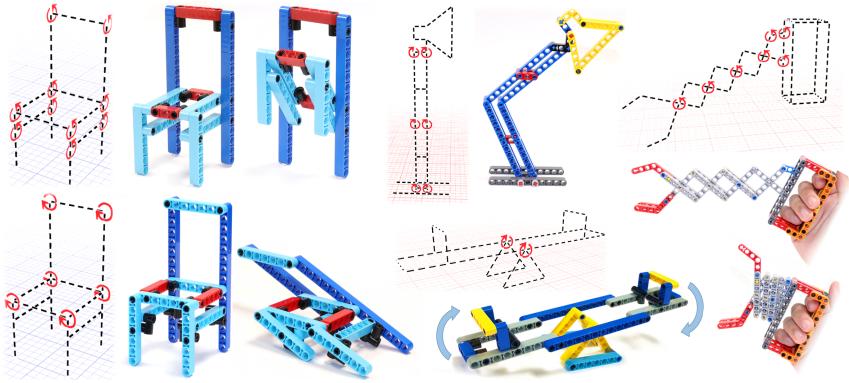


Fig. 24. Example models with annotated joint rotations generated from sketches. From left to right, we have CHAIR_FRONT (top left), CHAIR_SIDE (bottom left), TABLE_LAMP, SEASAW, and CLAW.

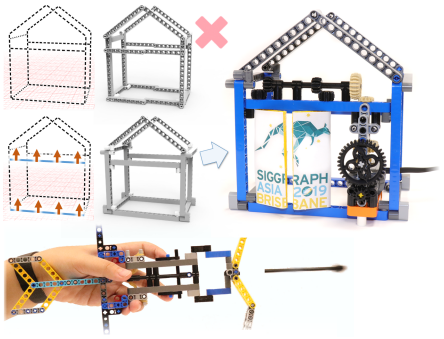


Fig. 25. Top: HOUSE directly generated from the input sketch. Middle: beams are constrained to exactly pass through the annotated sketch line (in blue) for embedding a gear system. Bottom: LONG_BOW embedded with a shooting mechanic.

Discussion on global rigidity. If we optimize the LONG_BOW model without the rigidity term, the model could be deformed due to the gravity, as demonstrated in Figure 22 (middle left). To evaluate the global rigidity of a LEGO®Technic structure is a very challenging problem. First, global non-rigidity is a result of multiple (a subset of) joints in the overall structure. Particularly, a joint may be transitively (or indirectly) blocked to rotate by others that are not located next to it. Here, trying every joint subset would require tremendous computation. Also, we have to deal with a diverse and irregular brick set that can be connected in many different ways. Further, we need a unified model to evaluate the effectiveness of individual connections, brick-blocking relations, and other physical constraints such as friction. Hence, we believe analytically evaluating the global rigidity is very challenging, and will require extensive works.

In the course of this work, we have thought of two approaches to this problem: (i) apply external forces on the structure, then see if the structure deforms in a physical simulation; and (ii) formulate a motion-based equation by setting velocity variables on every joint, constrain them based on the beam connections, solve it, and locate the joints with non-zero velocities. Clearly, these approaches are preliminary and require more thoughtful ideas to turn them into solid feasible solutions. Hence, we only consider local rigidity in this work, and leave global rigidity as our future work.

7 CONCLUSION, LIMITATION, AND FUTURE WORK

We presented a first attempt to computerize LEGO®Technic constructions. Altogether, there are three contributions in this work. First is an automatic computational method that can efficiently generate LEGO®Technic models from user input sketches. Particularly, the generated model is a coherently-connected structure composed of LEGO®Technic bricks, and we can aim for faithfulness to input sketch, model simplicity, and structural integrity in the model generation. Second is our comprehensive model for various aspects in LEGO®Technic constructions, including the enumeration of brick properties and connection mechanisms, conceptualization of the input sketch as a guiding graph, formulation of the construction requirements into an objective, and dynamic model constructions with hinge-style rotations and dynamic parts embedding. Third, we

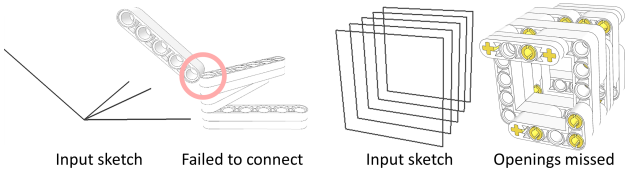


Fig. 26. Failure cases from our current computational method: intersecting sketch lines (left) and closely-packed sketch lines (right).

also developed a working system to sketch the inputs, and to analyze the balance, stress, and assemblability of the generated model. In the end, we employed our system to create LEGO®Technic constructions of various shapes, complexities and functionalities, compared it with four alternative methods, general users and a human expert, evaluated it for scalability, robustness and adaptiveness, as well as physically built most of the generated models.

Limitations. In terms of model generation, while our method makes an effort to adjust the beams, it may still fail to create connections in some situations, especially when several beams intersect/touch one another non-orthogonally; see Figure 26 (left). For dense and parallel sketch lines, the generated models may not retain the gaps between the lines; see Figure 26 (right). Also, our current method assumes most sketch lines are covered by the beam bricks. However, as shown in the expert's designs, axles may also be used to cover the sketch lines, where some special axle-related connector bricks can be used in the connections. Moreover, as discussed earlier, our current formulation for connection rigidity is local, not global. Lastly, in our sketching tool, diagonal lines in the sketches should follow certain Pythagorean triples, e.g., after we sketch an L-structure with two orthogonal lines, if we want to use an extra line to diagonally connect the previous two lines, the length of the diagonal line may be five-unit long, since $5^2 = 3^2 + 4^2$.

In terms of our handling of dynamic constraints in the input, while our method realizes user-annotated hinge-style rotations, the decision of where to put joints and how to make different parts work together to realize a desirable dynamic behavior still remains hard for novice users. Also, besides hinge-style rotations, other LEGO®Technic motions such as sliding, sheering, lifting, and their combinations are not yet considered in our system.

Discussion and future work. Addressing the various limitations above already suggests a comprehensive LEGO®Technic design system involving a number of sub-problems, such as global rigidity analysis, rigid structure generation, inverse joints computation, and inverse multi-functionality design with mechanical elements. Moreover, official LEGO®Technic models often contain customized parts specially-designed for the outer shell of the model. We would also like to design and fabricate customized 3D-printed parts to work with the beams and connectors in the LEGO®Technic system for driving 3D-printed customized models.

ACKNOWLEDGMENTS

We thank all the anonymous reviewers for their comments and feedback. We also acknowledge help from Ruihui Li for UI development, Shufang Wang and Tianwen Fu for model assembly, Chun Yu Liu for designing the models shown in Figure 21, and Wallace Lira and Johannes Merz for paper proofreading. Figures 2 (a), (b) and (c) are courtesy of YouTuber TECHNICally Possible, Nicolas Lespour, and Will Gorman, respectively. This work is supported in part by grants from the Research Grants Council of the Hong Kong Special Administrative Region (Project no. CUHK 14201918 and 14203416), NSERC grants (No. 611370), and gift funds from Adobe.

REFERENCES

Luca Baronti, Matteo Dellepiane, and Roberto Scopigno. 2010. Using Lego Pieces for Camera Calibration: a Preliminary Study. In *Eurographics (short paper)*. 97–100.

Jonathan Cagan, Drew Degentesh, and Su Yin. 1998. A simulated annealing-based algorithm using hierarchical models for general three-dimensional component layout. *Computer-aided design* 30, 10 (1998), 781–790.

Xuelin Chen, Honghua Li, Chi-Wing Fu, Hao(Richard) Zhang, Daniel Cohen-Or, and Baoquan Chen. 2018. 3D Fabrication with Universal Building Blocks and Pyramidal Shells. *ACM Trans. on Graph. (SIGGRAPH Asia)* 37, 6 (2018).

Paolo Cignoni, Nico Pietroni, Luigi Malomo, and Roberto Scopigno. 2014. Field-aligned Mesh Joinery. *ACM Trans. on Graph. (SIGGRAPH Asia)* 33, 1 (2014). Article no. 11.

Travis Cobbs and Peter Bartfai. 2018. LDview. http://www.ldraw.org/ [Online; accessed 18-May-2018].

Juan De Vicente, Juan Lanchares, and Román Hermida. 2003. Placement by thermodynamic simulated annealing. *Physics Letters A* 317, 5-6 (2003), 415–423.

Ruta Desai, James McCann, and Stelian Coros. 2018. Assembly-aware Design of Printable Electromechanical Devices. In *The 31st Annual ACM Symposium on User Interface Software and Technology*. ACM, 457–472.

Mario Deuss, Daniele Panozzo, Emily Whiting, Yang Liu, Philippe Block, Olga Sorkine-Hornung, and Mark Pauly. 2014. Assembling Self-supporting Structures. *ACM Trans. on Graph. (SIGGRAPH Asia)* 33, 6 (2014). Article no. 214.

Moritz Geilinger, Roi Poranne, Ruta Desai, Bernhard Thomaszewski, and Stelian Coros. 2018. Skaterbots: optimization-based design and motion synthesis for robotic creatures with legs and wheels. *ACM Trans. on Graph. (SIGGRAPH)* 37, 4 (2018).

Rebecca A. H. Gower, Agnes E. Heydtmann, and Henrik G. Petersen. 1998. LEGO: Automated Model Construction. In *European Study Group with Industry*. 81–94.

Yoshihito Isogawa. 2010. *LEGO® Technic Idea Book: Fantastic Contraptions.* No Starch Press.

Jae Woo Kim, Kyung Kyu Kang, and Ji Hyoung Lee. 2014. Survey on Automated LEGO Assembly Construction. In *Proc. WSCG*. 89–96.

Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (1983), 671–680.

Pawel "Sariel" Kmieć. 2016. *The Unofficial LEGO® Technic Builder's Guide, 2nd Edition.* No Starch Press.

Ming-Hsun Kuo, You-En Lin, Hung-Kuo Chu, Ruen-Rone Lee, and Yong-Liang Yang. 2015. PIXEL2BRICK: Constructing Brick Sculptures from Pixel Art. *Computer Graphics Forum (Pacific graphics)* 34, 7 (2015), 339–348.

Michael Lachmann. 2018. MLCad. http://mlcad.lm-software.com/ [Online; accessed 5-Nov-2018].

Bram Lambrecht. 2006. Voxelization of boundary representations using oriented LEGO® plates. University of California, Berkeley, http://lego.bldesign.org/ [Online; accessed 18-May-2018].

Manfred Lau, Akira Ohgawara, Jun Mitani, and Takeo Igarashi. 2011. Converting 3D Furniture Models to Fabricatable Parts and Connectors. *ACM Trans. on Graph. (SIGGRAPH)* 30, 4 (2011). Article no. 85.

Seung-Mok Lee, Jae Woo Kim, and Hyun Myung. 2018. Split-and-Merge-Based Genetic Algorithm (SM-GA) for LEGO Brick Sculpture Optimization. *IEEE Access* 6 (2018), 40429–40438.

Sheng-Jie Luo, Yonghao Yue, Chun-Kai Huang, Yu-Huan Chung, Sei Imai, Tomoyuki Nishita, and Bing-Yu Chen. 2015. Legolization: Optimizing LEGO Designs. *ACM Trans. on Graph. (SIGGRAPH Asia)* 34, 6 (2015). Article no. 222.

Robert B. McGhee and Andrew A. Frank. 1968. On the stability properties of quadruped creeping gaits. *Mathematical Biosciences* 3 (1968), 331–351.

Mark F. Medress, Franklin S. Cooper, Jim W. Forgie, CC Green, Dennis H. Klatt, Michael H. O'Malley, Edward P. Neuburg, Allen Newell, DR Reddy, B Ritea, et al. 1977. Speech understanding systems: Report of a steering committee. *Artificial Intelligence* 9, 3 (1977), 307–316.

Niloy J. Mitra and Mark Pauly. 2009. Shadow Art. *ACM Trans. on Graph. (SIGGRAPH Asia)* 28, 5 (2009). Article No. 156.

Stefanie Mueller, Tobias Mohr, Kerstin Guenther, Johannes Frohnhofen, and Patrick Baudisch. 2014. faBrickation: Fast 3D Printing of Functional Objects by Integrating Construction Kit Building Blocks. In *CHI*. 3827–3834.

Yaghout Nourani and Bjarne Andresen. 1998. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General* 31, 41 (1998), 8373.

Pavel Petrovič. 2001. Solving LEGO Brick Layout Problem using Evolutionary Algorithms. In *Proc. NIK (Norsk Informatikkonferanse)*. 87–97.

Nico Pietroni, Marco Tarini, Amir Vaxman, Daniele Panozzo, and Paolo Cignoni. 2017. Position-based Tensegrity Design. *ACM Trans. on Graph. (SIGGRAPH Asia)* 36, 6 (2017). Article no. 172.

Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. 2013. Make It Stand: Balancing Shapes for 3D Fabrication. *ACM Trans. on Graph. (SIGGRAPH)* 32, 4 (2013). Article no. 81.

Zhi-Gang Ren, Zu-Ren Feng, Liang-Jun Ke, and Zhao-Jun Zhang. 2010. New ideas for applying ant colony optimization to the set covering problem. *Computers & Industrial Engineering* 58, 4 (2010), 774–784.

Trevor Sandy. 2018. LPub3D. https://trevorsandy.github.io/lpub3d/ [Online; accessed 29-Dec-2018].

Philip Schneider and David H. Eberly. 2002. *Geometric Tools for Computer Graphics.* Morgan Kaufmann Publishers.

Adriana Schulz, Ariel Shamir, David I.W. Levin, Pitchaya Sitthi-Amorn, and Wojciech Matusik. 2014. Design and fabrication by example. *ACM Trans. on Graph. (SIGGRAPH)* 33, 4 (2014). Article no. 62.

Mélina Skouras, Stelian Coros, Eitan Grinspun, and Bernhard Thomaszewski. 2015. Interactive Surface Design with Interlocking Elements. *ACM Trans. on Graph. (SIGGRAPH Asia)* 34, 6 (2015). Article no. 224.

Peng Song, Bailin Deng, Ziqi Wang, Zhichao Dong, Wei Li, Chi-Wing Fu, and Ligang Liu. 2016. CofiFab: Coarse-to-fine Fabrication of Large 3D Objects. *ACM Trans. on Graph. (SIGGRAPH)* 35, 4 (2016). Article no. 45.

Peng Song, Chi-Wing Fu, and Daniel Cohen-Or. 2012. Recursive Interlocking Puzzles. *ACM Trans. on Graph. (SIGGRAPH Asia)* 31, 6 (2012). Article no. 128.

Ben Stephenson. 2016. A Multi-Phase Search Approach to the LEGO Construction Problem. In *Proc. Symposium on Combinatorial Search (SoCS)*. 89–97.

Romain Testuz, Yuliy Schwartzburg, and Mark Pauly. 2013. Automatic Generation of Constructible Brick Sculptures. In *Eurographics (short paper)*. 81–84.

The LEGO® Group. 2018a. LEGO® Digital Designer, version 4.3.11. https://www.lego.com/en-us/ldd/download/ [Online; accessed 10-January-2018].

The LEGO® Group. 2018b. LEGO® Technic official site. https://www.lego.com/en-us/technic/ [Online; accessed 18-May-2018].

Bernhard Thomaszewski, Stelian Coros, Damien Gauge, Vittorio Megaro, Eitan Grinspun, and Markus Gross. 2014. Computational Design of Linkage-based Characters. *ACM Trans. on Graph. (SIGGRAPH)* 33, 4 (2014). Article no. 64.

Pascal Van Hentenryck and Yannis Vergados. 2007. Population-based simulated annealing for traveling tournaments. In *Proceedings of the National Conference on Artificial Intelligence*, Vol. 22. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 267.

Martin Waßmann and Karsten Weicker. 2012. Maximum Flow Networks for Stability Analysis of LEGO® Structures. In *Proc. Annual European Conference on Algorithms (Lecture Notes in Computer Science, vol 7501)*. 813–824.

David V. Winkler. 2005. Automated Brick Layout. In *Proc. BrickFest*. 145–166.

Jiaxian Yao, Danny M. Kaufman, Yotam Gingold, and Maneesh Agrawala. 2017. Interactive Design and Stability Analysis of Decorative Joinery for Furniture. *ACM Trans. on Graph. (SIGGRAPH)* 36, 2 (2017). Article no. 20.

Hironori Yoshida, Takeo Igarashi, Yusuke Obuchi, Yosuke Takami, Jun Sato, Mika Araki, Masaaki Miki, Kosuke Nagata, Kazuhide Sakai, and Syunsuke Igarashi. 2015. Architecture-scale Human-assisted Additive Manufacturing. *ACM Trans. on Graph. (SIGGRAPH)* 34, 4 (2015). Article no. 88.

Grim Yun, Cheolseong Park, Heekyung Yang, and Kyungha Min. 2017. Legorization with multi-height bricks from silhouette-fitted voxelization. In *Proc. CGI*. Article No. 40.