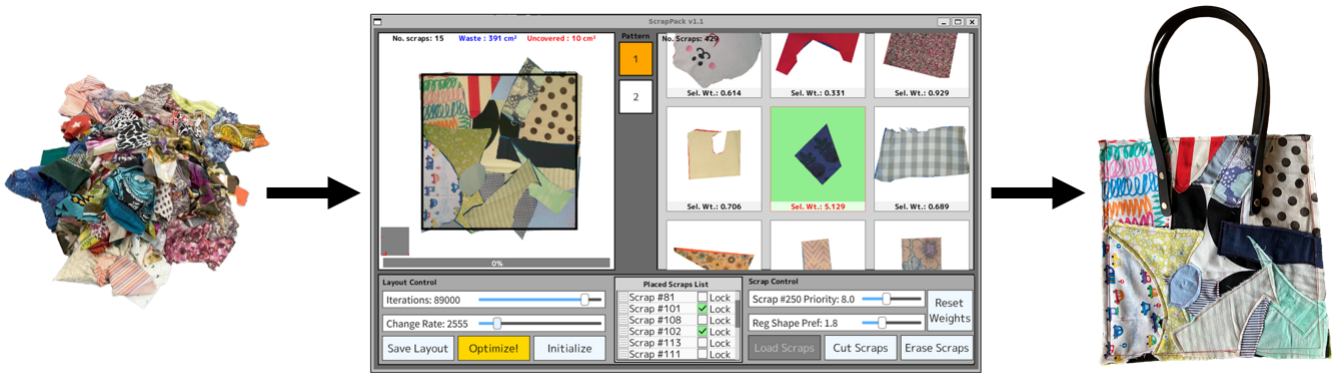
ScrapReCover: An Interactive Optimization System for Freeform Patchwork Layouts

Masahiro Kono
The University of Tokyo
Tokyo, Japan
marckono2825@g.ecc.u-tokyo.ac.jp

Maria Larsson
The University of Tokyo
Tokyo, Japan
ma.ka.larsson@gmail.com

I-Chao Shen
The University of Tokyo
Tokyo, Japan
jdilyshen@gmail.com

Takeo Igarashi
The University of Tokyo
Tokyo, Japan
takeo@acm.org



(a) Fabric scraps for reuse (b) *ScrapReCover* UI for interactive layout optimization (c) Fabricated patchwork

Figure 1: *ScrapReCover* is an interactive tool for designing freeform patchwork layouts from fabric scraps. (a) Users begin by loading the scraps they wish to reuse. (b) Using *ScrapReCover*, they can iteratively refine the layout by combining manual adjustments of individual scraps with automatic placement suggestions guided by intuitive parameters. For automatic suggestions, the system arranges scraps within the target area, following an optimization strategy that minimizes material waste while ensuring complete coverage. (c) Finally, the resulting layout can be fabricated into a unique patchwork product.

Abstract

We present *ScrapReCover*, an interactive tool for designing freeform patchwork layouts from small leftover fabric scraps. The system enables users to iteratively refine the layout by combining manual adjustments of individual scraps with automatic placement suggestions guided by user-controlled parameters. These parameters can be intuitively adjusted to control the degree of modification from the current layout and to prioritize specific types of scraps. For automatic suggestions, the system generates layouts by arranging arbitrarily shaped scraps within the target area, using an optimization strategy that minimizes material waste while ensuring complete coverage. To achieve this functionality, *ScrapReCover* employs simulated annealing (SA), a robust metaheuristic and stochastic algorithm known for its effectiveness in packing-like problems, integrated with a rasterized representation of both scraps and pattern shapes. The usability of the system was validated through a user study in which 21 participants interactively generated layouts and fabricated patchworks from their own scraps. Additionally, the optimization method was evaluated by baseline comparisons which demonstrate that our approach outperforms other naive methods.

SCF '25, Cambridge, MA, USA

ACM ISBN 979-8-4007-2034-5/25/11
https://doi.org/10.1145/3745778.3766653

CCS Concepts

• **Computing methodologies** → **Graphics systems and interfaces**.

Keywords

Patchwork, computer-aided design, fabrication, design tools, discrete optimization, geometric covering

ACM Reference Format:
Masahiro Kono, Maria Larsson, I-Chao Shen, and Takeo Igarashi. 2025. ScrapReCover: An Interactive Optimization System for Freeform Patchwork Layouts. In *ACM Symposium on Computational Fabrication (SCF '25), November 20–21, 2025, Cambridge, MA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3745778.3766653

1 Introduction

Manufacturing products like garments generates a significant amount of material waste, both as a by-product of production and through the disposal of worn-out fabrics. The growing volume of such waste has become a major environmental concern, highlighting the need for effective reuse strategies. One notable approach to repurposing discarded materials is patchwork, a traditional technique that upcycles fabric scraps into visually intricate handcrafts. However, the irregular shapes and varied colors of these scraps present challenges in creating efficient layouts with aesthetically pleasing designs, particularly for those with little experience in this form of craft.

Existing systems for assisting patchwork design primarily focus on conventional styles based on square grids or repetitive quilting patterns (e.g., Figure 2a) [Bakker and Verhoeff 2022; Leake and Daly 2024; Shinjo et al. 2024]. Although effective for certain design goals, these approaches cannot be directly applied to irregularly shaped fabric scraps. In contrast, our aim is to support the creation of freeform layouts that embrace the inherent irregularity of fabric scraps, promoting more efficient material use while offering a distinct appearance that reflects the aesthetic of randomness (e.g., Figure 2b). Related systems such as PatchProv [Leake et al. 2021] and Patchy [Igarashi and Mitani 2015] also support the creation of original patchwork designs through interactive procedures; however, they do not offer automatic layout suggestions based on optimization, requiring users to invest additional time and skill to achieve satisfactory results that effectively utilize discarded scraps.

To address this challenge, we present *ScrapReCover*, an interactive computer-aided design tool that supports the creation of original freeform patchwork layouts from a given set of small leftover fabric scraps. Rather than relying on a single-shot optimization, the system enables users to iteratively refine the layout by seamlessly combining manual adjustments of individual scraps with automatic placement suggestions guided by user-controlled parameters. These parameters can be adjusted via a simple slider to control the degree of modification from the current layout and to prioritize either regular or irregular scrap types. Since our goal is to encourage unplanned creativity through various outputs, inspired by *collage art* [Greenberg 2018], the optimization method is designed to help users achieve two necessary and quantifiable objectives: complete coverage and minimal waste. Given that our system operates on small scraps (typically under 20×20 cm), remnants of the layout often result in unusable offcuts. These fragments are treated as material waste, and minimizing them is key to improving fabric utilization. Therefore, for automatic layout suggestions, we formulate the task as a geometric covering problem [Contardo and Hertz 2021; Shragai and Elber 2013] involving irregularly shaped polygons. Our system addresses this problem by arranging arbitrarily shaped scraps within the target area, using an optimization strategy that minimizes overlaps between scraps and out-of-bounds placements while ensuring complete coverage, ultimately producing unique designs.

Figure 2: Comparison between (a) traditional patchwork with repetitive design[1] and (b) our target freeform design[2].

To achieve these capabilities and explore the immense solution space, *ScrapReCover* discretizes the problem through rasterization and employs simulated annealing (SA) [Kirkpatrick et al. 1983], a robust metaheuristic and stochastic algorithm known for its effectiveness in packing-like problems [Bajuelos et al. 2009; Dowsland 1993; Gomes and Oliveira 2006; Tole et al. 2023]. SA is well-suited to our context as it can converge on near-optimal solutions under complex constraints within a practical timeframe for interactive applications, and is able to perform multiple rounds of optimization from arbitrary configurations. Moreover, its inherent randomness aligns with our goal of generating diverse results. The combination of generating diverse outcomes and supporting iterative refinement creates opportunities for human intervention throughout the design workflow. We augment SA with a neighborhood function composed of five primitive operations and an objective function that incorporates a penalty term. These components are configured via a set of user-adjustable parameters, allowing users to specify optimization goals in an interpretable manner.

The effectiveness of our system was evaluated through both qualitative and quantitative studies. We conducted a user study with 21 participants, where each participant brought their own fabric scraps, and interactively designed patchwork layouts using our system. Some of the participants then fabricated their designs into real-world patchworks based on the resulting layouts, demonstrating the practical usability of our system. In addition, we compared our optimization-based method with naive baselines, which confirmed that our approach achieves better visual quality and reduces material waste more effectively.

To summarize, our main contributions are:

- An interactive computer-aided design tool that supports the creation of freeform patchwork layouts from small leftover fabric scraps.
- The formalization of a novel problem setting for arranging irregularly shaped scraps in patchwork layouts.
- A robust discrete algorithm based on simulated annealing that efficiently addresses the problem, supports user-driven iterative refinement, and yields diverse results.

Our code are available at https://marc2825.github.io/ScrapReCover.

[1] https://unsplash.com/photos/a-patchwork-quilt-with-an-orange-and-brown-design-rOjcU5L4CpM
[2] https://pixabay.com/illustrations/background-embroidery-pattern-7566184/

2 Related Work

2.1 Computational Design for Patchwork

Several systems have been proposed to assist with quilt and patchwork design. ScrapMap [Leake and Daly 2024] and Shinjo et al. [2024] focused on color arrangement tasks to help users create visually compelling designs. Other approaches have explored algorithmic generation for free-motion quilting [Li et al. 2019], and automated design of foundation paper pieceable (FPP) quilts [Bakker and Verhoeff 2022; Leake et al. 2022]. However, these works rely on structured patches or conventional design motifs. To support more original design creation, PatchProv introduced an improvisational quilting workflow [Leake et al. 2021], Patchy provided a painting-based interface for manual patchwork design [Igarashi and Mitani 2015], Liu et al. [2017] presented a method that automatically generates sewing patterns for whole-cloth quilts from photographs by extracting edges and solving a variant of the Rural Postman Problem, and McCormack and Schwarz [2024] explored human–machine co-creativity through generative embroidery. Nevertheless, none of these systems offer automatic placement suggestions during the interactive process. In contrast, our approach integrates automatic suggestions for freeform design that embraces the irregularity of fabric scraps, aiming to extend users' creativity through iterative, optimization-driven support using a limited amount of materials.

2.2 Designing for Low-Waste Production

Various approaches were explored to support low-waste or even zero-waste production through computer-aided methods. Wastebanned assisted users in editing zero-waste fashion patterns [Zhang et al. 2024], while Scrappy reduced material usage in 3D printing by reusing discarded material as infill [Wall et al. 2021]. Box Cutter provided an efficient UV atlas packing algorithm applicable to various domains [Limper et al. 2018]. McQuillan et al. [2018] offered an open-source system for zero-waste garment making through printed navigational markers, and also investigated the use of 3D modeling software for designing zero-waste garments [McQuillan 2020]. Koo et al. [2016] introduced a dynamic design adjustment system for furniture to minimize offcuts. Patching minimized plastic waste in iterative 3D printing by re-fabricating only the modified parts [Teibrich et al. 2015]. Fabricaide introduced a fabrication-aware design tool that integrates a custom packing algorithm to provide feedback on material use [Sethapakdi et al. 2021]. PacCAM presented a system that combines computer vision and interactive simulation to pack 2D parts onto source materials for digital fabrication [Saakes et al. 2013]. Rags2Riches proposed an algorithm for garment upcycling, formulating it as a quantized discrete assignment solved with an ILP solver [Qi et al. 2025]. Baas et al. [2025] presented a method for approximating 3D surfaces by reusing panels from discarded materials. Similarly, we aim to promote efficient placement strategies for reusing fabric scraps, with the goal of minimizing material waste as effectively as possible.

2.3 The Packing and Covering Problem

Several related well-known problems, such as the "packing problem" and its dual problem (the "covering problem"), has been extensively studied in the field of discrete mathematics and both has been proven to be NP-complete [Culberson and Reckhow 1994; Fowler et al. 1981]. Traditional packing focused on the non-overlapping arrangement of smaller objects within a confined space [Dyckhoff 1990], and many heuristic approaches such as the First-Fit-Decreasing (FFD) strategy [Johnson 1973], Bottom-Left (BL) algorithm [Chazelle 1983], No-Fit Polygon (NFP) method [Oliveira et al. 2000], or applying beam search (BS) [Bennell and Song 2010] had been proposed. Additionally, numerous studies had applied various metaheuristic algorithms to address the vast solution space [Hopper and Turton 1999, 2001], including our simulated annealing-based method [Dowsland 1993; Gomes and Oliveira 2006; Tole et al. 2023]. Our problem is closely related to the 2D irregular packing problem (nesting problem) [Guo et al. 2022], which has applications in various real-world scenarios like fabric utilization in garment manufacturing [Bennell and Oliveira 2008]; however, our setting permits overlaps, differentiating the nature of the problem.

Likewise, the covering problem aims to select a subset of elements to ensure complete coverage of a target while minimizing resource usage [Borndörfer 1998; Chvatal 1979]. Specifically, our problem setting can be classified as a geometric covering problem [Contardo and Hertz 2021; Shragai and Elber 2013]. However, previous research had typically focused on simplified configurations (e.g., involving only orthogonal edges [Franzblau and Kleitman 1984; Kumar and Ramesh 2003]), on practical scenarios (e.g., the "art gallery problem" [Bajuelos et al. 2009; De Berg 2000], sensor network coverage [Huang and Tseng 2003]), or on cases where some uncovered regions were tolerated [Kwan et al. 2016; Minarčík et al. 2024]. In our situation, the input shapes are irregular, finite, and pairwise distinct, which significantly increases the complexity of the problem. We speculate that this unique setting has remained relatively unexplored due to its complex mathematical assumptions despite its limited correspondence with real-world applications.

3 System Overview

To address the challenge of supporting users in creating freeform design patchworks, we propose a workflow centered around our interactive tool. The process begins with users selecting the fabric scraps they wish to reuse and capturing a photograph from directly above, using a fixed camera plane parallel to the surface. With pattern shapes defined beforehand, users launch *ScrapReCover* and import the selected fabric scraps through the segmentation

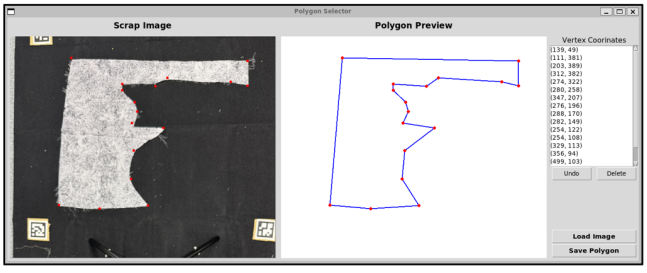


Figure 3: The segmentation interface for converting each scrap into a digital form. It allows users to manually specify the desired region of each photographed scrap by outlining it as a polygon, which is then imported into the system.

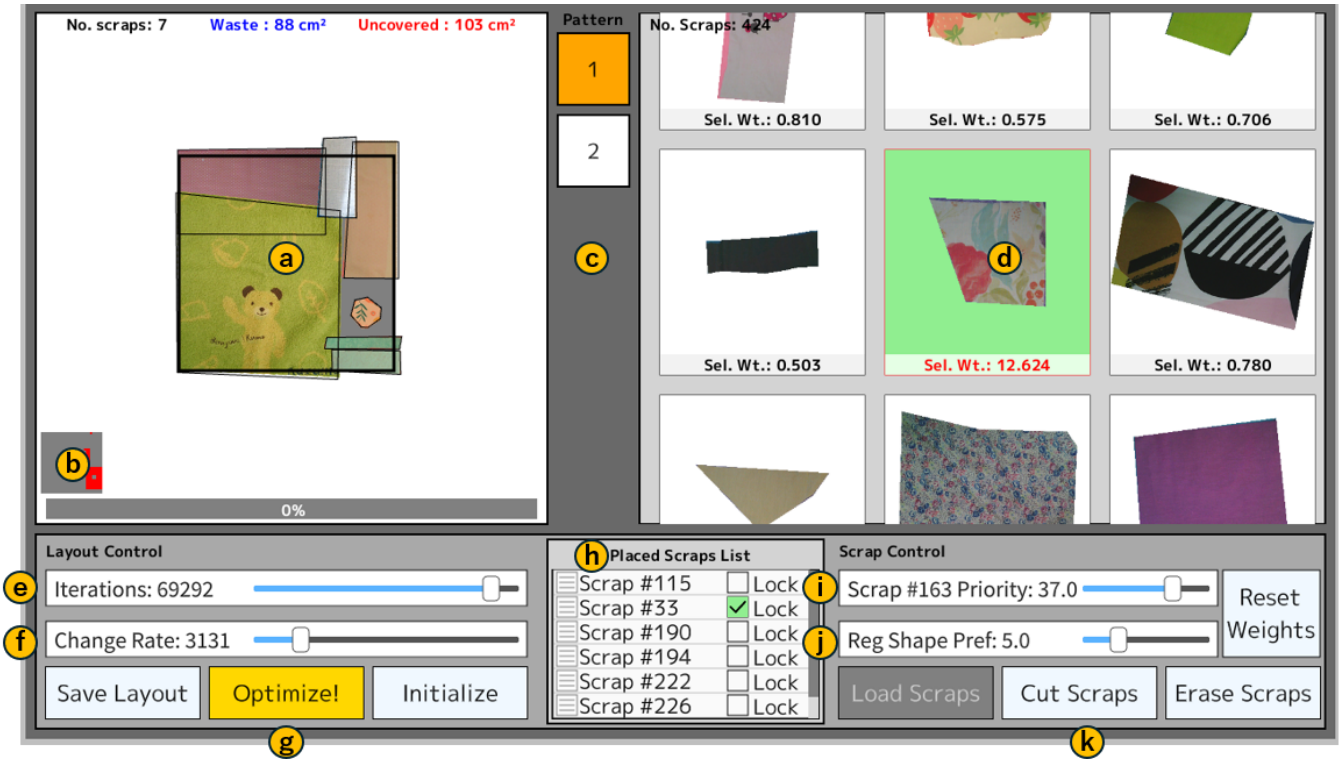


Figure 4: The *ScrapReCover* UI consists of five main components. *Layout Workspace* (top left, a-c), where scraps are placed on the pattern; *Available Scraps List* (top right, d), which displays the unplaced scraps; *Layout Control Panel* (bottom left, e-g), a set of global layout controls; *Placed Scraps List* (bottom center, h), which enumerates the placed scraps; and *Scrap Control Panel* (bottom right, i–k), a set of controls for individual scraps. By combining manual operations with user-controlled automatic placement suggestions, users can create their designs in an iterative and interactive manner.

interface as polygonal data, then iteratively create and refine their layout. *ScrapReCover* supports this process by providing automatic placement suggestions guided by user-controlled parameters. These parameters, adjustable via a simple slider interface, allow users to control the degree of modification from the current layout and to prioritize specific types of scraps. In addition to automatic suggestions, manual operations such as moving, rotating, and locking the position of individual scraps are also supported. Furthermore, users can return to the segmentation interface at any point to re-edit the input shapes, akin to physically trimming fabric scraps during the design process. Once the layout meets the user's satisfaction, it can be exported as an image file, which can then be printed and used as a reference for real-world fabrication.

3.1 Preparation

3.1.1 Scrap registration. As a fundamental step, each fabric scrap must be imported into the system in digital form. To support this process, we developed a segmentation interface (Figure 3) that allows users to manually define the desired portion of each photographed scrap by outlining it as a polygon, which provides greater flexibility and reliability. For example, even if a scrap is stained, users can intuitively "cut" out the usable portion, which is often difficult to automate. After photographing each scrap from directly above, with the fixed camera plane parallel to the surface, the user launches the interface by clicking the *Load Scraps* button in *ScrapReCover* (Figure 4k). An appropriate scaling factor is then applied based on the known distance between the camera and the photographed surface, and the coordinates of the polygon's vertices are imported into the system. This process effectively serves as a virtual cutting operation, enabling only the selected portion of the scrap to be represented and used as polygonal data within the system. Users can return to this step at any time to re-edit the currently loaded shape by clicking the *Cut Scraps* button (Figure 4k).

3.1.2 Target pattern. The user specifies a predefined target pattern, depending on what they want to make, for example, a rectangular pattern for making a tote bag or pillow case, or more complex patterns for making a stuffed toy or other custom fabric items.

3.2 Components of *ScrapReCover*

Figure 4 shows the components of the user interface for *ScrapReCover*. It is implemented in C++ with the Siv3D library [Suzuki 2020]. This UI is composed of five main components: *Layout Workspace* (top left, a-c), *Available Scraps List* (top right, d), *Layout Control Panel* (bottom left, e-g), *Placed Scraps List* (bottom center, h), and *Scrap Control Panel* (bottom right, i–k).

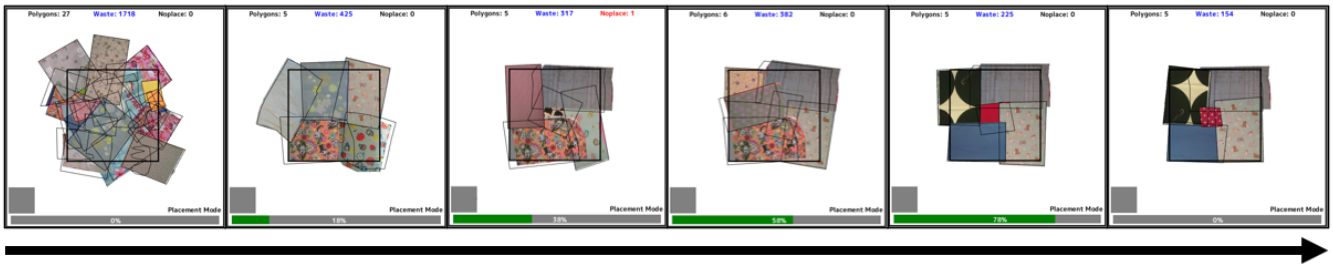


Figure 5: Visualization of the optimization process. The layout gradually converges from an initial state with many overlapping and out-of-bounds to a more packed configuration, where most are contained within the pattern and overlaps are reduced.

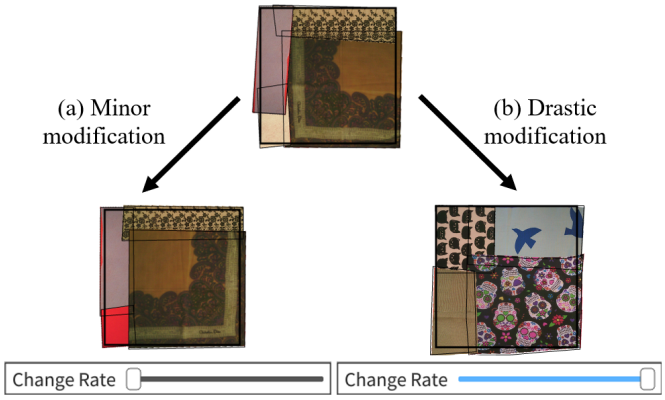


Figure 6: The *Change Rate* slider controls the extent to which the layout is modified after optimization. (a) Lower values maintain greater similarity to the current configuration, while (b) higher values promote more significant changes.

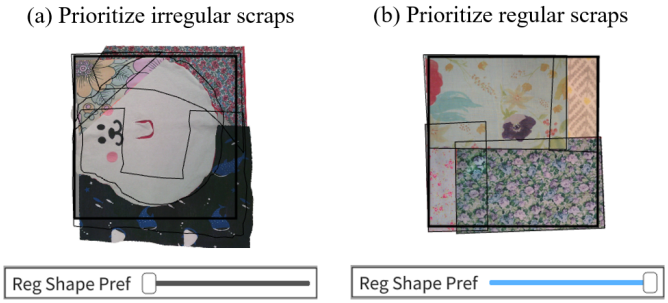


Figure 7: The *Reg Shape Pref* slider controls the type of scrap shape prioritized during optimization. (a) Lower settings favor irregular shapes with non-uniform edges, whereas (b) higher settings prioritizes regular shapes like rectangles.

3.2.1 Layout Workspace. The *Layout Workspace*, located at the top left of the *ScrapReCover* interface, is the main area for arranging scraps onto the pattern (Figure 4a). It visualizes the results of automatic layout suggestions while also supporting direct user interaction. Users can manually reposition any placed scrap, rotate it to any angle, or remove it by returning it to the *Available Scraps List* (Figure 4d). A preview pane (Figure 4b) provides an overview of the current layout, where placed regions are displayed in gray and uncovered areas are highlighted in red. When multiple pattern sheets are predefined, users can switch between them using the selector button (Figure 4c), with the active sheet indicated in orange.

3.2.2 Available Scraps List. The *Available Scraps List*, located at the top right of the *ScrapReCover* interface, displays unplaced scraps as thumbnail previews (Figure 4d). Each scrap is annotated with its current selection weight used in the optimization process. Users can manually place a specific scrap by selecting it (highlighted in green) and dragging it into the *Layout Workspace*.

3.2.3 Layout Control Panel. The *Layout Control Panel*, located at the bottom left of the *ScrapReCover* interface, provides a set of global controls for managing the layout. The *Iterations* slider (Figure 4e) specifies the approximate duration of the optimization, while the *Change Rate* slider (Figure 4f) adjusts the degree of modification applied to the current layout. The control buttons (Figure 4g) include the *Optimize* button, highlighted in yellow as a central feature of our system, which initiates automatic layout suggestions based on user-defined parameters.

3.2.4 Placed Scraps List. The *Placed Scraps List*, located at the bottom center of the interface, contains all scraps currently placed in the layout (Figure 4h). From this list, users can lock the position of a selected one, as if centering an emblem or logo.

3.2.5 Scrap Control Panel. The *Scrap Control Panel*, located at the bottom right of the interface, provides a set of controls for managing individual scrap. The *Selection Priority* slider (Figure 4i) determines how strongly the selected scrap in the *Available Scraps List* is prioritized during the optimization process. The *Reg Shape Pref* slider (Figure 4j) lets users choose whether to prioritize regular shapes such as rectangles, or irregular ones like polygons with non-uniform edges. The control buttons (Figure 4k) include *Load Scraps* and *Cut Scraps*, which launch the segmentation interface for importing new scraps or re-editing existing ones, and *Erase Scraps*, which removes the currently selected scrap from the system.

3.3 Interactive Optimization

3.3.1 Overview of the Layout Optimization Process. Figure 5 shows an example visualization of the optimization process for automatic layout suggestions in *ScrapReCover*. Initially, many scraps extend

beyond the pattern boundary, exhibit significant overlap, and a large number of polygons are placed. As the optimization progresses, the layout transitions to a state where most scraps are contained within the pattern, overlaps are visibly reduced, and the number of placed scraps decreases. In terms of optimization behavior, the layout changes substantially between iterations in the early stages, while such fluctuations gradually diminish over time, indicating convergence toward an approximate solution. Note that if the placement of certain scraps is locked via the *Placed Scraps List*, their occupied areas are excluded from the optimization target, and the remaining regions of the pattern are treated as the new effective boundary for layout computation. In cases where multiple pattern sheets are predefined in *ScrapReCover*, the optimization is applied only to the currently active sheet, while the layouts on non-active sheets remain fixed. Since each sheet functions independently, the layout task can be divided into smaller, more manageable units.

3.3.2 Change Rate Slider. Figure 6 illustrates how the *Change Rate* slider influences the behavior of the optimization process in *ScrapReCover*. This parameter enables users to adjust the extent to which the system modifies the current layout when generating a new one. A lower setting (Figure 6a) constrains the algorithm to minor adjustments, thereby supporting gradual refinements. In contrast, a higher setting (Figure 6b) permits more drastic and global changes, promoting the system to explore a wider variety of layout configurations. This flexibility is valuable for balancing layout stability with exploratory behavior during iterative design.

3.3.3 Reg Shape Pref Slider and Selection Priority Slider. Figure 7 illustrates how the *Reg Shape Pref* slider controls the type of scrap shapes prioritized during optimization in *ScrapReCover*. Lower values favor irregular shapes with non-uniform or curved edges (Figure 7a), while higher values prioritize regular shapes with orthogonal edges like squares or rectangles (Figure 7b). This global setting allows users to steer the optimization according to aesthetic preferences or practical considerations such as ease of sewing, thereby supporting a variety of design objectives. In addition to this global control, the system provides the *Selection Priority* slider for individual scraps. This parameter adjusts the likelihood of a specific scrap being selected during optimization. The combined use of both sliders enables users to guide the optimization toward using scraps that align with their design intentions, both in terms of shape characteristics and specific material selection.

3.4 Fabrication

Once the layout is finalized, *ScrapReCover* enables users to export the result in multiple formats to support real-world fabrication. The primary output is a high-resolution image of the layout, which can be printed and used as a visual reference during the assembly process. In addition, the system generates a structured list of the placed scraps, including their polygonal identifiers, center coordinates, rotation angles, and stacking order as an output. This metadata facilitates accurate reconstruction of the digital design in the real world. Fabrication methods may vary depending on the user's skill level and the complexity of the design. For example, users can follow traditional patchwork workflows by sewing the fabric pieces together according to the exported layout. Alternatively, they may opt for a simpler approach by directly adhering the scraps onto a base fabric using fusible interfacing or fabric glue. This flexibility supports a variety of fabrication practices, ranging from casual crafting to advanced textile craftsmanship.

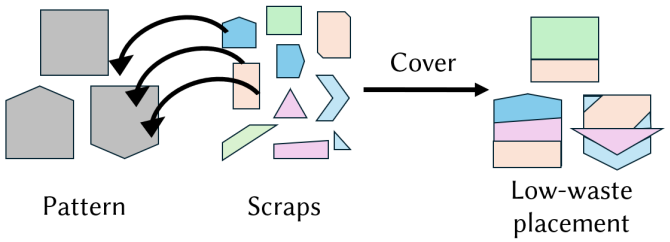


Figure 8: The overview of our problem statement

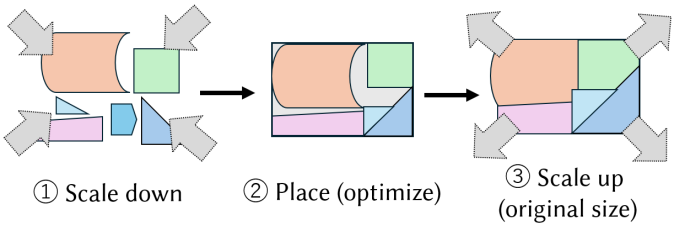


Figure 9: Preprocessing steps include shrinking scraps to reserve seam allowance for real-world fabrication.

4 Algorithm

4.1 Problem Statement

Before formalizing the problem, we collected approximately 300 real-world fabric scraps as part of this project. Our observations revealed a high degree of variability in their shapes, with no consistent patterns or clear tendencies. This highlights the need for a robust method capable of handling arbitrary input shapes.

As shown in Figure 8, the problem can be formulated as follows:

Input:

- **Scraps:** A limited number of distinct and irregular polygons (relatively small in size).
- **Pattern:** A limited number of distinct and irregular polygons (relatively large in size).

Output:

- Placement suggestion for a subset of scraps on the pattern (multiple suggestions are possible).

Problem Statement:

- Completely cover the pattern with a subset of scraps, allowing overlaps while minimizing the "waste."
- "Waste" is defined as the total sum of overlaps between scraps and the area of scraps extending outside the target region.

Constraint:

- All areas of the pattern must be fully covered.

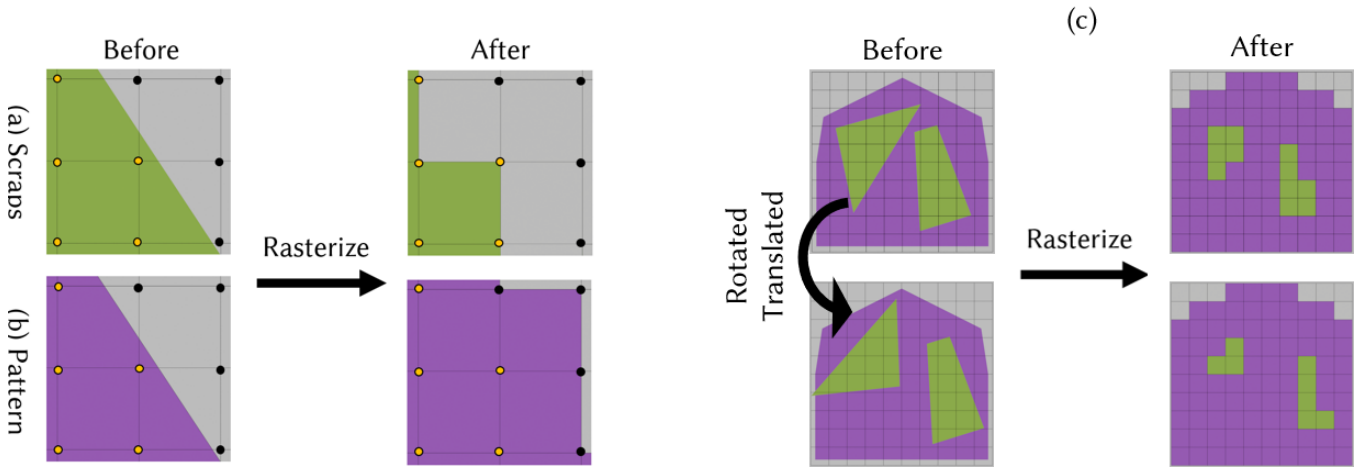


Figure 10: Both (a) scraps and (b) pattern are rasterized into 2D pixel arrays to enable efficient search operations. (c) Whenever the position or orientation of a scrap is modified, its rasterized representation is recomputed to ensure accurate processing within the discrete search space.

Formally, we can state the problem like:

$$\begin{aligned} &minimize \quad waste \\ &s.t. \quad \text{no uncovered area in the pattern} \end{aligned} \tag{1}$$

Note that in addressing this problem, we do not aim to compute a globally optimal solution, given the immense size of the solution space. Instead, our randomized algorithm supports multiple runs, each generating a plausible layout candidate. This approach enables users to flexibly select the most suitable configuration according to their individual preferences and design goals. Since *ScrapReCover* applies optimization only to the currently active pattern sheet, the following descriptions assume a single pattern sheet.

4.2 Preprocessing

To facilitate real-world sewing without relying on fabric glue, *ScrapReCover* automatically reserves seam allowances by slightly reducing the perimeter of each fabric scrap during preprocessing, prior to layout optimization. In the post-processing stage, when the layout is exported as an image, the scraps are restored to their original size (Figure 9). While these steps are not essential to the core optimization algorithm, it ensures that sufficient margin for seams is preserved regardless of how the scraps are arranged, enhancing usability in real-world fabrication.

Additionally, *ScrapReCover* employs a rasterization-based preprocessing step when placing scraps onto the pattern. This approach is motivated by the observation that, in real-world applications, pattern shapes can be treated as relatively large blocks measured in centimeters, making it reasonable to ignore minor geometric details. To discretize the problem and avoid the combinatorial explosion of possible placements in a continuous domain, each polygon is converted into a 2D array of pixels, as illustrated in Figure 10. To ensure precise coverage, two different rasterization criteria are applied. For scrap shapes (Figure 10a), a pixel is marked as occupied only if all four of its corners lie strictly within the scraps. In contrast, for pattern shapes (Figure 10b), a pixel is marked as occupied if at least one of its corners falls inside the pattern, excluding those that are exactly on the boundary. This geometric abstraction offers several advantages during optimization: it enables operations to be performed using integer values instead of floating-point numbers and simplifies tasks such as collision detection, placement validation, and other search operations. Note that whenever the position or orientation of a scrap is modified, its rasterized representation is recomputed to ensure accurate processing within the discrete search space (Figure 10c).

4.3 Layout Optimization

4.3.1 Preliminaries. For the optimization process, we employed simulated annealing (SA), a metaheuristic and probabilistic algorithm introduced by Kirkpatrick et al. [1983] (refer to *Appendix A* for details). As a metaheuristic, SA is not specialized for any particular problem domain, requiring appropriate configuration of the core functions and hyperparameters for the task [Blum and Roli 2003].

4.3.2 NEIGHBOR Function. The following five operations represent possible transitions that produce candidate states, as illustrated in Figure 11:

(a) Move: A placed scrap is randomly selected and translated by one pixel in one of the four cardinal directions (up, down, left, or right).
(b) Rotate: A placed scrap is randomly selected and rotated by an angle uniformly sampled from the interval $[0°, 360°)$. After rotation, the rasterized representation is updated accordingly.
(c) Swap: A placed scrap and an unplaced scrap are randomly selected; their positions are exchanged, and the newly placed polygon is rotated by a random angle in $[0°, 360°)$, followed by calculating its rasterized form.
(d) Erase: A placed scrap is randomly selected and removed from the layout.

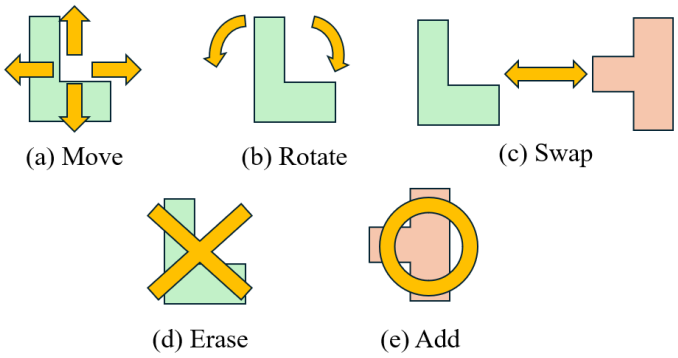


Figure 11: Our *NEIGHBOR* function definition. (a)–(e) illustrate candidate operations that may be applied to the current layout via a single transition.

(e) Add: An unplaced scrap is randomly selected, placed at a random location, and rotated by a random angle within $[0°, 360°)$, with its rasterized representation computed.

The selection probabilities for each operation were determined through hyperparameter tuning using Optuna [Akiba et al. 2019], conducted on a dataset of generated polygons described in *Appendix B*. The resulting probabilities were set to $p_{\text{move}} = 0.30$, $p_{\text{rot}} = 0.29$, $p_{\text{swap}} = 0.313$, $p_{\text{erase}} = 0.081$, and $p_{\text{add}} = 0.016$. The method for calculating the probability that each scrap is selected in these operations is described in Section 4.5.

4.3.3 EVAL Function. The *EVAL* function assigns a scalar score to each layout state based on three penalty components, each of which reflects a different type of undesirable configuration, as illustrated in Figure 12:

(a) Uncovered penalty (*uc*): Applies a penalty to each pixel within the pattern area that remains uncovered by any scrap. This term enforces the coverage constraint and is assigned a weight that is substantially larger than those of the other two penalties.

(b) Outrange penalty (*or*): Assigns a penalty to any part of a scrap that extends beyond the boundary of the predefined pattern, thereby discouraging out-of-bounds placements.

(c) Overlap penalty (*ol*): Applies a penalty to each pixel within the pattern area that is covered by more than one scrap, with the penalty increasing linearly according to the number of overlapping layers.

To enforce full coverage while minimizing material waste (which closely corresponds to the combination of the *outrange* penalty and the *overlap* penalty), we adopt an objective function that incorporates the coverage constraint as a penalty term, referred to as the *uncovered* penalty, following the concept of the penalty method [Boyd 2004]. Rather than directly minimizing material waste under a hard constraint, the *uncovered* penalty relaxes the coverage requirement into a soft constraint by assigning a significant cost to any uncovered regions. This encourages the optimization to avoid spending too much effort on infeasible configurations while retaining the flexibility to explore a broader solution space.

Throughout the optimization process based on this combined objective function, the algorithm searches for layout configurations

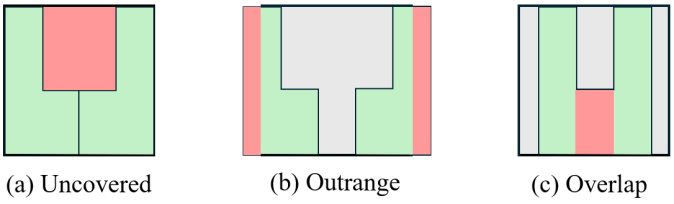


Figure 12: Three penalties that reflect different types of undesirable placements are incorporated into the *EVAL* function: (a) *uncovered* penalty, (b) *outrange* penalty, and (c) *overlap* penalty. In each illustration, green shapes represent placed scraps on a gray pattern area, and red regions indicate pixels where penalties are applied.

that fully cover the target pattern. Among those that satisfy the full coverage condition, the system selects as the final output the configuration with the lowest actual material waste, where the waste is computed independently of the combined objective function. It is important to note that the layout with the lowest evaluation score based on this combined objective function does not necessarily correspond to the layout with the lowest actual waste, and full coverage is not always guaranteed during the search. Therefore, *ScrapReCover* monitors not only the evaluation score but also the actual material waste and the number of uncovered pixels at each iteration. The system selects the final result from among the feasible candidates that achieve full coverage in a single run, prioritizing the one that minimizes material waste. This approach ensures both constraint satisfaction and efficient material usage.

The formal definition of the *EVAL* function is presented as follows (each penalty term is abbreviated as *uc*, *or*, and *ol*, respectively):

$$\text{EVAL}(State) = \sum_{(x,y)} \Big(\mathcal{L}_{\text{uc}}(x,y) + \mathcal{L}_{\text{or}}(x,y) + \mathcal{L}_{\text{ol}}(x,y) \Big) \tag{2}$$

where

$$\texttt{count}[x][y] = \text{number of scraps placed at } (x,y), \tag{3}$$

$$\mathcal{L}_{\text{uc}}(x,y) = \begin{cases} w_{\text{uc}}, & \text{if } (x,y) \in \text{pattern} \wedge \texttt{count}[x][y] = 0, \\ 0, & \text{otherwise}, \end{cases} \tag{4}$$

$$\mathcal{L}_{\text{or}}(x,y) = \begin{cases} w_{\text{or}} \cdot \texttt{count}[x][y], & \text{if } (x,y) \notin \text{pattern}, \\ 0, & \text{otherwise}, \end{cases} \tag{5}$$

$$\mathcal{L}_{\text{ol}}(x,y) = \begin{cases} w_{\text{ol}} \cdot \big(\texttt{count}[x][y] - 1\big), & \text{if } (x,y) \in \text{pattern} \wedge \\ & \quad \texttt{count}[x][y] \neq 0, \\ 0, & \text{otherwise}. \end{cases} \tag{6}$$

The value of penalty weights were determined through hyperparameter tuning using Optuna [Akiba et al. 2019], conducted on a dataset of generated polygons described in *Appendix B*. The final values were set as follows: $w_{\text{uc}} = 13600$, $w_{\text{or}} = 250$, and $w_{\text{ol}} = 100$.

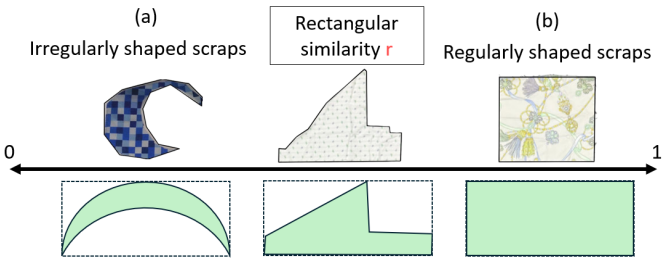


Figure 13: The rectangular similarity r is computed as the ratio of a scrap's area to that of its minimum bounding box. (a) Irregular scraps with non-uniform edges typically have low r values close to 0, whereas (b) regular shapes like rectangles tend to produce high r values close to 1.

4.4 Layout Modification Control

The *TEMP* and *ACCEPT* functions jointly govern the transition behavior in simulated annealing by controlling whether the algorithm accepts a proposed layout state. When a candidate layout improves the evaluation score, it is always accepted. Otherwise, acceptance occurs with a probability that decreases as the score degradation increases and as the optimization progresses. We adopt widely used formulations for both *TEMP* [Nourani and Andresen 1998] and *ACCEPT* function [Rutenbar 1989]:

$$\mathrm{TEMP}(iter, maxIter) = T_{start} + (T_{end} - T_{start}) \times \frac{iter}{maxIter} \quad \text{[Linear Decay]} \tag{7}$$

$$\mathrm{ACCEPT}(\Delta Score, \mathrm{TEMP}(iter, maxIter)) = \min\left(1,\ \exp\left(-\frac{\Delta Score}{\mathrm{TEMP}(iter, maxIter)}\right)\right) \tag{8}$$

In *ScrapReCover*, this mechanism is utilized to support user-driven design exploration. The initial temperature T_{start} is directly linked to the *Change Rate* slider in the interface, allowing users to influence the degree of layout modification during optimization. Higher values of the slider result in greater structural changes to the layout, encouraging more radical reconfigurations, whereas lower values constrain the system to make smaller, more stable adjustments. This coupling provides an interpretable mapping between algorithmic control and real-world user interactions. The default values are set to $T_{\text{start}} = 17500$ and $T_{\text{end}} = 5$.

4.5 Selection Weight for Scraps

ScrapReCover defines a *selection weight* w for each scrap so that users can flexibly guide the optimization process according to their preferences for specific shapes or scraps. This weight determines how likely each scrap is to be selected during layout generation and is influenced by two factors: the global preference for shape regularity and an individually assigned selection priority.

In order to quantify the geometric regularity of each scrap, we define a metric called *rectangular similarity* $r \in (0, 1]$, computed as the ratio between the area of the scrap and the area of its minimum bounding box. Specifically, each scrap is rotated exhaustively in small angular increments (we set this to 1°), and the maximum occupancy rate observed over all rotations is used as its r value. As a result, irregularly shaped scraps with non-uniform or high-curvature edges tend to have low r values close to 0 (Figure 13a). On the other hand, regularly shaped scraps with straight and orthogonal edges produce higher values near 1. To incorporate user preferences regarding shape regularity, a global parameter called the *Regular-shape Preference* $n \in (-\infty, \infty)$ is introduced via the *Reg Shape Pref* slider. This parameter modulates how strongly the system favors regular or irregular shapes. When n is large and positive, scraps with high r are heavily favored since r^n becomes much larger. Conversely, negative values of n favor scraps with lower r, as r^n becomes relatively larger for irregular shapes. If $n = 0$, no shape preference is applied since $r^n = 1$ for all r.

In addition to this global shape preference, users can assign *selection priority* $s \geqq 0$ to individual scraps using the *Selection Priority* slider. This parameter adjusts the likelihood of a specific scrap being selected in the layout, regardless of its shape.

The final selection weight w for each scrap is computed as:

$$w = s \cdot r^n$$

where s is the individual selection priority and r^n reflects the effect of the global shape preference. This formulation provides a flexible and interpretable way for users to control both the geometric style and specific material usage in their designs.

During the optimization process, these weights are integrated into the sampling steps of the *NEIGHBOR* function, specifically in the *swap*, *erase*, and *add* operations (see Section 4.3.2). For unplaced scraps, the computed weight w is directly used to increase their likelihood of being placed. For scraps already placed, the reciprocal value $(1/w)$ is used to reduce their probability of being removed. Here, all weights are normalized to form a valid probability distribution. This weighting mechanism enables *ScrapReCover* to produce layouts that reflect both global shape preferences and user-driven priorities for individual scraps.

Figure 14: In the workshop, each participant brought their own fabric scraps, and interactively designed patchwork layouts using *ScrapReCover*.

5 Evaluation

5.1 User Study

We conducted a workshop in which each participant brought their own fabric scraps and interactively created patchwork layouts using the prototype of *ScrapReCover* (Figure 14). For the user study, we intentionally selected a simple square pattern to help participants

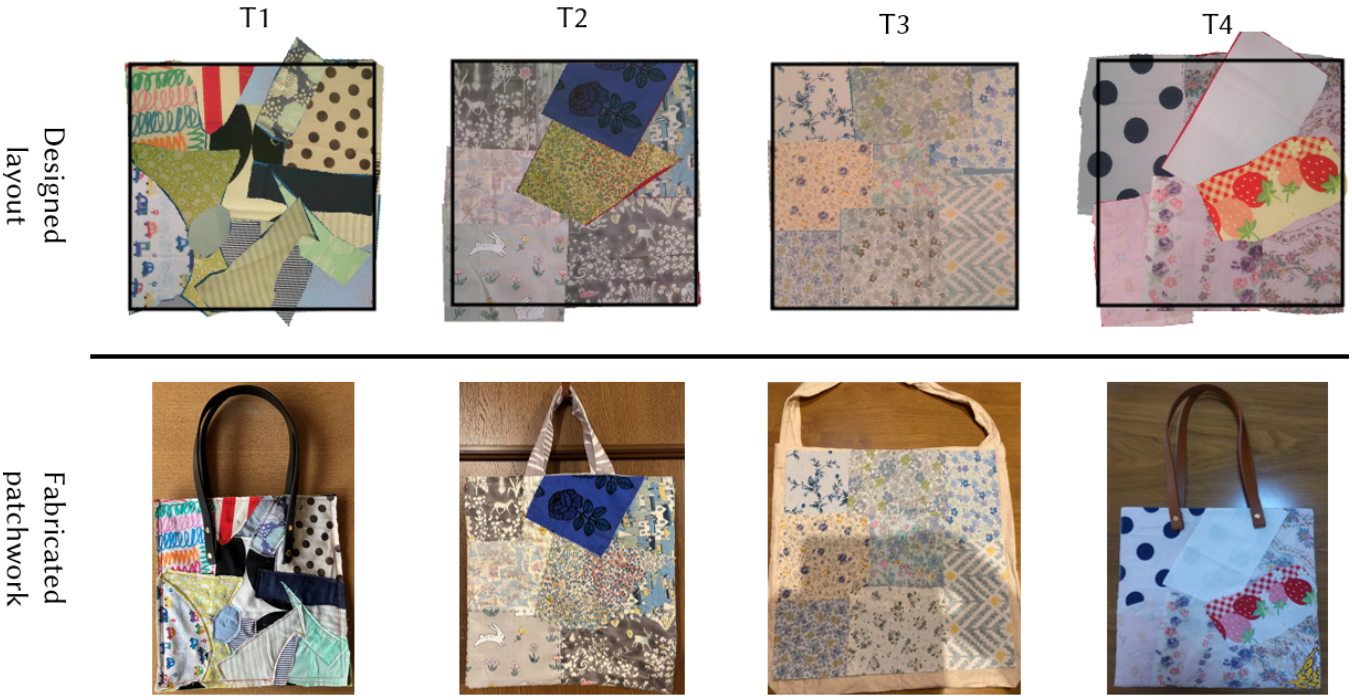


Figure 15: Four examples from workshop participants who designed layouts using *ScrapReCover* (top) and subsequently fabricated a tote bag in real based on their designs (bottom).

Table 1: Responses from 21 participants in the user study using a five-point scale (1 = strongly disagree, 5 = strongly agree).

Survey Statement	1	2	3	4	5
(1) I found the system easy to use.	1	2	1	15	2
(2) The optimization-based layout was better than manual placement.	0	2	7	8	4
(3) I was satisfied with the design suggestions generated by the system.	0	2	11	4	4

focus on the design process itself while producing a practical object (a tote bag), and to emphasize the evaluation of the usability and flexibility of the system, as well as the resulting designs.

At the beginning of the session, we provided an overview of the study and explained how to use *ScrapReCover*. The participants were instructed to design a layout for a square tote bag with a size of $30 \times 30cm$, and we provided a reference fabric sheet to indicate the target size. The participants were then given 50 minutes to work on their designs, which included photographing their scraps using a fixed camera and using the segmentation interface to define the regions they wished to use from each scrap. During this time, they freely explored the features of the system, composing layouts through a combination of manual placement and automatic layout suggestions generated by the optimization algorithm, using their own fabric materials. Following the workshop, participants were asked to complete a survey about their experience with the system and were compensated with a 1000 yen (≒ 7 dollars) Amazon gift card. Each session of the workshop accommodated up to three groups simultaneously, and a total of 21 participants, recruited in advance from the general public, took part in the entire workshop. The participants exhibited a wide range of experiences, both in terms of patchwork creation and in operating a design system.

Table 1 presents the results of a post-workshop questionnaire in which 21 participants rated three aspects of the system using a five-point scale (1 = very low, 5 = very high). Participants were asked to evaluate (1) the overall usability of the system, (2) how well the optimization-based layout suggestions performed compared to manual placement, and (3) how satisfied they were with the design proposals generated by the system. In addition, there were open-ended questions addressing topics such as the pros and cons of each survey statement and suggestions for new features, with the aim of highlighting both the strengths of the system and the potential direction for future improvement.

Regarding usability, to the question (*"Did you find the system easy to use?"*), the majority of participants responded positively. Specifically, 15 out of 21 participants assigned a score of 4, indicating a general consensus that the system was easy to use. In contrast, responses to the usefulness of the optimization-based layout compared to manual placement (*"Was the optimization-based layout better than manual placement?"*) were more varied. While 7

(a) Circular pattern (b) T-shirt pattern (c) Donut pattern

Figure 16: The user can predefine any kind of target pattern, as demonstrated in examples (a)–(c).

participants gave a neutral score of 3, 12 rated it positively (scores of 4 or 5), suggesting differences in users' preferences for manual control versus automation.

For the question *"Please describe any parts of the system you found especially easy to use"*, participants highlighted several aspects [1]. P10 and P11 noted that the task of arranging the layout was straightforward, and that watching the optimization process in progress was enjoyable (e.g., *"The placement was done easily, and I enjoyed watching it on the screen"* (P10)). P2, P12, and P13 appreciated the interface as user-friendly, describing it as accessible even to children or beginners (e.g., *"It was nice that the automatic placement button seemed applicable and useful. Even children were able to operate it"* (P13)). These responses, along with the overall positive usability ratings, indicate the intuitiveness of the system. P7, P9, P15, and P16 emphasized the value of having an initial layout suggestion when it can be difficult to generate ideas or decide on piece placement (e.g., *"Since it is quite difficult to come up with everything from scratch, I appreciated that the system could generate something to start with"* (P15)). This underscores the benefit of combining manual and automatic methods. P4 and P5 praised the workflow for defining scrap contours, describing it as intuitive and effective in capturing fabric shapes (e.g., *"I thought it was good that plotting allowed the fabric shape to be captured quite accurately"* (P4)). Their feedback highlights the usefulness of manual contour registration as an accurate and flexible tool.

Regarding design quality, responses were mixed, with most participants assigning mid-range scores. This suggests a need for further refinement, particularly in how the system adapts layout suggestions to users' individual design preferences. When asked *"What other elements do you think could be considered to make the proposed layout better?"*, participants proposed several directions for improvement. P1, P16, P20, and P21 recommended incorporating explicit design or color guidance (e.g., *"I would like the system to also recognize colors and make design suggestions [...] It would be nice if styles such as cute or mature could be added as options"* (P1)). P3 and P14 highlighted the importance of considering color combinations to enhance the aesthetic quality of the layout (e.g., *"Patterns are not taken into account. The appearance of colors and patterns is hard to interpret"* (P14)). Similar suggestions emerged in response to *"Please describe any features you would like to see added to the system."* P1, P3, and P14 requested optimization features that take color coordination into account (e.g., *"When characters are present, select the face area to ensure it remains visible. Unify the overall color scheme"* (P14)). P15 and P21 proposed support for language-based input to

[1]The Japanese feedback was translated into English using ChatGPT.

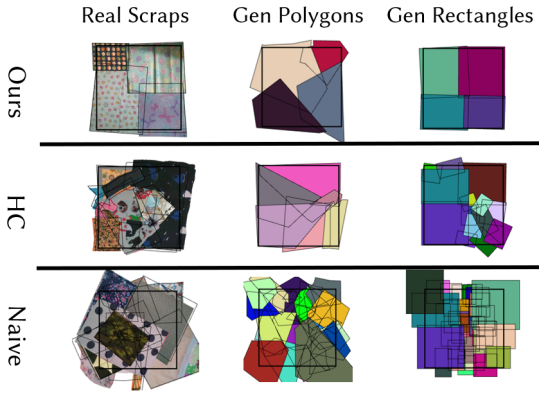


Figure 17: Visualization of layout results generated by our SA-based method, hill climbing (HC), and a naive random allocation baseline across three datasets. Our method achieves more densely packed layouts compared to the alternatives.

articulate design intentions (e.g., *"Sometimes the optimization results looked plain, or the accents ended up at the edges, so it would be nice if I could provide some initial input, like 'I want it to look this way'"* (P21)). Collectively, this feedback suggests a demand for integrating design considerations into the system, indicating a direction for future work. Other participants proposed enhancements related to system flexibility. P9 suggested that generating a rough layout outline at the scrap registration could improve efficiency. This preference contrasts with those who appreciated the ability to define contours manually (P4 and P5), indicating the potential benefit of offering manual contouring as an optional feature rather than a requirement. P10–P12, P18, and P19 requested the functionality to interrupt the optimization process midway (e.g., *"I couldn't manage to stop the optimization at the moment I felt, 'This is it!' with a design"* (P12)). This feedback suggests that intermediate results during optimization may, in some cases, be preferable to users compared to the final optimized outcome.

5.2 Visual Examples

5.2.1 Fabricated Product. Participants who wished to fabricate their resulting layouts into physical tote bags were provided with a full-scale printed version of their design, along with supplementary materials such as backing fabric and straps to support home fabrication. Figure 15 presents four examples from workshop participants who created tote bags (bottom) based on layouts designed using *ScrapReCover* (top). Participants employed different fabrication methods: some participants (T1, T2) fabricated their designs by sewing the fabric scraps together into a single fabric piece, while others (T3, T4) created their tote bags by pasting the scraps onto a base fabric. These completed artifacts demonstrate the practical usability of our system in supporting real-world creation.

5.2.2 Pattern Shape Variation. *ScrapReCover* supports arbitrary pattern shapes through the rasterization-based preprocessing described in Section 4.1. Figure 16 illustrates its application to three representative non-square patterns.

5.3 Algorithm Evaluation

Algorithm 1 Naive Random Allocation of Scraps

```
PDF ← Uniform 2D probability density map over pattern
while there are empty regions in pattern do
    position(x, y) ← sample from PDF
    Place a scrap at position(x, y)
    Set PDF[i][j] = 0 for all (i, j) covered by the placed scraps
    Normalize PDF
end while
```

5.3.1 Performance. We conducted quantitative experiments using a single 30×30 square pattern, which matches the size and shape used in the workshop. As a case study, we used either the scrap dataset collected during the workshop (composed of 431 scraps in total) or a set of randomly generated polygons to ensure robustness. The method for generating these polygons is described in *Appendix B*. In this section, we omit the scrap shrinking and resizing operations described in the preprocessing step (Section 4.1), as they do not affect the comparative results. The number of iterations was fixed at 50,000, and the optimization process was repeated 100 times for each dataset with different random seeds, after which the resulting waste values were compared. The average execution time for one optimization run using the real-world scrap dataset was approximately 28 seconds on a machine with an Intel(R) Core(TM) Ultra 9 285K 3.70 GHz CPU and 32 GB of RAM. (Note that this value also depends on the number and complexity of input scraps and the predefined pattern.)

5.3.2 Comparison with Naive Methods. We compared our randomized algorithm with a naive one, described in Algorithm 1, which repeatedly places scraps at random until no empty areas remain in the pattern (this method was also used as the initial state for the other two comparison algorithms). In this approach, the algorithm attempts to avoid already occupied regions as much as possible during scrap placement. Moreover, we also compared our method with the hill climbing (HC) approach [Selman and Gomes 2006], which was implemented by modifying our algorithm to allow state transitions only when the evaluation score improves. We chose HC because most previous approaches do not align with our goal of enabling diverse outputs through a randomized algorithm with tunable parameters. For comparison, in addition to the full dataset of real scraps collected through the workshop (*Real Scraps*), we also used two virtual scrap sets generated based on the method described in *Appendix B*: one set consisting of 100 randomly generated polygons (*Gen Polygons*), and another set consisting of 100 perfect rectangles (*Gen Rectangles*).

The results are visualized in Figure 17 and summarized using box plots in Figure 18 and tables in Table 2. These findings confirm that our SA-based approach not only produces more densely packed and visually refined configurations but also consistently achieves lower average waste scores than the other two methods across all three datasets. Furthermore, our method yields noticeably smaller standard deviations, indicating more stable and reliable performance. These advantages hold not only for simpler geometric cases, such as the rectangles-only dataset (*Gen Rectangles*), but also for more complex and irregular scrap shapes, demonstrating the robustness and adaptability of our approach across varying input types.

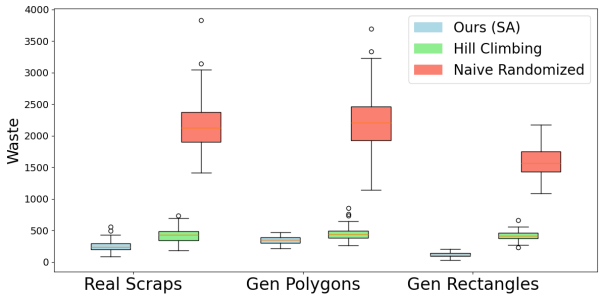


Figure 18: Comparison between our SA-based method, hill climbing (HC), and a naive random allocation baseline across three datasets. In all datasets, our method outperformed the others in reducing waste.

6 Conclusion and Limitations

We introduced *ScrapReCover*, an interactive design tool for generating freeform patchwork layouts from fabric scraps by formulating and solving a geometric covering problem using a simulated annealing-based optimization method. The system allows users to iteratively steer the randomized layout suggestions process through intuitive parameter adjustments, seamlessly integrating automatic optimization with manual refinement. A user study involving 21 participants demonstrated the practical usability of *ScrapReCover*, with users finding it both accessible and effective for creating original designs. Quantitative comparisons further confirmed that our optimization algorithm outperforms naive baselines by achieving superior visual quality and reducing material waste more effectively.

Meanwhile, several limitations remain in our current algorithm, which offer opportunities for future work. First, incorporating 3D constraints and enabling simultaneous modification of the pattern itself, which is currently fixed, could better take advantage of the characteristics of the problem. Second, allowing users to specify design preferences or define color combinations prior to the optimization process could enhance both the aesthetic and functional aspects of the patchwork. Lastly, incorporating practical aspects related to fabrication difficulty in real, such as explicitly considering seam length or the curvature of the scraps, could enhance the feasibility of the designs generated by the system.

Acknowledgments

This research is part of the results of Value Exchange Engineering, a joint research project between Mercari R4D Lab and RIISE (Research Institute for an Inclusive Society through Engineering), Japan Science and Technology Agency (JST) ASPIRE grant number JPMJAP2401, and Japan Society for the Promotion of Science (JSPS) KAKENHI grant number JP23K19994. We also sincerely thank Ochanomizu University and sciencepark.jp for their support in conducting the user study, and Siv3D for providing the development framework.

Table 2: Summary statistics of the resulting waste, comparing our SA-based method, hill climbing (HC), and a naive random allocation baseline across three datasets: real scraps, generated polygons, and generated rectangles.

	Real Scraps			Gen Polygons			Gen Rectangles		
	Naive	**HC**	**Ours**	**Naive**	**HC**	**Ours**	**Naive**	**HC**	**Ours**
Mean	2154.58	421.49	246.91	2241.17	442.07	344.64	1582.16	418.20	116.54
Med.	2123.00	429.50	236.00	2201.50	435.50	348.50	1564.00	416.00	112.00
Std.	387.42	104.87	83.05	443.31	100.95	60.09	212.88	71.79	36.39
Min	1416.00	179.00	81.00	1143.00	257.00	211.00	1086.00	226.00	32.00
Max	3826.00	730.00	555.00	3695.00	854.00	470.00	2170.00	658.00	208.00

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2623–2631.

Berend Baas, David Bommes, and Adrien Bousseau. 2025. Shape Approximation by Surface Reuse. In *Computer Graphics Forum*, Vol. 44. Wiley Online Library, e70204.

AL Bajuelos, Ana Mafalda Martins, S Canales, and G Hernández. 2009. Metaheuristic approaches for the minimum vertex guard problem. In *2009 Third International Conference on Advanced Engineering Computing and Applications in Sciences*. IEEE, 77–82.

Anton Bakker and Tom Verhoeff. 2022. Algorithms to Construct Designs for Foundation Paper Piecing of Quilt Patchwork Layers. In *25th Annual Bridges Conference 2022: Mathematics, Art, Music, Architecture, Cullture*. Tessellations Publishing, 347–350.

Julia A Bennell and Jose F Oliveira. 2008. The geometry of nesting problems: A tutorial. *European journal of operational research* 184, 2 (2008), 397–415.

Julia A Bennell and Xiang Song. 2010. A beam search implementation for the irregular shape packing problem. *Journal of Heuristics* 16 (2010), 167–188.

Christian Blum and Andrea Roli. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)* 35, 3 (2003), 268–308.

Ralf Borndörfer. 1998. *Aspects of set packing, partitioning, and covering*. Ph. D. Dissertation.

Stephen Boyd. 2004. Convex optimization. *Cambridge UP* (2004).

Chazelle. 1983. The bottomn-left bin-packing heuristic: An efficient implementation. *IEEE transactions on computers* 100, 8 (1983), 697–707.

Vasek Chvatal. 1979. A greedy heuristic for the set-covering problem. *Mathematics of operations research* 4, 3 (1979), 233–235.

Claudio Contardo and Alain Hertz. 2021. An exact algorithm for a class of geometric set-cover problems. *Discrete Applied Mathematics* 300 (2021), 25–35.

Joseph C Culberson and Robert A Reckhow. 1994. Covering polygons is hard. *J. Algorithms* 17, 1 (1994), 2–44.

Mark De Berg. 2000. *Computational geometry: algorithms and applications*. Springer Science & Business Media.

Kathryn A Dowsland. 1993. Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research* 68, 3 (1993), 389–399.

Harald Dyckhoff. 1990. A typology of cutting and packing problems. *European journal of operational research* 44, 2 (1990), 145–159.

Robert J Fowler, Michael S Paterson, and Steven L Tanimoto. 1981. Optimal packing and covering in the plane are NP-complete. *Information processing letters* 12, 3 (1981), 133–137.

Deborah S Franzblau and Daniel J Kleitman. 1984. An algorithm for covering polygons with rectangles. *Information and control* 63, 3 (1984), 164–189.

A Miguel Gomes and José F Oliveira. 2006. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research* 171, 3 (2006), 811–829.

Clement Greenberg. 2018. Collage. In *Modern Art and Modernism*. Routledge, 105–108.

Baosu Guo, Yu Zhang, Jingwen Hu, Jinrui Li, Fenghe Wu, Qingjin Peng, and Quan Zhang. 2022. Two-dimensional irregular packing problems: A review. *Frontiers in Mechanical Engineering* 8 (2022), 966691.

E Hopper and B Turton. 1999. A genetic algorithm for a 2D industrial packing problem. *Computers & Industrial Engineering* 37, 1-2 (1999), 375–378.

Eva Hopper and Brian CH Turton. 2001. A review of the application of meta-heuristic algorithms to 2D strip packing problems. *Artificial Intelligence Review* 16 (2001), 257–300.

Chi-Fu Huang and Yu-Chee Tseng. 2003. The coverage problem in a wireless sensor network. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*. 115–121.

Yuki Igarashi and Jun Mitani. 2015. Patchy: An interactive patchwork design system. In *ACM SIGGRAPH 2015 Posters*. 1–1.

David S Johnson. 1973. *Near-optimal bin packing algorithms*. Ph. D. Dissertation. Massachusetts Institute of Technology.

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

Bongjin Koo, Jean Hergel, Sylvain Lefebvre, and Niloy J Mitra. 2016. Towards zero-waste furniture design. *IEEE transactions on visualization and computer graphics* 23, 12 (2016), 2627–2640.

VS Anil Kumar and H Ramesh. 2003. Covering rectilinear polygons with axis-parallel rectangles. *SIAM J. Comput.* 32, 6 (2003), 1509–1541.

Kin Chung Kwan, Lok Tsun Sinn, Chu Han, Tien-Tsin Wong, and Chi-Wing Fu. 2016. Pyramid of arclength descriptor for generating collage of shapes. *ACM Trans. Graph.* 35, 6 (2016), 229–1.

Mackenzie Leake, Gilbert Bernstein, and Maneesh Agrawala. 2022. Sketch-Based Design of Foundation Paper Pieceable Quilts. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–11.

Mackenzie Leake and Ross Daly. 2024. ScrapMap: Interactive Color Layout for Scrap Quilting. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–17.

Mackenzie Leake, Frances Lai, Tovi Grossman, Daniel Wigdor, and Ben Lafreniere. 2021. Patchprov: Supporting improvisational design practices for modern quilting. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–17.

Yifei Li, David E Breen, James McCann, and Jessica K Hodgins. 2019. Algorithmic Quilting Pattern Generation for Pieced Quilts.. In *Graphics Interface*. 13–1.

Max Limper, Nicholas Vining, and Alla Sheffer. 2018. Box cutter: atlas refinement for efficient packing via void elimination. *ACM Trans. Graph.* 37, 4 (2018), 153.

Chenxi Liu, Jessica Hodgins, and James McCann. 2017. Whole-cloth quilting patterns from photographs. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*. 1–8.

Jon McCormack and Monika Schwarz. 2024. Piecing Generative Patterns Into Contemporary Quilts. (2024).

Holly McQuillan. 2020. Digital 3D design as a tool for augmenting zero-waste fashion design practice. *International Journal of Fashion Design, Technology and Education* 13, 1 (2020), 89–100.

Holly McQuillan, Jen Archer-Martin, Greta Menzies, Jo Bailey, Karl Kane, and E Fox Derwin. 2018. Make/Use: A system for open source, user-modifiable, zero waste fashion practice. *Fashion Practice* 10, 1 (2018), 7–33.

Jiří Minarčík, Sam Estep, Wode Ni, and Keenan Crane. 2024. Minkowski penalties: Robust differentiable constraint enforcement for vector graphics. In *ACM SIGGRAPH 2024 Conference Papers*. 1–12.

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

JF Oliveira, AM Gomes, and JS Ferreira. 2000. TOPOS – A new constructive algorithm for nesting problems. *OR SPEKTRUM* 22 (2000), 263–284.

Anran Qi, Nico Pietroni, Maria Korosteleva, Olga Sorkine-Hornung, and Adrien Bousseau. 2025. Rags2Riches: Computational Garment Reuse. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*. 1–11.

Rob A Rutenbar. 1989. Simulated annealing algorithms: An overview. *IEEE Circuits and Devices magazine* 5, 1 (1989), 19–26.

Daniel Saakes, Thomas Cambazard, Jun Mitani, and Takeo Igarashi. 2013. PacCAM: material capture and interactive 2D packing for efficient material usage on CNC cutting machines. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. 441–446.

Bart Selman and Carla P Gomes. 2006. Hill-climbing search. *Encyclopedia of cognitive science* 81, 333-335 (2006), 10.

Ticha Sethapakdi, Daniel Anderson, Adrian Reginald Chua Sy, and Stefanie Mueller. 2021. Fabricaide: Fabrication-aware design for 2d cutting machines. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–12.

Algorithm 2 Simulated Annealing

```
curState ⟵ InitialState
curScore ⟵ EVAL(InitialState)
bestState ⟵ curState
bestScore ⟵ curScore
for iter = 1 to maxIter do
    nextState ⟵ NEIGHBOR(curState)
    nextScore ⟵ EVAL(nextState)
    ΔScore ⟵ nextScore − curScore
    if ACCEPT(ΔScore, TEMP(iter, maxIter)) then
        curState ⟵ nextState
        curScore ⟵ nextScore
        if curScore is better than bestScore then
            bestState ⟵ curState
            bestScore ⟵ curScore
        end if
    end if
end for
return bestState
```

Algorithm 3 Non-Convex Polygon Generation

Input: $v_{\min}, v_{\max}, r_{\text{mean}}, r_{\text{var}}$
Output: List of 2D coordinates forming a non-convex polygon

$v \leftarrow \text{RandomInt}(v_{\min}, v_{\max})$
$[r_{\min}, r_{\max}] \leftarrow \text{minmax}(\text{Normal}(r_{\text{mean}}, r_{\text{var}}, 2))$
$R \leftarrow \text{Uniform}(r_{\min}, r_{\max}, v)$ *// radii*
$\Theta \leftarrow \text{Uniform}(0, 2\pi, v)$ *// arguments (angles)*
Sort Θ *// to ensure polygon is non-intersecting*
for $i = 1$ **to** v **do**
    $x_i \leftarrow R[i] \cdot \cos(\Theta[i])$
    $y_i \leftarrow R[i] \cdot \sin(\Theta[i])$
    $P[i] \leftarrow (x_i, y_i)$
end for
return P

Musashi Shinjo, Maria Larsson, and Hironori Yoshida. 2024. A Design and Fabrication Workflow for Upcycling Leftover Fabrics as Mosaic Art. (2024).

Nadav Shragai and Gershon Elber. 2013. Geometric covering. *Computer-Aided Design* 45, 2 (2013), 243–251.

Ryo Suzuki. 2020. Siv3D: C++ Library for creative coding.

Alexander Teibrich, Stefanie Mueller, François Guimbretière, Robert Kovacs, Stefan Neubert, and Patrick Baudisch. 2015. Patching physical objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. 83–91.

Kevin Tole, Rashad Moqa, Jiongzhi Zheng, and Kun He. 2023. A simulated annealing approach for the circle bin packing problem with rectangular items. *Computers & Industrial Engineering* 176 (2023), 109004.

Ludwig Wilhelm Wall, Alec Jacobson, Daniel Vogel, and Oliver Schneider. 2021. Scrappy: Using scrap material as infill to make fabrication more sustainable. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–12.

Ruowang Zhang, Stefanie Mueller, Gilbert Louis Bernstein, Adriana Schulz, and Mackenzie Leake. 2024. Wastebanned: Supporting zero waste fashion design through linked edits. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–13.

A Simulated Annealing

Simulated annealing (SA) is a metaheuristic and probabilistic algorithm introduced by Kirkpatrick et al. [1983]. It improves upon the basic hill climbing (HC) method, which accepts transitions to neighboring states only if they yield a better score [Selman and Gomes 2006]. Unlike HC, SA also allows transitions to worse states with a probability that gradually decreases over time, enabling the algorithm to escape local optima and explore a broader solution space. The specific procedure is outlined in Algorithm 2 [Rutenbar 1989]. At each iteration, a candidate state is generated by the *NEIGHBOR* function and evaluated using the *EVAL* function. The decision to transition is then determined by the *ACCEPT* function, which always approves better states and probabilistically accepts worse states based on the current temperature defined by the *TEMP* function, often referred to as the *cooling schedule*.

B Polygon Generation Method

The method used to generate polygons, which are not necessarily convex (corresponding to scraps), is outlined in Algorithm 3.

In practice, each variable was assigned a value corresponding to the intended size of the pattern. For the experiments reported in *Section 6.1*, where the pattern was a 30×30 square, we used the following settings: $v_{\min} = 4$, $v_{\max} = 10$, $r_{\text{mean}} = 15$ for large polygons, $r_{\text{mean}} = 7.5$ for small polygons, and $r_{\text{var}} = 2.5$ for both.

To prevent degenerate outputs, we imposed additional validity checks. Specifically, polygons with bounding boxes narrower than a certain threshold (e.g., width or height < 2.5 for large shapes and < 1.0 for small shapes) are rejected. Furthermore, any polygon not containing the origin is discarded to avoid exceptions.

For rectangle generation, the height and width were sampled independently from a uniform distribution. Large rectangles used the range $[5.0, 20.0)$, and small rectangles used $[5.0, 10.0)$. All rectangles were axis-aligned and centered at the origin.

Each shape was finally assigned a randomly sampled color from a uniform RGB distribution over $[0, 255]^3$. For each experiment, we generated a dataset of 100 shapes, evenly divided into 50 large and 50 small instances.

C Ablation Study on Layout Optimization

C.1 The effectiveness of NEIGHBOR function

Table 3: Ablation study of the *NEIGHBOR* function. Each row shows the statistics when a specific operation is excluded.

	Mean	Med.	Std.	Min	Max
Baseline	246.91	236.00	83.05	81.00	555.00
No Move	343.91	338.50	101.34	158.00	714.00
No Rotate	277.96	273.50	78.78	85.00	525.00
No Swap	428.11	416.50	126.66	120.00	703.00
No Add	240.38	242.50	95.55	83.00	435.00
No Erase	1559.07	1566.50	257.99	941.00	2183.00

As shown in Table 3 and Figure 19, we conducted an ablation study using the real scraps dataset to evaluate the contribution of each operation in the *NEIGHBOR* function, defined in *Section 4.3.1*. Excluding each operation in turn, we observed that *Erase* is indispensable; its removal caused a dramatic increase in mean waste

Table 4: Summary statistics for the ablation study by scaling parameters Unplaced, Outrange, and Overlap penalty in *EVAL* function. Each parameter was scaled by ×10 and ÷10. The baseline corresponds to the default setting.

	Baseline	Uncovered		Outrange		Overlap	
		×10	÷10	×10	÷10	×10	÷10
Mean	246.91	1730.00	2056.09	1241.52	370.48	1358.90	281.27
Med.	236.00	1818.00	2039.00	1360.50	339.50	1678.50	285.50
Std	83.05	639.29	400.63	852.39	118.13	914.86	85.13
Min	81.00	105.00	182.00	171.00	86.00	122.00	75.00
Max	555.00	3162.00	3140.00	2768.00	665.00	2986.00	489.00

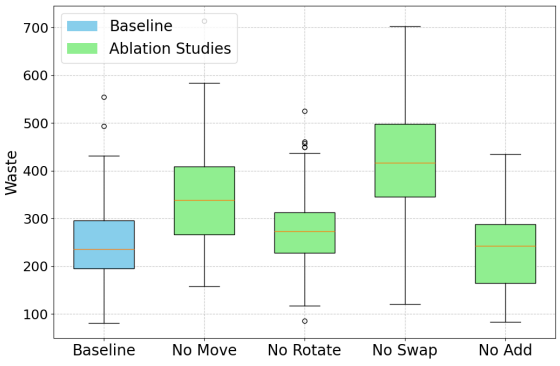


Figure 19: Effect of removing each operation from the *NEIGHBOR* function.

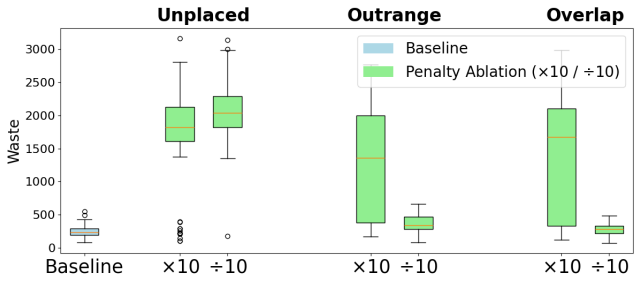


Figure 20: Effect of scaling each penalty in the *EVAL* function (*Uncovered*, *Outrange*, and *Overlap*) by factors of ×10 and ÷10, compared to the baseline configuration.

(from 246.91 to 1559.07). Removing *Swap* also significantly degraded performance (mean = 428.11), likely due to its essential role in effectively replacing poorly placed polygons. *Move* had a noticeable impact (mean = 343.91), while *Rotate* yielded a moderate decrease in performance (mean = 277.96), as both are useful for local refinement. Interestingly, removing *Add* slightly improved the mean waste (240.38), but increased variance (standard deviation = 95.55), suggesting that while the solution becomes less stable, it may occasionally yield better results. In fact, the median waste in this setting (242.50) was slightly worse than the baseline (236.00), indicating that performance is less consistent despite occasional improvements. These results indicate that all five operations contribute to efficient optimization, with *Erase*, *Swap*, and *Move* being particularly critical.

C.2 The effectiveness of EVAL function

As shown in Figure 20, we conducted an ablation study on the *EVAL* function by varying the relative magnitudes of the three penalties (*Uncovered*, *Outrange*, and *Overlap*) defined in Section 4.3.2, using the real scraps dataset. Specifically, each penalty was scaled by factors of ×10 and ÷10 relative to the baseline configuration, and the resulting impact on layout quality was quantified in terms of waste. Table 4 summarizes the main findings. Notably, the baseline achieved a mean waste of 246.91, while adjusting the penalties in either direction led to considerable degradation. Reducing the *Uncovered* penalty by a factor of ÷10 severely impaired layout completeness (2056.09), and increasing it to ×10 also resulted in high waste (1730.00). For the *Outrange* penalty, scaling up produced a mean waste of 1241.52, and scaling down yielded 370.48, both substantially worse than the baseline. Similarly, extreme scaling of the *Overlap* penalty led to significant performance drops: increasing it resulted in a mean waste of 1358.90, and decreasing it yielded 281.27. These results collectively highlight the importance of maintaining carefully balanced penalty ratios, as improper weighting of *Uncovered*, *Outrange*, or *Overlap* disrupts the ability to balance between competing objectives, including complete coverage, boundary containment, and overlap minimization.