# Interactive Layout Transfer

Niraj Ramesh Dayama
Department of Communications and Networking, Aalto University
Helsinki, Finland
niraj.dayama@aalto.fi

Simo Santala
Department of Communications and Networking, Aalto University
Helsinki, Finland
simo.santala@icloud.com

Lukas Brückner
Department of Communications and Networking, Aalto University
Helsinki, Finland
luksuriousb@gmail.com

Kashyap Todi
Department of Communications and Networking, Aalto University
Helsinki, Finland
kashyap.todi@gmail.com

Du Jingzhou
Huawei Technologies
China
du.jingzhou@huawei.com

Antti Oulasvirta
Department of Communications and Networking, Aalto University
Helsinki, Finland
antti.oulasvirta@aalto.fi

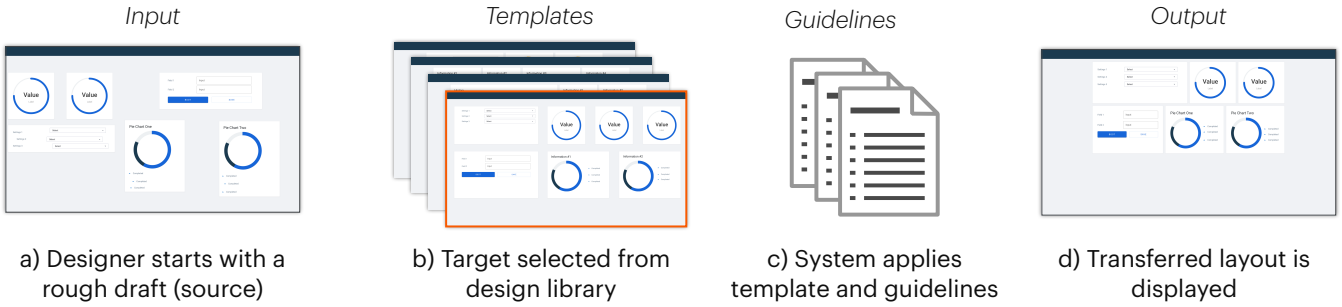



**Figure 1: In interactive layout transfer, a draft (incomplete) layout is automatically transferred to the structure of a target template selected by the designer. Transfer obeys pertinent design guidelines and can warn about possible guideline violations.**

## ABSTRACT

During the design of graphical user interfaces (GUIs), one typical objective is to ensure compliance with pertinent style guides, ongoing design practices, and design systems. However, designing compliant layouts is challenging, time-consuming, and can distract creative thinking in design. This paper presents a method for *interactive layout transfer*, where the layout of a source design – typically an initial rough working draft – is transferred automatically using a selected reference/template layout while complying with relevant guidelines. Our integer programming (IP) method extends previous work in two ways: first, by showing how to transform a rough draft into the final target layout using a reference template and, second, by extending IP-based approaches to adhere to guidelines. We demonstrate how to integrate the method into a real-time interactive GUI sketching tool. Evaluation results are presented from a case study and from an online experiment where the perceived quality of layouts was assessed.


*IUI '21, April 13–17, 2021, College Station, TX, USA*

ACM ISBN 978-1-4503-8017-1/21/04…$15.00
https://doi.org/https://doi.org/10.1145/3397481.3450652

## CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**.

## KEYWORDS

User Interface Design, Computational Methods, MILP, Optimisation, Style Transfer

**ACM Reference Format:**
Niraj Ramesh Dayama, Simo Santala, Lukas Brückner, Kashyap Todi, Du Jingzhou, and Antti Oulasvirta. 2021. Interactive Layout Transfer. In *26th International Conference on Intelligent User Interfaces (IUI '21), April 13–17, 2021, College Station, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/https://doi.org/10.1145/3397481.3450652

## 1 INTRODUCTION

A central challenge in the design of graphical user interfaces (GUIs) is that there are several objectives and constraints expressed in a number of documents and places, such as in design briefs, style guides, design systems, and design examples. Ensuring that the end-result is compliant with all of them is challenging. Designers typically address this iteratively, creating drafts, comparing them against requirements, improving the drafts, and so on. Even when guidelines are explicitly specified and available, designers face difficulties in ensuring compliance [26]. Moreover, professional designers often need to ensure consistency within an *ecosystem* of such requirements, and find a balance between general corporate

identity and the particular project at hand [39]. Ensuring consistency is a labour-intensive, time-consuming process which may distract the designer from the primary goal of creatively exploring the design space.

The starting point for our research is the observation that a lot of the information needed for ensuring compliance already exists in known *design examples*. Designers routinely seek, curate, and utilise design examples, or examples of known good designs, in their work [14, 20]. However, because design examples are often scattered across different sources – across the internet and on personal folders [16] – it may be practically hard to find relevant examples. Further, even after identifying an example, designers still require manual effort to transform the draft at hand while ensuring that the final design adheres to all expected design principles and organisation-specific rules. Our goal is a computational method that can assist designers by exploiting design examples, in particular by proposing automatically generated layouts that ensure consistency with a user-specified example and compliance with requirements like guidelines.

This paper contributes to the growing body of literature on computational design methods that can aid designers in complex tasks such as design exploration and refinement [6, 17, 20, 32, 40]. By mathematically formulating design objectives, and by using learning- or optimisation-based approaches, these methods generate appropriate and diverse solutions. However, with few exceptions (see Related Work), previous literature has considered design in isolation – independent of other co-existing designs – when generating solutions. In addition, previous work on computational retargeting has not addressed consistency and compliance as a factor. This paper addresses the identified gap.

We propose *interactive layout transfer* as a novel approach for generating layouts by exploiting known good designs. In *layout retargeting*, a source layout is transferred to a target layout by mapping the elements and visual properties [18]. Layout transfer extends this in two ways. First, our approach requires no unambiguous mapping between the source and the target. Our method finds that mapping algorithmically. This extends the scope of layouts that can be retargeted. Second, our approach also ensures adherence to existing guidelines when transferring the layout. To sum up, given a reference library (template designs), and a *source* (initial) draft design, our approach first identifies most relevant existing GUIs from the library by computing distances between designs. A *retargeted* (final) design is constructed by transferring the elements from the source design to the layout of the matching design.

In contrast to fully-automated approaches, our approach can interactively support the designer. It enables designers to create well-formed layouts simply by specifying a source and a target. It helps the designer by also identifying and proposing suitable targets. The outputs are aligned to specified guidelines, ensuring consistency. In case guidelines are violated, the system can point out the violations and suggest repairs. Throughout, designers retain control over design solutions and can intervene to override or modify the computationally generated results.

The computational approach we present in this paper extends computational layout design methods based on *integer programming* (IP). Recent literature has identified IP as a capable platform

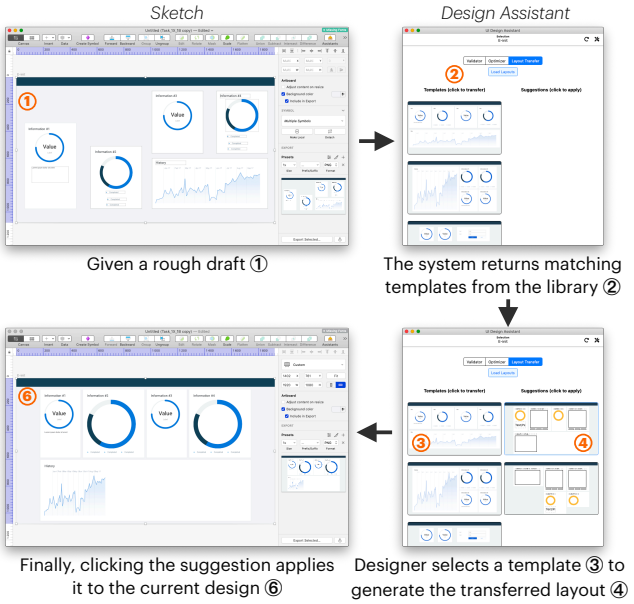



**Figure 2: Workflow for interactive layout transfer, where a draft design is transformed using a target layout.**

for computational GUI design for several reasons [32]: As a mathematical optimisation method, it *guarantees* an optimal solution to the stated problem while adhering to constraints. When time budget is more limited, IP can also generate a close-to-optimal solution. The representation of layouts as decisions can achieve computational performance suitable for interactive usage. IP methods can also be defined to re-direct the search process – interactively – within a feasible region [6]. The challenge in developing IP-based methods is the mathematical formulation of objectives and constraints. Thus, the approach complements supervised learning based approaches that rely on human annotated datasets.

In the rest of the paper, after reviewing relevant literature, we present our interactive style transfer method, and interaction techniques to integrate it into design tools. We assess the efficacy of our approach by conducting a crowd-sourced ratings study of generated layouts, and discuss preliminary results from a case study conducted with a large ICT company.

## 1.1 Walkthrough

Our tool supports two novel use cases:

*1. Search and transfer (Figure 2):* The designer starts by creating or selecting a rough draft as the source layout. The design assistant finds the most similar layouts from a library of existing design examples, and displays them to in an example gallery. Upon selecting a reference layout, the assistant applies the spatial structure of the reference layout to the content of the source layout while following design guidelines. The resulting retargeted layout is displayed to the designer, who can then make further changes if required.

*2. Validate and optimise (Figure 3):* A designer can consult the design assistant to identify any guideline violations in a source layout. The

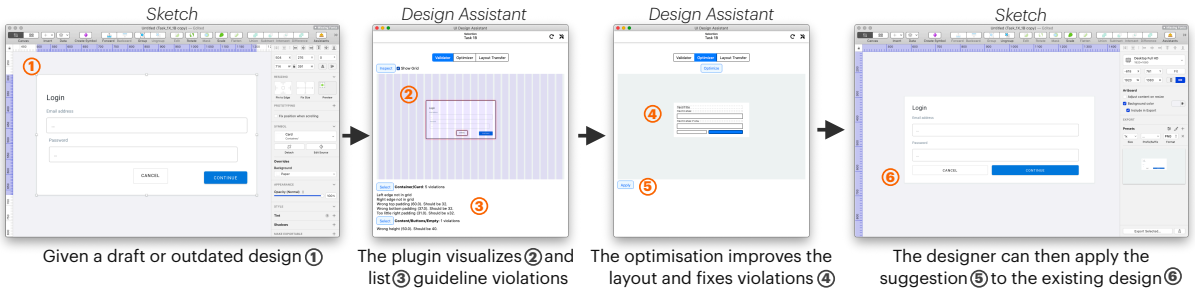



**Figure 3: Workflow for layout validation and optimisation, where violations are identified and fixed.**

assistant visualises these violations, and provides options to automatically repair them. Given a single source layout, our algorithm generates a layout suggestion according to the design system guidelines and aesthetic qualities with minimal changes to the source layout. The resulting validated layout is displayed to the designer.

## 1.2 Problem Definition

The primary underlying computational problem for layout transfer is defined as follows:

> *Given a source layout containing rectangular elements* $\mathbb{E}$*, and given a library* $\mathbb{L}$ *of several different existing designs, compute a new retargeted layout comprising* $\mathbb{E}$ *by identifying and utilising a suitable reference target layout from* $\mathbb{L}$ *while adhering to the guidelines of the design system.*

To address this problem, we identify the following objectives:

(1) Enforce structural consistency with the most relevant design from the library $\mathbb{L}$.
(2) Adhere to the guidelines of the design system.
(3) Ensure layout aesthetic principles such as alignment, symmetry, and packing.

In addition to the primary use-case of layout transfer, the above objectives also enable standalone guideline verification and repair for a source layout. Here, it is sufficient to apply objectives 2 and 3 only.

## 1.3 Scope of this paper

For the purposes of this paper, we will restrict ourselves to a specific, constrained, and simplified characterisation of layouts in terms of resolution, number of widgets, types, composition of layout in header + content, etc. These restrictions can be relaxed (potentially with some costs in terms of computational performance).

The techniques discussed in this paper can be applied for a wide range of graphic design applications. Although the current paper is restricted to desktop/mobile applications only, web-based adaptation is also possible. It is quite standard to convert an existing web page JSON layout format via JavaScript. Thereafer, the model-based approach extends to self-adapting web GUIs such as [19]. Such web-based applications are then ideal platforms where all our objectives 1, 2 and 3 can be applied directly.

# 2 RELATED WORK

Our goal in this work is to provide interactive support to facilitate the design of compliant UI layouts that follow specified style guides and design systems and are consistent with design examples. Our method extends prior work on computational approaches for UI design, and style transfer and layout retargeting.

## 2.1 Promoting Consistency in Design

Consistency in design increases usability and acceptance of GUIs [28]. Template-based design [7, 42] is a commonly used approach to ensure consistency between designs. However, templates lack flexibility and restrict designer freedom. Many toolkits and layout managers [24] offer interactive aids like grid-snapping and auto-alignment [17] to support the creation of layouts that follow grid-based principles. While this promotes uniformity on the elementary-level, it does not consider consistency between designs and adherence to specifications. Apple's Human Interface Guidelines [1] and Google's Material Design [13] are two well-known examples of design systems – documentations that provide detailed guidance on creating consistent and usable GUIs across an ecosystem [34, 35]. Integrated development environments (IDEs) such as Xcode and Android Studio enable the development of GUIs that follow these guides. However, these tools neither ensure that final designs are compliant nor do they encourage consistency between design examples – they instead rely upon the designer. As such, these tools are *passive* in that they provide users with tools and functionalities for carrying out design tasks, but they do not play a role in achieving design results. Inspired by [27], we believe that design tools can play a more *active* role in the design process by employing computational approaches for UI design.

## 2.2 Optimisation for UI Design

Several prior works have investigated computational techniques to assist designers during layout design. *Constraint-based tools* [2, 5, 12, 15, 31, 41] support specifying constraints, or rules, that elements within a GUI needed to follow. This enables designers to specify an initial design, which can be automatically adapted based on aspects such as screen resolution or aspect ratio. While such an approach is useful for responsive designs that rely upon designer-defined constraints, it does not offer methods to ensure compliance with given style guides, or to ensure consistency between designs.

*Optimisation-based approaches* have been applied to automatically generate layouts from task specifications [9, 10]. While this

can be beneficial for developers, requiring task specifications is ill-suited for interactive design; further, while output designs follow device constraints, they do not consider design consistency. More recently, design tools have used optimisation to offer designers interactive assistance during tasks such as poster design [30], menu organisation [3], and wireframe sketching [40]. However, as they often employ random-search based black box optimisation techniques, they do not scale to larger problems and offer no compliance guarantees.

Integer programming offers a promising method for solving design problems with solution guarantees, and has been used for varying problems such as menus, keyboards, and layouts [32]. Close to our work, it has recently been used interactive grid-based layout design [6]. While the presented system generated layouts that guaranteed qualities such as alignment and rectangularity, compliance with explicit rules or design examples was not feasible. As such, previous works that have applied optimisation towards UI design have treated each design task in isolation, and not considered compliance between multiple designs.

## 2.3 Style Transfer and Layout Retargeting

*Style transfer* methods in computer vision achieve consistency by applying image statistics of a reference image to a source image. This has found applications in colour adaptation [33] and in transferring the overall style of portrait images [36]. Convolutional neural networks (CNNs) were presented to transfer a painter's artistic style to natural images [11], which has inspired to further research in neural style transfer to improve quality and expand the scope of applications [22, 23, 25, 38]. By adding a structural component to address UI details and usability, CNNs have been recently used to restyle GUIs with artistic images [8]. A drawback of style transfer using these techniques is that they primarily rely on image-based visual features, and do not typically consider structural aspects. As such, they are not suitable for transferring layouts between GUIs, where component-level reorganisation is required.

In *layout retargeting*, the content of one design is reorganised using the layout of an design. Retargeting has been used previously to, for example, generate new webpage designs from existing examples [18], to increase familiarity of layouts [39], and to generate consistent remote control UIs [29]. Data-driven approaches [21, 43] have been used to create layouts that are consistent with a set of prior designs. However, previous methods require extensive training data, rely on an exact mapping between source and target layouts, can not consider style guides or design rules, and do not support designer interventions during the retargeting process.

With *layout transfer*, our work contributes by addressing these challenges by extending IP-based optimisation, and enables designers to interactively ensure compliance with all design objectives, and consistency with design examples.

# 3 APPROACH TO LAYOUT TRANSFER

This section discusses the logical tenets of layout transfer and detail our approach to achieving the objectives as listed in Section 1.2.

## 3.1 Overview

Our method comprises four key elements: (1) A *delta (Δ) algorithm* that calculates the difference between two layouts, (2) a *library search* that identifies the most relevant layout from $\mathbb{L}$ using the Δ algorithm, (3) the core *layout transfer* model that segments a source layout, identifies a suitable mapping with design examples, and transforms it according to the layout of a matching example, and (4) the *guideline model* that ensures elements follow component guidelines, layout rules, and aesthetic principles. The layout validation case then only employs the guideline model on the source layout to identify and repair violations.

*Concepts.* We describe a layout as a distribution of well-defined elements on a configurable fixed-size canvas. This fixed-size layout expectation is quite reasonable while doing UI layout (whether web, desktop or mobile). Every element is defined in terms of its position on the canvas (top-left coordinates), its size (width and height), and its component type (button, input, chart, etc.). Further, we require that the set of permissible component types is defined in advance and semantically grouped, e.g., by their name (such as 'Content/Text/Label', 'Content/Text/Paragraph', 'Content/Action/Button', 'Container/Card', etc.). This naming structure follows the approach inside the Sketch application to group component definitions in related categories, and each element can only belong to a single component. These component types are broadly classified into one of three main categories, matching the first part of their name: header, container, or content. We expect content components to be always placed inside a container component. Header components are always placed at the top of the canvas, which is computationally trivial. Hence for simplicity, the following discussion focuses on the container and content components.

## 3.2 Computing difference (Δ) between layouts:

Consider a pair of GUI layouts $l_1$ and $l_2$. Initially let us make the simplifying assumptions that both these layouts have the same resolution, and that they comprise of the same *number* and *types* of GUI elements. However, these elements may be placed with different sizes and in different locations in the two layouts. Then the logical distance between these layouts can be represented in several different ways:

(1) Summation of relative changes in size
(2) Summation of relative change in location (Euclidean distance moved)
(3) Difference in the number of elements
(4) Difference in the types of elements

If a user or some logical system were to already map every element from $l_1$ to the corresponding element from $l_2$, then it is trivial to calculate the metrics 1 and 2 mentioned above. However, to generalise, we assume that the layouts are given to us without such ready-made mapping. Here, we need to first deduce the best possible mapping for pairs of elements in layouts. This is a non-trivial exercise that we denote as the Δ algorithm. The algorithm accepts a pair of layouts, deduces the best possible mapping between all pairs and then reports the difference between these layouts for that deduced mapping. We model this in terms of an MILP formulation

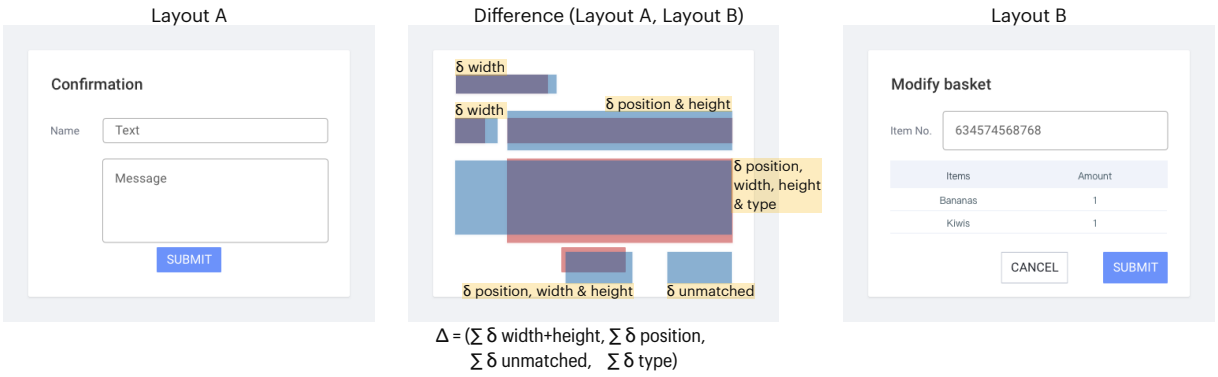



**Figure 4: Delta visualisation. The differences between Layout A and B are highlighted in the middle. The full delta is a vector of four terms as shown at the bottom.**

with the objective of minimising the perceived difference. This perceived difference is reported in terms of a multi-dimensional vector of differences as calculated from different perspectives (Figure 4). The difference can also be collapsed to a single dimensional scalar. The $\Delta$ algorithm by nature is required to be symmetric and should satisfy the triangle inequality. Further, we also require that it can be extended to the more generic case where the number and type of the constituent elements is not the same. Each of the terms in the $\Delta$ function takes the following form:

$$\frac{1}{|\mathbb{E}_s| \cdot \max \delta} \sum_{e_s}^{\mathbb{E}_s} \sum_{e_t}^{\mathbb{E}_t} (\Gamma_{e_s}^{e_t} \cdot \delta(e_s, e_t)) \tag{1}$$

where $\Gamma_{e_s}^{e_t}$ indicates whether two elements are mapped together ($\Gamma_{e_s}^{e_t} = 1$) or not ($\Gamma_{e_s}^{e_t} = 0$), and $\delta(e_s, e_t)$ corresponds to the difference between those elements. The following element-wise difference functions are used:

- $\delta_{\text{size}}(e_s, e_t)$: The absolute difference between the element circumference
- $\delta_{\text{position}}(e_s, e_t)$: The absolute approximate Euclidean distance between element positions
- $\delta_{\text{similarity}}(e_s, e_t)$: The relative number of shared component categories between the components

Element circumference is chosen over area due to its linearity. Euclidean distance likewise not being a linear function, it is approximated using the following formula (from [4])

$$f(x, y) = \frac{1007}{1024} \max(|x|, |y|) + \frac{441}{1024} \min(|x|, |y|) \tag{2}$$

As each element belongs to a known component, and the components are organised according to component categories, the similarity of two components is defined as the relative number of shared categories between them (e.g., components 'Content/Text/Label' and 'Content/Text/Paragraph' share $\frac{2}{3}$, while 'Content/Text/Label' and 'Content/Action/Button' share $\frac{1}{3}$). The values for max $\delta$ are as follows:

- max $\delta_{\text{size}}$: The layout circumference
- max $\delta_{\text{position}}$: The length of the layout diagonal
- max $\delta_{\text{similarity}}$: 1 (normalisation is achieved at the level of each component pair)

We formulate three additional terms for use in different functions of the system:

(1) $\Delta_{\text{unmapped}}(L_a, L_b)$ (in the library search): The number of unmapped elements in the source layout.
(2) $\Delta_{\text{mismatched}}(L_a, L_b)$ (in the layout transfer): The number of totally dissimilar elements mapped together function.
(3) $\Delta_{\text{area}}(L_a, L_b)$ (in the layout transfer): The difference in the area of each target element and the sum of the source element areas mapped to it.

As the layout transfer function allows multiple source elements to be mapped to a single target element, their total area should match the area of the target element (Figure 6a).

Computation of difference between layouts must also distinguish between different element types. This was a critical feature: for example, we require to know that one specific rectangle in the draft actually refers to an image and should not be mapped to a button. So, we map containers only to other containers; individual UI elements and widgets are preferably mapped to the same element type.

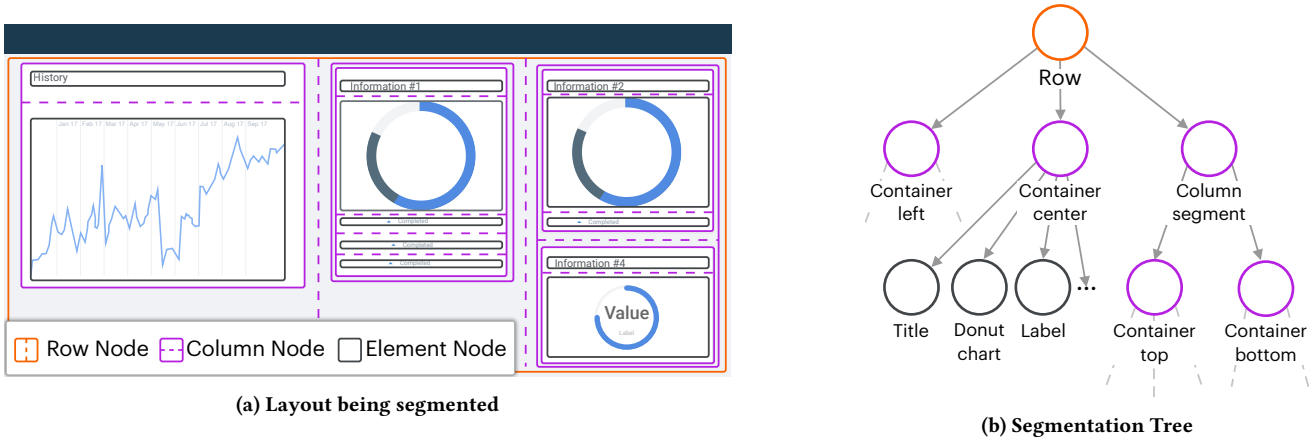
## 3.3 Library Search

Consider a library of $n$ layouts has been provided as $\{L_1, L_2, \ldots, L_n\}$. In our implementation, we assume that these layouts are available as standard Sketch format files. These Sketch files inherently devolve to a tree structure. We translate Sketch files into a common JSON format. Alternatively, it is straightforward to create similar JSON-formatted layout files using other tools such as Figma, Adobe Illustrator, or manually.

Now, when a new source layout $\mathbb{S}$ is given, we wish to find a target layout in the library that best matches $\mathbb{S}$. To address this problem, we first individually compare every layout from the library with $\mathbb{S}$ using the $\Delta$ algorithm. For example, we denote the difference between $L_1$ and $\mathbb{S}$ as $\Delta(L_1, \mathbb{S})$. Thereafter, we pick non-dominated layouts based on the multi-dimensional difference. Finally, we pick the best-matching layouts based on the single-dimensional (scalar) difference.

## 3.4 Layout Transfer

The actual layout transfer occurs when the designer selects a suitable target layout $L_T$ from the identified matches. Now, we wish to




**(a) Layout being segmented**

**(b) Segmentation Tree**

**Figure 5: (a) A layout is segmented by finding all separation lines that divide elements either horizontally or vertically. This produces an hierarchy of bounding boxes (b) A tree structure represents this hierarchy, and consists of virtual segment nodes, container element nodes, and content element nodes. Each internal node is tagged as either a row or a column.**

compute a new retargeted layout $L^*$ that has the same components as $\mathbb{S}$ but arranges them in the structure of $L_T$. To build this $L*$, we argue as follows: The ideal $L^*$ is the one for which the semantic changes to the source layout and the visual difference $\Delta(L^*, L_T)$ to the target layout are minimised. It should be noted that semantic similarity between layouts differs from geometric similarity in that it also considers the component types and their organisation; semantic changes should be minimised so as to retain the intended logical structure of the layout. Again, we model this as an optimisation problem and formulate it in terms of a MILP. The objective of the MILP is to minimise $\Delta(L^*, L_T)$; the constraint is that the $L^*$ must comprise of $\mathbb{S}$ alone.

The layout transfer algorithm first categorises elements in the source layout, and segments the layout (Figure 5a) into a tree structure, where each internal node is classified either as a row or a column. Container elements are considered as internal nodes and content elements as leaf nodes (Figure 5b).

The algorithm then combines the source and target layouts together by placing the content of one layout (source) to the structure of the other (target). This is achieved by finding a mapping between the elements of the two layouts. Each element in the source layout is mapped to a single element in the target layout. Multiple elements are allowed to be mapped to one element in the target. In the case of container elements, these are then merged to form a new container. For content elements, they are placed next to each other in the retargeted output layout (Figure 6a). The mapping of containers and content is linked such that the content of each source container must be mapped to the content of the corresponding target container.

While each element in the source layout must be mapped to one element in the target layout, elements in the target layout are allowed to mapped to none or any number of elements in the source layout. If an element in the target layout is not mapped to any of the source layout elements, the size of the corresponding node is set to zero. If more than one source element is mapped to the same target element, the corresponding node is set to encompass all of the mapped elements, and the elements are laid out along the direction of the parent group node (Fig. 6a).

To maintain semantic features of the source layout, the objective function is augmented with two additional terms:

- The amount of downscaling of each element, including width, height, and circumference
- The change in relative length of circumference of any two elements in the source layout $\mathbb{S}$

These objectives ensure that the design is not changed against the designer's intent, as the size of an element indicates its relative saliency and thus importance in the layout.

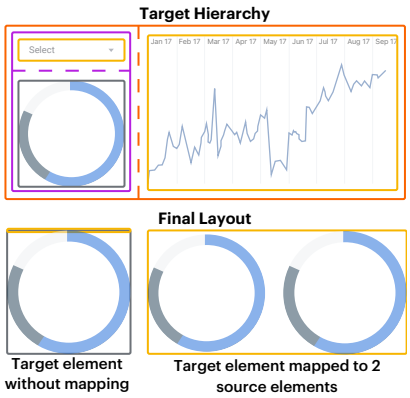
## 3.5 Applying Design Guidelines

The layout transfer model is further augmented with constraints derived from the design guidelines and objectives to ensure aesthetically pleasing results. These constraints include component-specific design system rules applied to elements individually, as well as inter-element constraints, such as margins and padding. For each node in the layout tree, its children are constrained to be placed within its bounds, with padding applied if the node is a container. If the node is designated as a row or a column, the children are stacked next to each other in the respective direction.

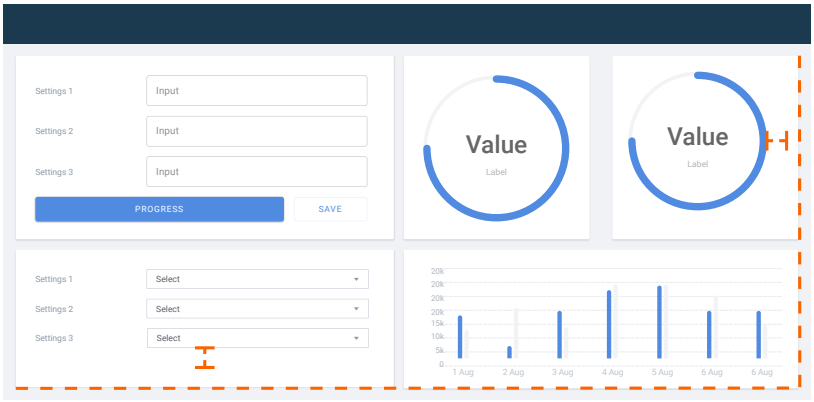
Component guidelines are applied as constraints to each element according to its type. These include minimum, maximum, or fixed dimensions, or aspect ratio. For example, input elements often have a fixed height, but are allowed to stretch horizontally; graphical elements such as pie charts or icons instead have a fixed aspect ratio.

Layout guidelines are represented through the following objectives:

- Layout symmetry: consecutive child nodes should have the same width.
- Container packing: containers should be stretched to use available width and height in the layout.




(a) Element Mappings




(b) Contradictory Constraints

**Figure 6: (a) When target elements are not mapped to any source elements, their size is set to zero. When multiple source elements are mapped to a single target element, it is preferable that their combined area matches that of the target element. (b) In some cases, aesthetic constraints may be contradictory. Here, good container alignment necessitates extra padding in some containers.**

- Container filling: content should extend to the right and bottom edges of the container.

In some cases, the last two objectives may be contradictory (Figure 6b). Here, container packing is preferred over filling to improve alignment.

For guideline validation use-cases, where design guidelines are applied to the source layout without layout transfer, an additional term is added to the objective function to minimise changes to element positions. This ensures that the output retains the overall layout of the source while fixing identified violations.

## 3.6 Implementation

Our models are implemented in Python 3.8. We use the Gurobi solver, running on a server using a stack of Docker containers and orchestrated using Docker Compose. Our design assistant is integrated within the Sketch[1] application, a widely-used design platform, as a plugin. The plugin is implemented in JavaScript, and uses React to load and modify layouts via the Sketch API. While our system supports distributed workflows, we performed all experiments running the server locally on a MacBook Pro (16-inch, 2019) with a 2.3 GHz 8-core Intel i9 CPU.

We conducted a performance analysis covering layouts containing between 20 and 30 elements (see Figure 7). Solving the layout validation model took less than 100 milliseconds in all cases. Solving the $\Delta$ model between two layouts took 10–100 milliseconds per pair, which is well-suited for searching small design libraries (e.g., with 50 layouts in the design library, the result should be returned within 3 seconds). For large design libraries, distances between existing layouts can be precomputed, and the triangle inequality of the $\Delta$ function can be exploited to find the closest layouts more efficiently. Transferring the layout from a perfectly-matched target took a median time of 4.0s [3.1, 4.7], and for well-matched targets 6.3s [4.1, 15.2], where the values in brackets represent the 0.25 and 0.75 quantiles. When the source and target layouts were significantly different, the layout transfer model took a median time of 14.3s [4.4, 38.5]. These results suggest sufficient capabilities for interactive real-time use of our tool.

[1]https://www.sketch.com

# 4 CASE STUDY: DESIGN SYSTEM OF AN ICT COMPANY

We evaluated the compatibility of the approach with professional practice in a case study of a design team in a large-scale consumer electronics manufacturer that develops hardware and software products. The case was motivated by the company's search for approaches and solutions to improve the design workflow within their design teams. The 6 participants in our study were expert professional designers with 4+ years of experience in UI design with Sketch application. 4 designers worked with the installed system directly (with support from authors) while 2 other operated it remotely. The UI design team defined a design system, with a complete set of style and design guidelines (listed below). For reference, a well-curated design library containing existing designs was provided. All future designs for new applications are required to comply with this design system; further, future designs must also follow examples and ongoing practices from the design library.

## 4.1 Design Guidelines

The provided guidelines are composed of two parts: 1) individual components and 2) overall layout system.

*1) Component guidelines:* These specify permissible sizes in terms of minimum/maximum width and height (or a fixed aspect ratio), and the classification type of each component (header, container, content). Table 1 list examples of these guidelines.

*2) Layout guidelines:* These specify the interactions between multiple components. For example, components should be aligned according to a grid, except the header containers, which should be placed at the top of the screen and span the whole layout width.

| Component | Type | Min. Width | Max. Width | Min. Height | Max. Height | Aspect Ratio |
|---|---|---|---|---|---|---|
| Header | Header | 100% | - | 64px | 64px | - |
| Card | Container | - | - | - | - | - |
| Title | Content | - | - | 28px | 28px | - |
| Paragraph | Content | - | - | 20px | - | - |
| Input Label | Content | - | 60px | 20px | 20px | - |
| Text Field | Content | - | - | 50px | 50px | - |
| Button | Content | 100px | - | 40px | 40px | - |
| Bar Chart | Content | - | - | 120px | - | ≥4:3 |
| Pie Chart | Content | - | - | 120px | - | 1:1 |

**Table 1: Example component guidelines defining the dimensions of known components. Content components must always be placed within the bounds of a container element.**

Other containers and content should be placed into two nested grids as defined below. These grid dimensions are defined using a base unit that, along with the number of columns, depends on the screen resolution. At a large screen width of > 1680px, 24 columns are used and the base unit is defined to be 16px. This then scales down as the screen width decreases. For example, for screen width between 385 and 768px, only 12 columns are employed and the base unit is 12px. To perform the actual placements, two grids are defined: for container and content elements. The grids are defined by their gutter (margin between columns) and the margin on both sides towards the canvas. The configuration employed is shown below.

| Gutter | Left Margin | Right Margin |
|---|---|---|
| 1 base unit | ≤ gutter | left margin (+ 1px, if necessary) |

## 4.2 Implementation

Our interactive layout transfer approach is especially suited for deploying such a case. The specified design guidelines were implemented within our system without requiring any modification to our basic implementation (discussed in Section 3). More specifically, they were defined as constraints in the MILP optimisation program. These can be further configured using a human-readable configuration file, without any code. The provided design library was included within the system, and is directly usable by our tool to identify matching targets for layout transfer. Using our custom implementation, designers are now provided with tools to validate and improve their designs, or to transfer their draft designs to the target layouts found in the design library.

## 4.3 Sample Results

Figure 7 illustrates the results as obtained using standard components and design practices of the concerned organisation. In the figure, the X-axis represents the target layout and the Y-axis represents the source layout. The results, presented in a matrix, demonstrate the capability of our layout transfer method for generating different retargeted results using the same source layout against several different targets. Conversely, it also demonstrates retargeting of multiple sources using a single target layout.

In our preliminary observations, designers noted that the final resultant layout matched their expectations quite closely. Further, the interactive performance (results generated within seconds) and the avoidance of tedious manual efforts were acknowledged as major strengths of the system. When target layouts were reasonably compatible with the source, the system met its objectives of creating compliant layouts with minimal effort.

# 5 EVALUATION: PERCEIVED QUALITY OF WIREFRAMES

To evaluate whether our optimisation-based approach can yield good design results, as measured by the perceived quality, we conducted a ratings-based study. Here, we compared three groups (conditions) of layout designs – initial draft designs (Draft), designer-made designs (Designer), and optimiser-generated designs (Optimiser) – by asking participants to rate a series of layout designs.

## 5.1 Materials

A set of 7 varying design tasks was use for each condition. For the Draft condition, initial draft layouts were designed for each of the 7 design tasks (e.g. Figure 8. For the Designer condition, experienced designers were asked to manually improve the draft designs to create 7 designer-made layouts. For the Optimiser condition, our optimiser took draft layouts as input (source), and generated a optimised solutions for each of them. This resulted in a total of 21 layout designs for the study. To enable participants to these designs, we implemented an online survey. Each design was included as an individual question, and a 10-point rating scale was used for user response. The presentation order of these 21 questions was randomised between participants.

## 5.2 Participants

Our participants were recruited via Prolific[2], an online platform with a large number of high-quality participants. Participation was restricted to those with a worker approval rate above 95%, to ensure high-quality responses. Previous research has observed that expert and non-expert participants alike can differentiate good designs from bad ones [37]. Taking this into account, participation was not restricted to designers.

A total of 50 participants (24 feminine, 25 masculine, 1 prefer not to say) aged between 18 and 53 years (Mean = 26.4, SD = 8.3) completed the study. The study took under 10 minutes to complete, and participants were paid £1.25 upon completion, corresponding to an hourly wage of £7.50/hr. Participation was under informed consent, and the study adhered to European privacy laws (GDPR).

[2]https://prolific.co

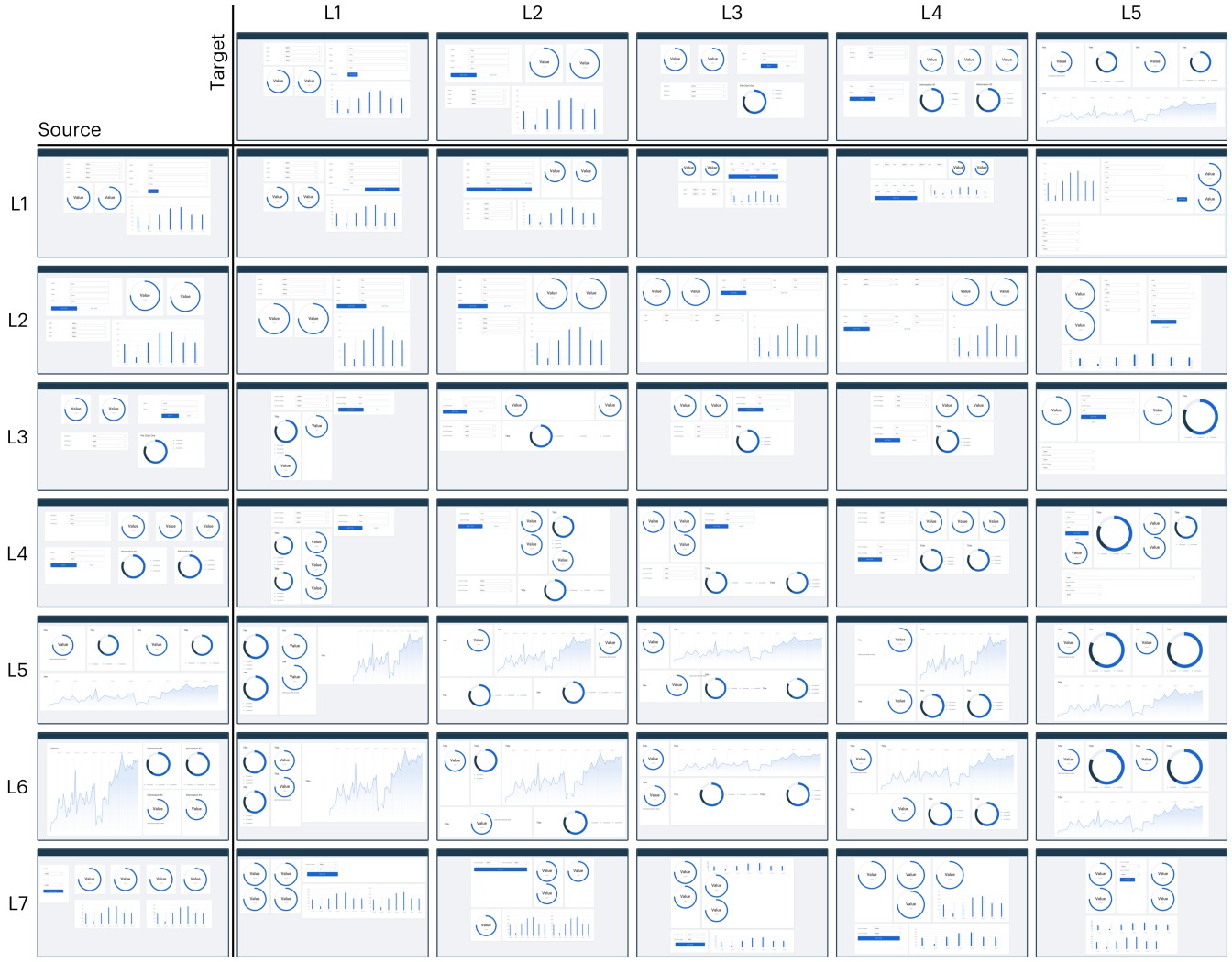
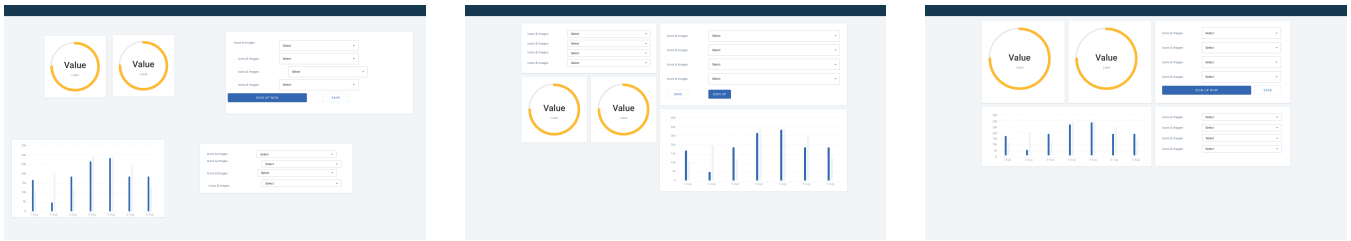
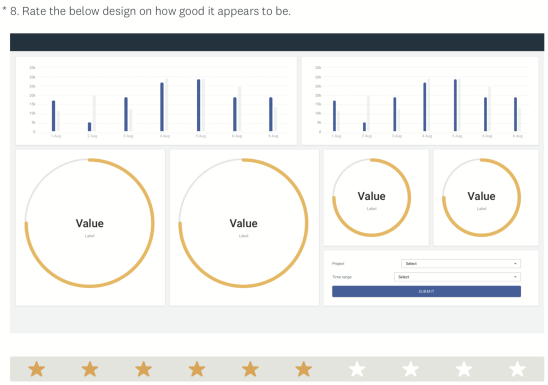



**Figure 7: Examples of combining the content of source layouts (y-axis) with the design of target layouts (x-axis). Note, that in the diagonal source and target are the same layout, and changes are due to the application of design guidelines.**

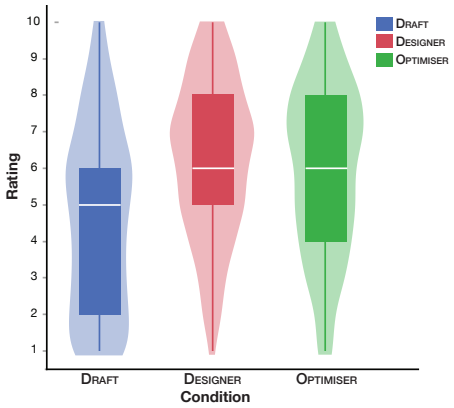



DRAFT DESIGNER OPTIMISER

**Figure 8: For each design task, layouts for three conditions – DRAFT, DESIGNER, and OPTIMISER – were independently rated by participants on a scale of 1 to 10.**




(a) A screenshot from one question.




(b) Results from the survey

**Figure 9: Our ratings-based study evaluated perceived quality of designs. Ratings for Optimiser condition were higher than Draft and comparable to Designer.**

## 5.3 Procedure and Design

Participants on the Prolific platform received a link to the survey. The survey began with an introductory briefing and participant consent, followed by layout-rating tasks. Participants were not informed about the condition for each question. As stimulus, they were shown the image of a layout (see Figure 8 for examples). Participants were asked to pay attention to the overall quality of the designs, and to important aspects such as the alignment of elements, while providing ratings on a scale of 1 (worst) to 10 (best) as illustrated in Figure 9a. In a *within-subject design*, each participant rated all 21 designs in randomised order. At the end of the survey, participants provided demographic information (age and gender identification) and were redirected back to Prolific to claim their compensation. To summarise, the design is: 50 participant × 3 conditions × 7 layout designs = 1050 trials.

## 5.4 Results

The mean user ratings for the Draft, Designer, and Optimiser conditions were 4.6 (SD = 2.5), 6.2 (SD = 2.1), and 6.0 (SD = 2.3) respectively. The results are illustrated in Figure 9b. The effect of the primary dependent variable *condition* on user rating, tested with repeated-measures ANOVA, was found to be statistically significant, $F(2,98) = 41.01$, $p < 0.0001$. Post-hoc test using Tukey HSD revealed that both Optimiser and Designer had significantly higher ratings as compared to Draft (Optimiser - Draft = 1.35; Designer - Draft = 1.64). However, the difference between Designer and Optimiser (= 0.28) was not statistically significant. These results provide evidence for our optimisation-based approach. Optimised layouts have higher ratings compared to draft designs, and similar to those designed by professionals.

# 6 SUMMARY AND DISCUSSION

We presented a novel integer programming (IP) approach for interactive layout transfer that retargets and optimises a rough draft using a target layout. The system helps the designer to select a target layout to which a given source is transferred. A mapping between layouts can be algorithmically deduced even if there is no obvious one-to-one mapping between the elements. Another key feature of our system is to automatically ensure adherence to design guidelines. This feature reduces designer effort while improving consistency. We also presented results from a case study and a ratings study suggesting that (1) the method can be integrated in a real interactive design tool and thereby incorporated in professional practice and (2) that the outputs are of high quality. Besides transfer, the tool can detect violations to design guidelines and suggest optimisation-based repairs.

The presented techniques – the delta algorithm, library search, and transfer methods for IP – constitute a computationally assisted mechanism to leverage and utilise UI design assets. Our IP formulation successfully finds optimal (or close-to-optimal) solutions within a short time (often few seconds) thus enabling interactive and unobtrusive use. Thus, the paper contributes to the evolving understanding of algorithmic approaches for graphical layouts.

We identify several possible improvements that support the applicability of this system. For example, our underlying model does not currently support extending the tree structure of the segmented target layout. Future work should look into ways of allowing the model to extend beyond the components available in the target hierarchy. Second, while the optimiser can handle canvases of different widths, we have not tested how targets that are of different aspect ratio work out. Lastly, the current paradigm depends on the availability of relevant target layouts in the design library. We can also support the generation of good layouts (see [6]) even when the library does not already include suitable templates. Finally, we foresee that the approach presented here may encourage design teams to collect and curate design examples. The shared knowledge base of design teams includes the design artefacts they have produced, such as layouts and outcomes. This knowledge base can be considered an asset. Computational tools that allow directly exploiting them in design tools and improve the designer's productivity may encourage mature practices of curation and consolidation of UI design assets.

# 7 OPEN SCIENCE

We support replication and further research by releasing mathematical formulations and an open code base on our project page: https://userinterfaces.aalto.fi/layout-transfer.

## ACKNOWLEDGMENTS

This work was funded by Huawei Technologies and the Academy of Finland (grants 328813 "Human Automata" and 318559 "BAD").

## REFERENCES

[1] Apple. 2020. Human Interface Guidelines. https://developer.apple.com/design/human-interface-guidelines/ [Retrieved 9 October, 2020].

[2] Greg J. Badros, Alan Borning, and Peter J. Stuckey. 2001. The Cassowary Linear Arithmetic Constraint Solving Algorithm. *ACM Trans. Comput.-Hum. Interact.* 8, 4 (Dec. 2001), 267–306. https://doi.org/10.1145/504704.504705

[3] Gilles Bailly, Antti Oulasvirta, Timo Kötzing, and Sabrina Hoppe. 2013. MenuOptimizer: Interactive Optimization of Menu Systems. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. ACM, New York, NY, USA, 331–342. https://doi.org/10.1145/2501988.2502024

[4] Rafael Baptista. 2003. Fast Approximate Distance Functions. https://www.flipcode.com/archives/Fast_Approximate_Distance_Functions.shtml. Accessed: 9 October 2020.

[5] Alan Borning, Richard Lin, and Kim Marriott. 1997. Constraints for the Web. In *Proceedings of the Fifth ACM International Conference on Multimedia* (Seattle, Washington, USA) *(MULTIMEDIA '97)*. Association for Computing Machinery, New York, NY, USA, 173–182. https://doi.org/10.1145/266180.266361

[6] Niraj Ramesh Dayama, Kashyap Todi, Taru Saarelainen, and Antti Oulasvirta. 2020. GRIDS: Interactive Layout Design with Integer Programming. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376553

[7] William Drenttel and Jessica Helfand. 1999. Method and system for computer screen layout based on a recombinant geometric modular structure. https://patents.google.com/patent/US7124360B1/en

[8] Michael H Fischer, Richard R Yang, and Monica S Lam. 2020. ImagineNet: Restyling Apps Using Neural Style Transfer. *arXiv preprint arXiv:2001.04932* (2020).

[9] Krzysztof Gajos and Daniel S. Weld. 2004. SUPPLE: Automatically Generating User Interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI '04)*. ACM, New York, NY, USA, 93–100. https://doi.org/10.1145/964442.964461

[10] Krzysztof Gajos and Daniel S. Weld. 2005. Preference Elicitation for Interface Optimization. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST '05)*. ACM, New York, NY, USA, 173–182. https://doi.org/10.1145/1095034.1095063

[11] Leon A. Gatys, Alexander S. Ecker, and M. Bethge. 2015. A Neural Algorithm of Artistic Style. *ArXiv* abs/1508.06576 (2015).

[12] Michael Gleicher. 1992. Briar: A Constraint-based Drawing Program. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '92)*. ACM, New York, NY, USA, 661–662. https://doi.org/10.1145/142750.143074

[13] Google. 2020. Material Design. https://material.io/design [Retrieved 9 October, 2020].

[14] Scarlett R. Herring, Chia-Chen Chang, Jesse Krantzler, and Brian P. Bailey. 2009. Getting Inspired! Understanding How and Why Examples Are Used in Creative Design Practice. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*. Association for Computing Machinery, New York, NY, USA, 87–96. https://doi.org/10.1145/1518701.1518717

[15] Yue Jiang, Wolfgang Stuerzlinger, Matthias Zwicker, and Christof Lutteroth. 2020. ORCSolver: An Efficient Solver for Adaptive GUI Layout with OR-Constraints. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3313831.3376610

[16] Janin Koch, Magda Laszlo, Andres Lucero Vera, Antti Oulasvirta, et al. 2018. Surfing for Inspiration: digital inspirational material in design practice. In *Design Research Society International Conference: Catalyst*. Design Research Society.

[17] Yuki Koyama, Daisuke Sakamoto, and Takeo Igarashi. 2014. Crowd-Powered Parameter Analysis for Visual Design Exploration. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (Honolulu, Hawaii, USA) *(UIST '14)*. Association for Computing Machinery, New York, NY, USA, 65–74. https://doi.org/10.1145/2642918.2647386

[18] Ranjitha Kumar, Jerry O Talton, Salman Ahmad, and Scott R Klemmer. 2011. Bricolage: example-based retargeting for web design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2197–2206.

[19] Markku Laine, Ai Nakajima, Niraj Dayama, and Antti Oulasvirta. 2020. Layout as a Service (LaaS): A Service Platform for Self-Optimizing Web Layouts. In *Web Engineering*, Maria Bielikova, Tommi Mikkonen, and Cesare Pautasso (Eds.). Springer International Publishing, Cham, 19–26. https://doi.org/10.1007/978-3-030-50578-3_2

[20] Brian Lee, Savil Srivastava, Ranjitha Kumar, Ronen Brafman, and Scott R Klemmer. 2010. Designing with interactive example galleries. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 2257–2266.

[21] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. 2019. LayoutGAN: Generating Graphic Layouts with Wireframe Discriminators. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. https://openreview.net/forum?id=HJxB5sRcFQ

[22] Xueting Li, Sifei Liu, J. Kautz, and M. Yang. 2018. Learning Linear Transformations for Fast Arbitrary Style Transfer. *ArXiv* abs/1808.04537 (2018).

[23] Minxuan Lin, Fan Tang, Weiming Dong, Xiao Li, Chongyang Ma, and C. Xu. 2020. Distribution Aligned Multimodal and Multi-Domain Image Stylization. *ArXiv* abs/2006.01431 (2020).

[24] Simon Lok and Steven Feiner. 2001. *A Survey of Automated Layout Techniques for Information Presentations*.

[25] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. 2017. Deep Photo Style Transfer. *arXiv preprint arXiv:1703.07511* (2017).

[26] Jonas Löwgren and Ulrika Laurén. 1993. Supporting the use of guidelines and style guides in professional user interface design. *Interacting with Computers* 5, 4 (1993), 385 – 396. https://doi.org/10.1016/0953-5438(93)90003-C

[27] Brad Myers, Scott E. Hudson, and Randy Pausch. 2000. Past, Present, and Future of User Interface Software Tools. *ACM Trans. Comput.-Hum. Interact.* 7, 1 (March 2000), 3–28. https://doi.org/10.1145/344949.344959

[28] Jeffrey Nichols, Duen Horng Chau, and Brad A Myers. 2007. Demonstrating the viability of automatically generated user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1283–1292.

[29] Jeffrey Nichols, Brad A. Myers, and Brandon Rothrock. 2006. UNIFORM: Automatically Generating Consistent Remote Control User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montréal, Québec, Canada) *(CHI '06)*. Association for Computing Machinery, New York, NY, USA, 611–620. https://doi.org/10.1145/1124772.1124865

[30] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2015. DesignScape: Design with Interactive Layout Suggestions. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 1221–1224. https://doi.org/10.1145/2702123.2702149

[31] Stephen Oney, Brad Myers, and Joel Brandt. 2012. ConstraintJS: Programming Interactive Behaviors for the Web by Integrating Constraints and States. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 229–238. https://doi.org/10.1145/2380116.2380146

[32] Antti Oulasvirta, Niraj Ramesh Dayama, Morteza Shiripour, Maximilian John, and Andreas Karrenbauer. 2020. Combinatorial optimization of graphical user interface designs. *Proc. IEEE* 108, 3 (2020), 434–464.

[33] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley. 2001. Color Transfer between Images. *IEEE Computer Graphics and Applications* 21 (2001), 34–41.

[34] Mindy Reyes (Ed.). 2019. *Hack the design system*. Idean Publishing. http://idean.com/learn

[35] Karri Saarinen. [n.d.]. Building a Visual Language–Behind the scenes of our new design system. https://airbnb.design/building-a-visual-language/. Accessed: 23 April 2020.

[36] Yi-Chang Shih, Sylvain Paris, Connelly Barnes, W. Freeman, and F. Durand. 2014. Style transfer for headshot portraits. *ACM Trans. Graph.* 33 (2014), 148:1–148:14.

[37] Kesler Tanner and James Landay. 2019. *"I Know It When I See It": How Experts and Novices Recognize Good Design*. Springer International Publishing, Cham, 249–266. https://doi.org/10.1007/978-3-319-97082-0_13

[38] Ondřej Texler, David Futschik, Michal Kučera, Ondřej Jamriška, Šárka Sochorová, Menglei Chai, Sergey Tulyakov, and Daniel Sýkora. 2020. Interactive Video Stylization Using Few-Shot Patch-Based Training. *ACM Transactions on Graphics* 39, 4 (2020), 73.

[39] Kashyap Todi, Jussi Jokinen, Kris Luyten, and Antti Oulasvirta. 2018. Familiarisation: Restructuring Layouts with Visual Learning Models. In *23rd International Conference on Intelligent User Interfaces* (Tokyo, Japan) *(IUI '18)*. Association for Computing Machinery, New York, NY, USA, 547–558. https://doi.org/10.1145/3172944.3172949

[40] Kashyap Todi, Daryl Weir, and Antti Oulasvirta. 2016. Sketchplore: Sketch and Explore with a Layout Optimiser. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems (DIS '16)*. ACM, New York, NY, USA, 543–555. https://doi.org/10.1145/2901790.2901817

[41] Clemens Zeidler, Christof Lutteroth, and Gerald Weber. 2012. Constraint solving for beautiful user interfaces: how solving strategies support layout aesthetics. In *Proceedings of the 13th International Conference of the NZ Chapter of the ACM's Special Interest Group on Human-Computer Interaction*. 72–79.

[42] Shuyu Zheng, Ziniu Hu, and Yun Ma. 2019. FaceOff: Assisting the Manifestation Design of Web Graphical User Interface. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining* (Melbourne VIC, Australia) *(WSDM '19)*. Association for Computing Machinery, New York, NY, USA, 774–777. https://doi.org/10.1145/3289600.3290610

[43] Xinru Zheng, Xiaotian Qiao, Ying Cao, and Rynson WH Lau. 2019. Content-aware generative modeling of graphic design layouts. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 133.