Responsive and Personalized Web Layouts with Integer Programming

MARKKU LAINE, Aalto University, Finland
YU ZHANG, Aalto University, Finland
SIMO SANTALA, Aalto University, Finland
JUSSI P. P. JOKINEN, University of Helsinki, Finland
ANTTI OULASVIRTA, Aalto University, Finland

Over the past decade, responsive web design (RWD) has become the *de facto* standard for adapting web pages to a wide range of devices used for browsing. While RWD has improved the usability of web pages, it is not without drawbacks and limitations: designers and developers must manually design the web layouts for multiple screen sizes and implement associated adaptation rules, and its "one responsive design fits all" approach lacks support for personalization. This paper presents a novel approach for automated generation of *responsive and personalized web layouts*. Given an existing web page design and preferences related to design objectives, our integer programming -based optimizer generates a consistent set of web designs. Where relevant data is available, these can be further automatically personalized for the user and browsing device. The paper includes presentation of techniques for runtime adaptation of the designs generated into a fully responsive grid layout for web browsing. Results from our ratings-based online studies with end users (N = 86) and designers (N = 64) show that the proposed approach can automatically create high-quality responsive web layouts for a variety of real-world websites.

CCS Concepts: • **Human-centered computing** → **Web-based interaction**; *Human computer interaction (HCI)*; *User interface design*; • **Information systems** → **Web interfaces**.

Additional Key Words and Phrases: responsive web design; web personalization; computational design; integer programming; retargeting

ACM Reference Format:
Markku Laine, Yu Zhang, Simo Santala, Jussi P. P. Jokinen, and Antti Oulasvirta. 2021. Responsive and Personalized Web Layouts with Integer Programming. *Proc. ACM Hum.-Comput. Interact.* 5, EICS, Article 213 (June 2021), 23 pages. https://doi.org/10.1145/3461735

1 INTRODUCTION

Delivering a good user experience is vital for the success of any modern website, and having a responsive website is one of the most effective means of doing so. Defined simply, *responsive web design* (RWD) is an approach to web design and development that allows the website's layout and content to adapt to various devices used for browsing. Amid increasing global mobile-data traffic

Authors' addresses: Markku Laine, Aalto University, Helsinki, Finland, markku.laine@aalto.fi; Yu Zhang, Aalto University, Helsinki, Finland, yu.zhang@alumni.aalto.fi; Simo Santala, Aalto University, Helsinki, Finland, simo.santala@alumni.aalto.fi; Jussi P. P. Jokinen, University of Helsinki, Helsinki, Finland, jussi.p.jokinen@helsinki.fi; Antti Oulasvirta, Aalto University, Helsinki, Finland, antti.oulasvirta@aalto.fi.

2573-0142/2021/6-ART213
https://doi.org/10.1145/3461735

[10] and the constant emergence of new display devices and screen resolutions [1, 34], the role of RWD is growing more and more important.

Although RWD has become the prevailing approach to web design, the amount of effort involved restricts its application [26]. A web layout must be designed separately for every screen size, with each demanding the implementation of corresponding adaptation rules [21]. In consequence, personalization gets compromised: often, the large amount of work involved renders it prohibitively costly to design pages for individual users or market segments.

Computational design methods have recently gained popularity in assisting with websites' adaptation to various constraints [27]. For instance, integer programming can be used for quickly generating various user interface designs [28]. This is a mathematical optimization method that relies on an explicit model of the design problem. By extending applications of integer programming for grid layouts [28] to RWD, practitioners can automate and personalize this process. Hence, less manual effort and time are required for RWD, and the end result can be more personalized.

In this paper, we present a method for *computational responsive web design* (C-RWD) with two goals in mind: (1) decreasing the amount of manual effort involved in design and development and (2) improving usability for individual users via personalization. Given an existing web page design, which may be for any screen size, our method generates a fully responsive web page with breakpoint designs and smooth transitions between them. For example, it can generate pages for much larger displays (e.g., of laptop and desktop devices) than those behind its initial mobile layout -based seed. Moreover, these designs can be personalized where corresponding data on the individual user is available. For instance, in the case of a user known to frequently access three categories of content, these could be made more visually prominent and easier to access. Central to our approach is consistency of the designs generated: the designs in the set can be made as visually consistent with each other as desired, maintaining a certain overall look and feel.

The paper presents the following main contributions:

- A novel approach for automated generation of responsive and personalized web layouts with integer programming
- An integer programming -based model used for generating a consistent set of personalized web designs for given breakpoints
- Web engineering techniques for collecting optimization constraints and adapting the resulting designs at runtime to form a fully responsive grid layout
- Results of two ratings-based online studies, with end users ($N = 86$) and designers ($N = 64$)

The remainder of this paper is organized as follows. We begin with a brief introduction to RWD and a review of related work, in Section 2. We then describe our approach (C-RWD) in detail (in Section 3) and present the results from the two online studies (in Section 4). With Section 5, we continue by analyzing our findings and outlining ideas for future work, before offering conclusions in Section 6.

2 RELATED WORK

With the computational design method proposed in this paper, designers can for the first time generate both responsive and personalized web layouts. As the following review attests, prior work has presented the main elements of the approach; however, they have not been brought together in a working system before. The section concludes with a comparison of our approach with previous methods.

2.1 Responsive Web Design

Responsive web design is an approach in which a website's layout and content are adapted to the size and capabilities of the user's browsing device. The term, coined by Ethan Marcotte in 2010, encompasses a set of techniques, including fluid grids, flexible media, and media queries [20]. To address the complexity involved, several CSS frameworks (e.g., Bootstrap[1]), JavaScript grid-layout libraries (e.g., Masonry[2]), layout templates [25], and visual website builders (e.g., Squarespace[3]) have been developed over the years. For example, Bootstrap abstracts away most of the complexities surrounding RWD and allows developers to define complex responsive grid layouts simply by using a set of predefined CSS classes. While these solutions do reduce development effort, much manual work still remains, and regular maintenance procedures are needed. Moreover, the "one responsive design fits all" approach offers quite limited support for personalization: it merely adjusts the designs to fit the screen width available on the browsing device.

2.2 Web Personalization

The personalization of user interfaces has long been a subject of research. While customization involves users stating their preferences explicitly (for example, via settings), in personalization the interface is automatically changed to suit the preferences and capabilities of the individual user. *Web personalization*, in particular, is defined as "any action that tailors the web experience to a particular user, or a set of users" [23]. The topic has gained significant attention in research on web engineering and intelligent systems [5, 6], and the main artificial intelligence -based methods have been examined in the context of personalization. *Rule-based approaches* have been proposed for presenting content – e.g., for choosing the ordering, emphasis, or scaling of content or how to frame messages or use colors [7, 9, 12]. One clear challenge is to articulate a rule system that can account for numerous individual-specific characteristics and diverse design spaces. *Machine-learning techniques* involve trying to learn personalization policies from a sample dataset [23, 31]. Thus far, they have been limited to selection of contents. A key challenge here is where to obtain a dataset rich enough to cover the vast space of designs and individuals' differences. *Optimization-based methods*, especially model-based approaches that employ predictive models of users for objectives, address this challenge [28, 36]. Predictive models can express individual-level traits either as parameters or as input to the model. Design spaces, on the other hand, are expressed as decisions and constraints in the optimization system. The benefit to this approach is that it can work when there are very few data on the user. For instance, our implementation's predictive models (for saliency and selection time) merely need data on the estimated frequency with which the user accesses items on a page. A recently presented method called Layout as a Service (LaaS) uses integer programming (see below) for web personalization. It takes user click data as input and personalizes a web page to make important content more readily found and used [17]. No previously proposed integer programming -based method exists, however, for RWD, in which the central challenge is to maintain consistency among the web layouts generated.

2.3 Computational Design of Web Pages

In computational design, a full user interface is generated algorithmically. Multi-target or multi-device design is a special case. In *model-based approaches*, transfer is mediated by at least one model. Model-driven engineering (MDE) begins with forming an abstract description of the layout. For instance, Bellucci et al. [2] computed a unified representation for pages that use various

[1]Bootstrap, https://getbootstrap.com/
[2]Masonry, https://masonry.desandro.com/
[3]Squarespace, https://www.squarespace.com/

combinations of web technologies – such as JavaScript, CSS, and HTML. Representing pages in standard notation aids in computational design. Representations in such a form can then be transformed to a target via transformation rules, resulting in a concrete user interface (UI) [8, 12, 18, 38]. Creating these rules is labor-intensive, though, and a recognized bottleneck [38]. This issue parallels that of rule-based and logic-based artificial intelligence in the general case: they scale up poorly.

Model-based approaches using *combinatorial optimization* are attempts to address this issue. Instead of as transformation rules, transfer is formulated as an optimization problem. The solver finds a way to transfer elements of the source layout in a way that optimizes for given objectives. SUPPLE is a pioneering work in this field [14]. It adapts widget layouts by using a perceptual rule and Fitts' law. This approach still requires the designer to specify the functionality of the UI prior to optimization. Our method, in contrast, works directly from a given layout representation, without additional input. Moreover, SUPPLE uses a different model for user perception, which emphasizes font size, whereas ours applies a model related to visual saliency. While SUPPLE does address personalization, it places greater focus on sensorimotor differences than on differences in what users are interested in. Finally, SUPPLE does not deal with the problem of responsive design. Generating multiple consistent-looking pages across different screen sizes requires adding a consistency term to the objective.

While early research into optimization-based approaches used constraints to define the transformations, more recently the predictive models have allowed optimization in light of user-related objectives, such as selection time, clutter, and so on [28]. When a layout is generated, its quality is assessed against these models. In Familiariser [36], a visual search model is parameterized in line with the user's website-visit history and is used browser-side to lay the page out again, for facilitation of search. In LaaS [17], saliency and pointing models inform the generation of web pages for a given user click history. However, these approaches have not been extended to cover RWD. The benefit over MDE is that no abstract model of the user's task or device is needed: the optimizer and objectives operate in the display space ("concrete UI" in MDE terms). Such an approach was long beyond reach, because of the lack of user-related objectives; however, the HCI field has seen a significant number of user-related metrics proposed, some of them adopted from psychology. Studies of layouts, keyboards, and interaction techniques alike are producing mounting empirical evidence that carefully selected metrics can lead to high-quality designs [28].

2.4 Applications of Integer Programming

Our work focuses on the spatial aspects of web layouts; aspects related to colors are not considered. We use *grid layout* as a principle for spatial organization [11]. Grid layout is an expressive and flexible way to present content on a web page. Commonly used in web design, it has recently been studied in the context of UI wireframes [11] and web pages' personalization [17] but not in conjunction with RWD.

Integer programming (IP) is a mathematical optimization method that has been widely adopted for computational generation and adaptation of graphical user interfaces (GUI) [28]. It has been successfully applied for generating GUIs automatically for multiple devices [14] and screen orientations [45], learning cost functions for user interaction -based GUI optimization [15], designing GUIs via high-level constraints [19], optimizing GUIs for aesthetics [44], and generating alternative layouts for design exploration [11]. In IP, a problem is defined as a set of equality or inequality constraints on decision variables, such as elements' coordinates in a layout. Desirable solution properties, such as layout aesthetics, are described by formulating an objective expression using those variables. Zeidler et al. [44] have found constraints to offer a powerful approach to layout management, in that they support defining any type of layout in modular fashion. Because IP is a

form of mathematical optimization, its use cases are limited by the model's complexity and the performance of the available solvers. In the general case, the problem is to fit rectangles of specified widths and heights on a given canvas, finding feasible solutions in which all elements are properly packed in a non-overlapping, non-overflowing layout. In work on combinatorial optimization, grid-based layouts have been studied in the context of bin packing, rectangular packing, and the guillotine-cutting problem [11]. With this paper, we extend IP's application to the case of responsive and personalized web layouts.

2.5 Retargeting of Existing Web Pages

In web page retargeting, an existing web design (the source) is programmatically manipulated to generate an alternative that is closer to a given example (the target). Prior work has used templates [39], visual-search models [36], and machine learning [16] for retargeting content to several alternative designs and screen sizes. In ReMorph [3], elements on a static web page are decomposed into a hierarchical structure from which screen size -specific layouts are generated via complete restructuring of the web page. While these approaches are capable of retargeting a web page, none of them produces fully responsive web pages with smooth transitioning between breakpoint designs.

2.6 Summary

Our technique can be characterized as extending model-based approaches to layout transfer by means of an integer programming -based solution. A key benefit is that layout generation can be achieved with neither extensive manual modeling effort nor large sets of data on the user being required. Another benefit is extensibility: while our demonstration takes layout similarity, elements' selection time and visual saliency as objectives, other objectives and constraints can be plugged in for improved personalization. Recent work elucidates many models that may be suitable [28].

3 THE C-RWD APPROACH

In this section, we present our approach for automated generation of *responsive and personalized web layouts*. We begin with an overview of the approach, followed by a brief description of the Layout as a Service platform and our extensions to it. This is followed by a detailed description of the three key enabling components of our approach.

3.1 Overview

The *computational responsive web design* (C-RWD) approach automates the generation of responsive and personalized web layouts through integer programming. Figure 1 provides a high-level overview of the C-RWD approach. Given an existing web page design along with user interactions and the optimization objectives and constraints (as inputs), our layout optimizer automatically generates a consistent set of personalized designs for targeted breakpoints. The personalized designs are then used to retarget the existing web page for (as output) a fully responsive web page that increases usability.

3.1.1 Extending Layout as a Service. The core of our approach consists of a mathematical optimization model and a web architecture built on top of it. For the latter, we implemented a set of extensions to LaaS [17], a service platform that enables *objective-level personalization of web layouts*. With LaaS, websites' owners are able to control personalization via intuitive objectives that affect the full layout for an individual. Its deployment for existing web pages is straightforward: simply add a short script element to those pages for which usability needs to be improved. The brief

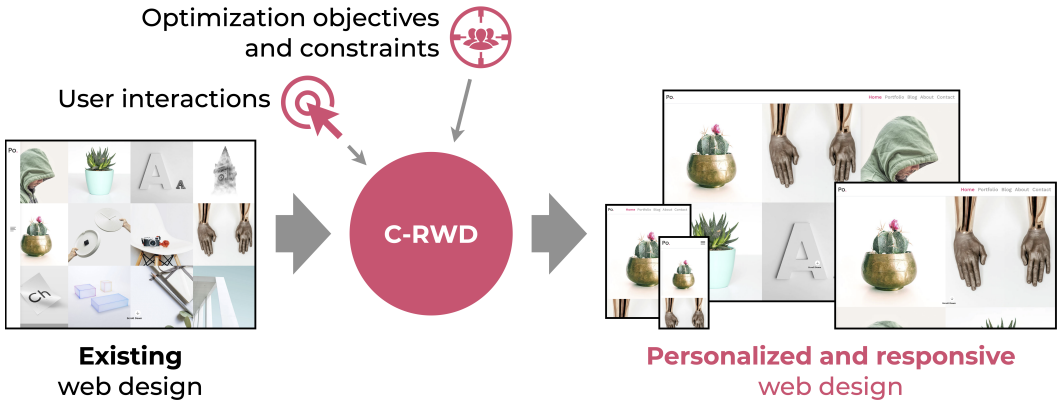

Fig. 1. Overview of the C-RWD approach: an existing web design is automatically retargeted to a fully responsive web page personalized for the user and device.

description below covers the core LaaS components and their function, along with our extensions to support RWD with personalization.

(1) **The layout parser** creates a layout model by parsing the structure and style information of the web page *before* it is displayed to the user. It adds system-specific metadata to HTML elements in the process. We extended the parsing algorithm to simulate key elements in several sizes so that their suitability for optimization can be judged better (see Section 3.2).
(2) **The event logger** listens and records user interactions with the web page, for use as input to layout personalization. Our extension supports a wide range of event types, from clicks and scrolling to page visits and exits. Among the user data collected are the page URL and visit duration, device type and screen dimensions, and each clicked element's identifier and the associated timestamp.
(3) **The design task generator** periodically generates new layout and user -specific design tasks for the layout optimizer. For each design task, it (i) specifies various design objectives and constraints for the web layout and (ii) computes a personalized importance weight for each key element, based on inferred user preferences. Our extension to the generator adds support for *responsive* design tasks, in which target breakpoints are either predefined (configurable) or set dynamically to suit the type and screen size of the user's device.
(4) **The layout optimizer** periodically generates new grid-based web layouts that are optimized and personalized in line with the given design tasks. We extended our integer programming -based model to generate a consistent set of personalized web designs for given breakpoints (see Section 3.3).
(5) **The layout adapter** restructures the web page on the basis of an optimized web layout (if one is available) *before* it is shown to the user. We extended this component to adapt the system-generated web designs at runtime into a fully responsive grid layout (see Section 3.4).

The descriptions below present solutions and key enabling components for (i) *creation of shape sets* (element-specific constraints) for the optimizer, (ii) an IP-based model used for *generation of breakpoint web designs*, and (iii) *retargeting of web pages* to fully responsive ones with a grid layout.

3.2 Creation of Shape Sets

From a visuo-spatial standpoint, a web layout can be viewed as a rectangular canvas with geometric elements placed atop it in a non-overflowing manner. These elements are composed of media content, such as text, images, and videos. As the width of an element changes, its height usually does so too – often in a non-linear manner in response to content-flow changes, such as jumps in text content. We call the set of all possible shapes for an element its *shape set*. Within the IP context, this functions as a set of element-specific constraints, from which the optimizer picks ideal candidates. Therefore, creating shape sets is an essential step in our approach, one handled via the following technique in the layout-parsing stage:

(1) **Shape simulation** changes an element's width to simulate its various shapes in the browser.
(2) **Shape filtering** filters out shapes that overflow or overlap with other elements.
(3) **Shape labeling** uses heuristic rules to label each of the shapes before adding them to the element's shape set.

In the simulation of element shapes, changing the element width pixel by pixel would result in numerous shapes. Most of these shapes are unnecessary, since they produce subtle changes only. Also, this approach would require vast computation power. Therefore, C-RWD uses a set increment of units when changing the element width. For example, to simulate the range of shapes and obtain the full shape set, it might start with 100-pixel element width and increase the width by 100 pixels until reaching 2000 pixels. All these variables, such as the starting value and increment, can be configured as needs dictate.

To guarantee good usability, shapes that overflow or overlap with other elements are filtered out and will not be added to the element's shape set. This further reduces the quantity of shapes in the shape set, thereby reducing the optimizer's search space too.

Through a set of heuristics, each shape is labeled as either an ideal or a non-ideal candidate. Our example employs two heuristic rules (the set is configurable and extendable):

(1) **Characters per line:** For good readability, an element's text content must have 45–80 characters per line [4, 22, 42]. For a single line of text, such as an article title, C-RWD does not impose a lower limit.
(2) **Maximum image height:** Elements' image content must be no more than 700 pixels in height, so that most of the image fits within the device's viewport.

Figure 2 illustrates shape simulation, filtering, and labeling. The 100-pixel-wide shape is filtered out because it presents overflow and readability issues, illustrated by the red lines. The 200- and 400-pixel-wide shapes are added to the element's shape set; the former is labeled as non-ideal for reasons of poor readability, whereas the latter is designated as an ideal shape candidate.

3.3 Generation of Breakpoint Web Designs

The optimization of web layouts is formulated as a mixed-integer linear programming (MILP) model, which provides good computation performance and solution quality. Our implementation allows us to reorder, resize, and reposition elements and guarantees a non-overlapping, non-overflowing grid layout that is perfectly packed. Continuous decision variables represent location details for all four edges of each element while avoiding pixel-level discretization; this is important for the solver's performance. Furthermore, the size of our MILP model depends solely on the number of elements involved, rather than on canvas size. There are three main objectives for the optimization:

(1) **Layout similarity:** The element order should follow that of the original layout.
(2) **Layout quality:** The result should be a perfectly packed symmetric grid layout.

Fig. 2. Shape-set creation: element width is simulated in several sizes, for filtering out infeasible shapes (100px) and distinguish between non-ideal (200px) and ideal (400px) candidates.

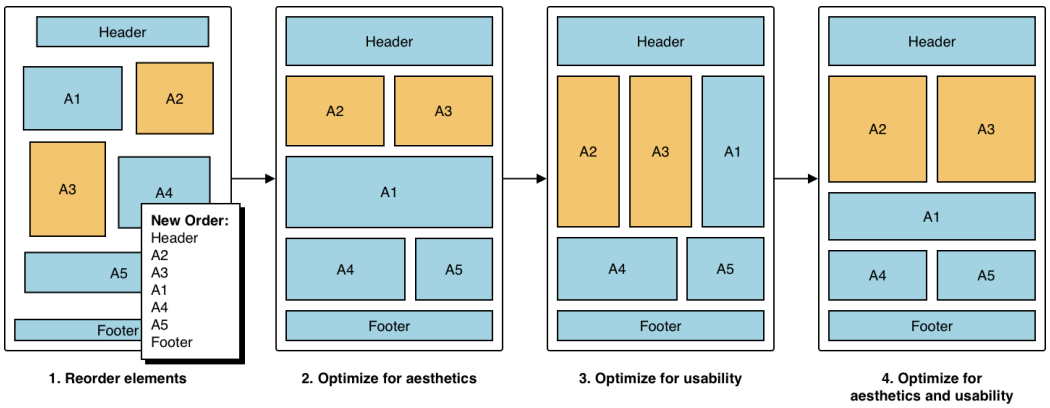

Fig. 3. Layout optimization comprises four phases. The input layout is reordered, after which it is optimized, in three steps: for aesthetics, for usability, and then for both simultaneously. In this example, elements A2 and A3 are considered the most important for the user.

(3) **Layout usability:** The saliency of the most important elements should be maximized, while their selection time should be minimized.

The website's owner can control the optimization result by adjusting objectives 1 and 3. The optimization process is divided into four phases: (i) resolving the element order, (ii) optimizing for layout quality, (iii) optimizing for layout usability, and (iv) optimizing for both layout quality and usability (see Figure 3).

3.3.1 Resolving the Element Order. For the layout to be consistent whatever the responsive version and to improve optimizer performance, the element order is defined irrespective of screen resolution and layout. The elements are ordered in accordance with their assigned importance, original order, or both, as website's owner preferences dictate. The ordinal of each element is calculated by means of the formula:

$$O'_e = S \cdot O_e + (1 - S) \cdot I_e$$

where O_e corresponds to the element's original ordinal, O'_e to the new ordinal, and S to the similarity weight, all normalized to the range $[0, 1]$. The I_e value refers to the element's normalized inverted importance. This means that the greater the importance of an element, the more likely it is for this element to be the first in the layout.

In any case, the hierarchical structure of the layout is kept intact, element order notwithstanding, and elements are reordered only in relation to their siblings. The only exceptions to this are header and footer elements; these are always positioned first and last in the layout, respectively.

3.3.2 Optimizing for Layout Quality. Although our MILP approach can support any type of layout model [44], we focus here on an example implementation of a simplified version of the CSS Flexible Box Layout model [41], where elements are stacked side by side on one or more rows. In this example, we constrain each group to a fully justified, stretched, and perfectly packed grid wherein all elements on a given row are of equal height and the row extends the full width of its container (see Figure 3). Since the element order is predefined, the optimizer, in essence, selects the "line breaks" for the layout. In addition to the overall layout constraints, the size of each element comes from a predefined set of shapes. This is to prevent content overflow.

The constraints and objects are applied iteratively to each level of the layout hierarchy. Thus, the following constraints and objectives are applied to each group of sibling nodes in the layout.

As the element order is known, overlap is prevented by placing positional constraints on consecutive elements. Consecutive elements on the same row are required to share top and bottom edge coordinates, and the right edge of the first element must be flush against the left edge of the second element (Constraints 8–11). If consecutive elements fall on different rows, they must be placed at the end and the beginning of their rows, respectively, and the first element bottom edge must be aligned with the top edge of the second one (Constraints 4–7). This results in a non-overlapping, non-overflowing, well-aligned and perfectly packed grid.

We define following binary decision variables for each element:

- $\rho \in \{0, 1\}$: element is on the same row as the previous element
- $\upsilon \in \{0, 1\}$: element has the same width as the previous element
- $\xi \in \{0, 1\}$: element is not on the same row or is not the same width as the previous element

We define following continuous decision variables for each element:

- λ: element/group left edge coordinate
- τ: element/group top edge coordinate
- γ element/group right edge coordinate
- β: element/group bottom edge coordinate

Finally, we define following constants:

- w_l: total layout width, i.e. the breakpoint
- $\max(h_l)$: maximum layout height

The symbols i and g refer to the element and group indices, respectively, and $\mathbb{E}_g$ denotes the set of elements in a given group. For all $i \in \{1, \ldots, |\mathbb{E}_g|\}$, we require that the element is placed within its

group:

$$\begin{aligned} \lambda_i &\geq \lambda_g \\ \tau_i &\geq \tau_g \\ \gamma_i &\leq \gamma_g \\ \beta_i &\leq \beta_g \end{aligned} \tag{1}$$

We define layout quality $\mathbb{Q}$ as a regular and symmetric layout, and optimize for it by minimizing the number of elements that have a different width than their previous sibling on the same row:

$$\mathbb{Q} = \min \sum_{i=2}^{|\mathbb{E}_g|} \xi_i \tag{2}$$

For all $i \in \{2, \ldots, |\mathbb{E}_g|\}$, we require that $\xi_i = 0$ only if the element is the first on its own row, or it is the same width as the previous element:

$$\xi_i \geq \rho_i - \upsilon_i \tag{3}$$

If the element is on a different row than the previous element, i.e. the first on its own row, its left edge must align with the group left edge, its top edge must align with the bottom edge of the previous row, and the right edge of the previous element must align with right edge of the group:

$$\begin{aligned} \neg\rho_i \implies & \lambda_i = \lambda_g \\ & \wedge\ \tau_i = \beta_{i-1} \\ & \wedge\ \gamma_g = \gamma_{i-1} \end{aligned}$$

which can be more efficiently computed using the following linear constraints:

$$\lambda_i \leq \lambda_g + \rho_i w_l \tag{4}$$

$$\gamma_{i-1} \geq \gamma_g - \rho_i w_l \tag{5}$$

$$\tau_i \leq \beta_{i-1} + \rho_i \max(h_l) \tag{6}$$

$$\tau_i \geq \beta_{i-1} - \rho_i \max(h_l) \tag{7}$$

Conversely, if the element is placed on the same row as the previous element:

$$\begin{aligned} \rho_i \implies & \tau_i = \tau_{i-1} \\ & \wedge\ \beta_i = \beta_{i-1} \\ & \wedge\ \lambda_i = \gamma_{i-1} \end{aligned}$$

or, using linear constraints:

$$\tau_i \leq \tau_{i-1} + (1 - \rho_i) \max(h_l) \tag{8}$$

$$\tau_i \geq \tau_{i-1} - (1 - \rho_i) \max(h_l) \tag{9}$$

$$\beta_i \leq \beta_{i-1} + (1 - \rho_i) \max(h_l) \tag{10}$$

$$\beta_i \geq \beta_{i-1} - (1 - \rho_i) \max(h_l) \tag{11}$$

If $\upsilon_j^i = 1$, the element must have equal width with the previous element:

$$\upsilon_i \implies w_i = w_{i-1}$$

or, using linear constraints:

$$w_i \leq w_{i-1} + (1 - v_i)w_l \tag{12}$$

$$w_i \geq w_{i-1} - (1 - v_i)w_l \tag{13}$$

3.3.3 Optimizing for Layout Usability. After optimizing for layout quality $\mathbb{Q}$, the model objective is replaced with layout usability $\mathbb{U}$, which consists of maximizing the saliency σ and minimizing the selection time δ for important elements in the layout. The complete objective for this stage is:

$$\mathbb{U} = \min \sum_{i=1}^{|\mathbb{E}|} I_i \cdot (\delta_i - \sigma_i) \tag{14}$$

We use Fitts' law to compute the time required to reach a specific element on the screen, which is widely used in model-based optimization as an objective function (see [28]). Selection time ST is a function of target distance D and size W:

$$\mathrm{ST} = a + b \log_2(\frac{2D}{W})$$

In our examples, we assume that the user starts scanning the screen from the top-left corner, but our implementation supports any point as a starting point. To improve the optimizer performance, we use a piece-wise-linear approximation of the logarithm function. For rectangular objects, ST in the in direction of movement equals the ST along the worse of the cardinal directions. Furthermore, for the optimization, the constants a, b, and 1 are not relevant. Our MILP implementation therefore defines the following two constraints:

$$\delta_i \geq \log_2 \frac{\lambda_i + \gamma_i}{2} - \log_2 w_i \tag{15}$$

$$\delta_i \geq \log_2 \frac{\tau_i + \beta_i}{2} - \log_2 h_i \tag{16}$$

Saliency refers to how attention-grabbing an element is given the rest of the page [32]. To measure saliency, we use the area of an element as a proxy, although other factors, such as color could be used. As the Weber–Fechner law states that perceived intensity is proportional to the logarithm of the stimulus, we use the logarithm of the element's area as the saliency function: $\sigma_i = \log_2(h_i \cdot w_i)$ Although the layout quality $\mathbb{Q}$ is removed from the objective function, a new constraint is added to the model that prevents the optimiser from decreasing the achieved level of quality:

$$\mathbb{Q} \leq \mathbb{Q}_1 \tag{17}$$

where $\mathbb{Q}_1$ corresponds to the value achieved in the previous stage. No other restrictions are added to the model at this stage.

3.3.4 Optimizing for Both Layout Quality and Usability. If the optimizer reaches an optimal solution in the previous stage, that solution is considered the final layout and this stage is skipped. If an optimal solution has not yet been found, the objectives from the two previous stages are combined to achieve a balance between the aesthetics of the layout and its usability. First, the usability objective from the previous stage is normalized to the range $[0, 1]$, and the full objective of the final optimization stage becomes $\min(\mathbb{Q} + \mathbb{U}')$, where $\mathbb{U}'$ corresponds to the normalized layout quality. The full objective of the final optimization stage becomes:

$$\min(\mathbb{Q} + \mathbb{U}') \tag{18}$$

As $\mathbb{Q}$ takes an integer value, more weight is given for improving the layout aesthetics. At the same time, a constraint is added to allow the solver to minimally decrease the achieved usability level if it can find a more aesthetic layout:

$$\mathbb{U}' \leq 1.1 \tag{19}$$

This constraint essentially allows the usability objective $\mathbb{U}$ to be decreased 10% of the difference between what was reached at the previous stage and what was estimated as optimal.

3.3.5 Adjusting Results. If the design task instance prioritizes selection time, the optimizer attempts to minimize the predicted time required for important elements. The most obvious effect is that very important elements may become larger and move closer to the top-left corner of the web page. However, the reading order of the elements will not be changed. If the most important elements are not the first elements (i.e., when the similarity weight is high) the first elements can be expected to decrease in size, if their shape sets allow it. Prioritizing saliency in turn will likely lead to increasing the size of the most important elements.

3.4 Retargeting of Web Pages

After the solver produces optimized web designs at multiple breakpoints, C-RWD needs to retarget the original web page and synthesize the series of designs to generate the final responsive web layout. Simultaneously, since the optimizer provides only static page design for each breakpoint width, C-RWD must consider the usability of responsive interfaces also beyond them. The main steps of web page retargeting in C-RWD are the following:

(1) **Layout conversion:** Convert the optimizer-supplied web page designs at different breakpoints into designs consumed by the underlying the responsive-layout framework, and integrate them into a responsive design by dynamically creating media queries.
(2) **Layout adaptation:** Proceeding from the responsive web page designs produced in the conversion step, adapt the page to the corresponding design and improve the page's usability at different widths by manipulating the page elements' style and position after resizing of the web page.
(3) **Navigation-bar generation:** Adapt the navigation bar by generating a responsive one on the basis of the configuration.
(4) **Image cropping:** For image elements, use smart cropping to improve the presentation of key information at various sizes.

In the layout-conversion process, for the optimized design at each breakpoint, C-RWD uses CSS Grid [43] as a responsive-layout framework for conversion of the pixel-based element-position information provided by the optimizer into an element position based on grid rows and columns. The main reason C-RWD uses CSS Grid is that, unlike other responsive-layout frameworks, it can control element position from two dimensions to meet complex design requirements and support the dynamic changes of element height caused by content changes. Figure 4 illustrates a post-conversion grid-based web page. Simultaneously, C-RWD can support responsive-layout frameworks very different from CSS Grid, such as Flexible Box Layout [41] and Positional Layout, for handling other specific responsive-layout needs. Then, C-RWD uses the matchMedia API [40] for dynamically adding the corresponding media query to each breakpoint to monitor the change in page size and trigger layout adaptation.

When the web page changes in response to user behavior such as window resizing or mobile-device rotation, C-RWD performs real-time adaptation of the page in accordance with the converted grid design corresponding to the breakpoint triggered. If the post-resizing page width triggers a breakpoint different from the current one, the grid rows' and columns' number and the page

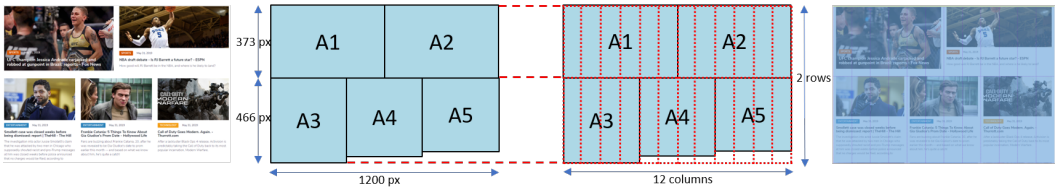

Fig. 4. Conversion of an optimized pixel-based design to a grid-based design.

elements' position in the grid get updated. Likewise, font size and image aspect ratios for the various elements will be updated on the basis of the converted design. If there is no triggering of a different breakpoint after resizing or rotation, the page continues using the current optimized design and takes advantage of the full viewport size – as with a fluid grid. The transition is smooth thanks to the adapted web page's use of the grid framework, and most elements are adjusted to use relative position units.

Special components of the web page, such as its navigation bar, are adapted in accordance with the configuration. For example, if a static desktop-version website is used as input and the configuration indicates to adapt its navigation bar to be responsive, the static navigation bar will be adjusted for responsiveness, with a hamburger menu used in cases of a narrow page space. As for image elements on the web page, they may no longer display important information, on account of their size reduction. Therefore, C-RWD uses smart cropping and face recognition, provided by smartcrop.js[4] and tracking.js,[5] to crop images for various size conditions and optimize image display.

Through this sequence of steps, C-RWD completes the retargeting of the input web page and renders a personalized, responsive web layout for users.

3.5 Implementation

As noted in the overview section, our C-RWD approach is implemented as an extension to the LaaS platform. All of the client-side components are handled by vanilla JavaScript, while all server-side ones are implemented in Python 3.7. We use Gurobi 9.0 as the MILP solver and MongoDB to store data, such as layouts and user interactions.

4 EVALUATION

In this section, we evaluate the efficacy of C-RWD in two ways. Firstly, we demonstrate its two main technical capabilities separately, through practical examples. Secondly, we assess the quality of generated web designs empirically in two user studies, one with end users and the other with designers.

4.1 Examples

4.1.1 Breakpoint Web Designs. Figure 5 depicts a retargeting scenario in which a non-responsive (i.e., static) mobile-layout web page design for a portfolio website (at left) is adapted to several screen sizes on the basis of targeted breakpoints: mobile, tablet, laptop, and desktop (at right). Target breakpoints can be manually configured by the website's owner to match the screen sizes of the

[4]smartcrop.js, https://github.com/jwagner/smartcrop.js
[5]tracking.js, https://trackingjs.com/

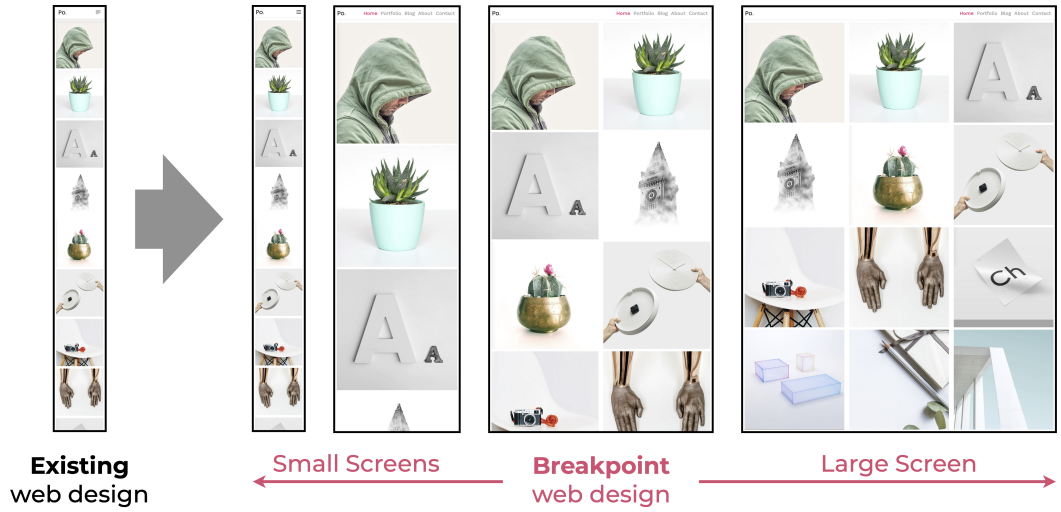

Fig. 5. Breakpoint web designs: an existing web design (here, for a mobile device) can be retargeted to other screen sizes on the basis of predefined or automatically inferred breakpoints.

most popular devices currently on the market[6] or automatically inferred from user interactions via data including, among other details, the type of the device and its screen and viewport dimensions. For example, C-RWD can retarget an existing web page design to the (portrait and landscape) width of the user's mobile device, to improve website responsiveness further. Addressing this scenario with traditional RWD solutions (see Section 2.1) is very time-consuming and challenging since they use fixed breakpoint widths, the changing of which, if possible in the first place, typically requires manual redesign effort.

4.1.2 Personalized Web Designs. Figure 6 illustrates how C-RWD can be used to generate different levels of personalized web design for a custom news-aggregator website on the basis of user-specific interests. In this personalization scenario, the existing web page design that is used as input (left) is fully responsive and viewed on a laptop device with a window width set to 1200 pixels. While the already responsive website automatically adapts the content to the available screen real estate, it fails to support the user's unique interests (see the red circles in the figure). With C-RWD, the website's owner can control distinct optimization objectives (e.g., layout similarity, key elements' visual saliency and selection time) and their effectiveness, thereby supplying personalized web designs (right) to enhance user experience further.

4.2 Study 1: Perceived Quality by End Users

The aim of this study was to evaluate how *end users* perceive the quality of various web designs. In particular, we sought to determine whether our integer programming -based approach can generate high-quality web designs for the range of screen sizes considered (mobile, tablet, laptop, and desktop). In this ratings-based online study, participants were asked to rate web designs for several screen sizes, which were created by means of three distinct layout-design methods: *Original*, *Optimized* (our approach), and *Masonry*. We had two main hypotheses:

H1. Layout-design method impacts end-user ratings.

[6]Per the StatCounter Screen Resolution Stats Worldwide data for Jan. 2010 – Dec. 2020, available at https://gs.statcounter.com/screen-resolution-stats#monthly-201001-202012.

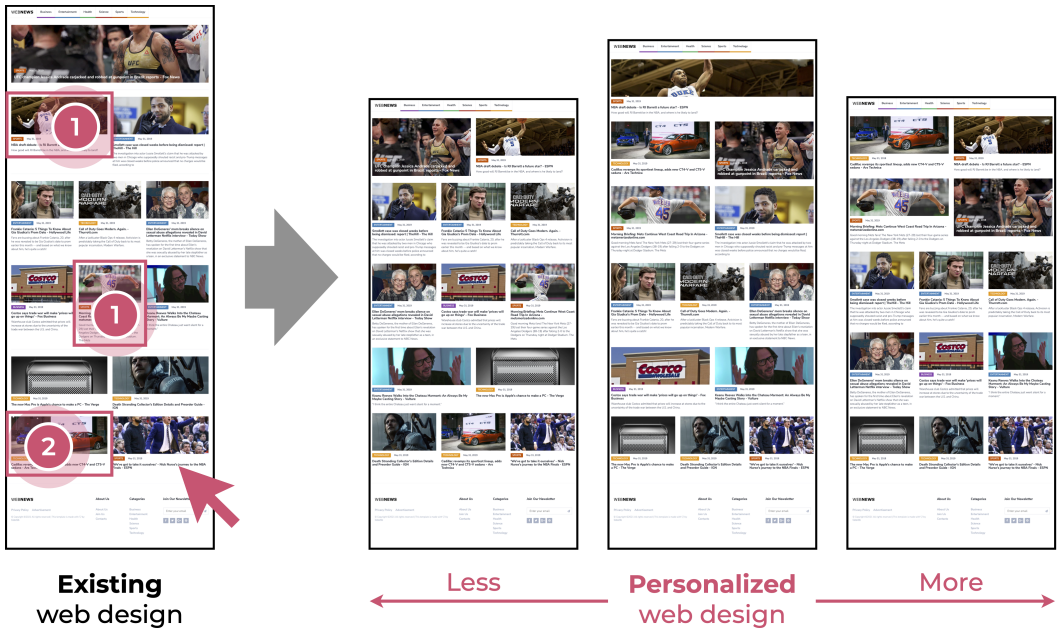

Fig. 6. Personalized web designs: an existing web design (here, for a laptop) can be personalized in line with given user interactions and controllable optimization objectives.

H2. Screen size has an effect on the impact of layout-design method on end-user ratings.

4.2.1 Materials. Four fully responsive websites based on popular modern Bootstrap templates were created for the study (see Figure 7). We selected the websites to support generalization yet be unfamiliar to participants, to avoid bias in the results. The websites thus created represented the *Original* layout-design method. For the *Optimized* method, a set of website variants was created with each website containing the LaaS script and annotations for key elements. Then, the website variants were loaded in the Chrome web browser in mobile-device mode (320px) for purposes of obtaining the input page designs for our layout optimizer. Finally, the optimizer generated optimized page designs for predefined breakpoints (one per screen size), using neutral objective values and a uniform distribution of user interests over all key elements. For the *Masonry* layout-design method, another set of website variants was created, from the same set of originals, with the Isotope[7] JavaScript grid-layout library and annotations for key elements along with JavaScript code and CSS styles to initialize the library in Masonry[8] layout mode. In three of the four website variants, key elements used fixed-width element sizing; otherwise, 100% width was used. All key-element containers were centered on the screen. Figure 8 shows an example website created with each layout-design method.

We took a *full-page screenshot* of all websites at each screen size[9] created with each of the layout-design methods, for a total of 48 screenshots (4 websites × 4 screen sizes × 3 layout-design methods). In addition, a full-page screenshot of two other websites was taken for attention-check

[7]Isotope, https://isotope.metafizzy.co/
[8]Masonry, https://masonry.desandro.com/
[9]We used 320px, 768px, 1366px, and 1920px, for mobile, tablet, laptop, and desktop layouts, respectively. The values were based on StatCounter Screen Resolution Stats Worldwide data for Dec. 2020 [34].

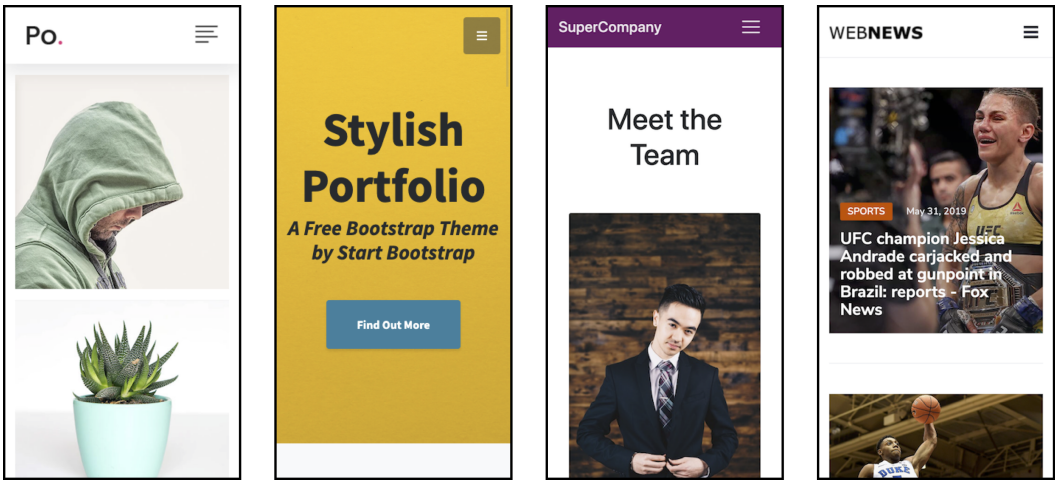

Fig. 7. A set of fully responsive websites created for the study (the mobile view is shown here).

questions. All screenshots were centered on a fixed-width (but variable-height) gray canvas, to create equal-width web-design images for presentation in our study. All screenshots and web-design images used in the study are provided in the supplemental material for this paper.

4.2.2 Participants. Participants were recruited via Prolific[10], an online recruitment platform that academics use for high-quality participant samples [29, 30]. For eligibility, users were required to be at least 18 years of age and legally competent, generally healthy and with normal or corrected-to-normal vision, and able to participate with a laptop or desktop computer.

In total, 106 people completed the study. Each was paid £1.25 for participating, which corresponds to an hourly rate of £7.50 (per recommendation). Twenty of them failed attention checks (i.e., they did not respond correctly to the prompt "Give the below web design a rating of..") so were excluded from further analyses. Therefore, for the final sample, $N = 86$, with 53 male and 33 female participants, all between 18 and 68 years of age ($M = 27.7,\ SD = 10.4$) and having a Prolific score of 96/100 or higher.

4.2.3 Procedure and Design. Participants were given a link to complete an online survey on the SurveyMonkey platform.[11] The survey began with a description of the study and an informed-consent form, followed by items for provision of the participant's Prolific ID and basic demographic information (age, gender, and proficiency in English). In the main part of the survey, participants were asked to rate the quality of all 50 web designs as they perceived it on a scale of -3 (extremely low quality) to 3 (extremely high quality). We reminded participants that "quality" is subjective and noted that "When providing your ratings, you can pay attention to, for instance, how well the graphical elements are aligned, use of white space, and text readability." We used a *within-subject* experiment design and randomized the order of successively presented web designs. We obtained data from, in all, 4128 trials (excluding attention checks), with the following experiment design: 86 participants × 3 layout-design methods × 4 websites × 4 screen sizes (mobile, tablet, laptop, and desktop). Upon completion, participants were thanked and directed back to Prolific to claim their compensation.

[10]Prolific, https://www.prolific.co/
[11]SurveyMonkey, https://www.surveymonkey.com/

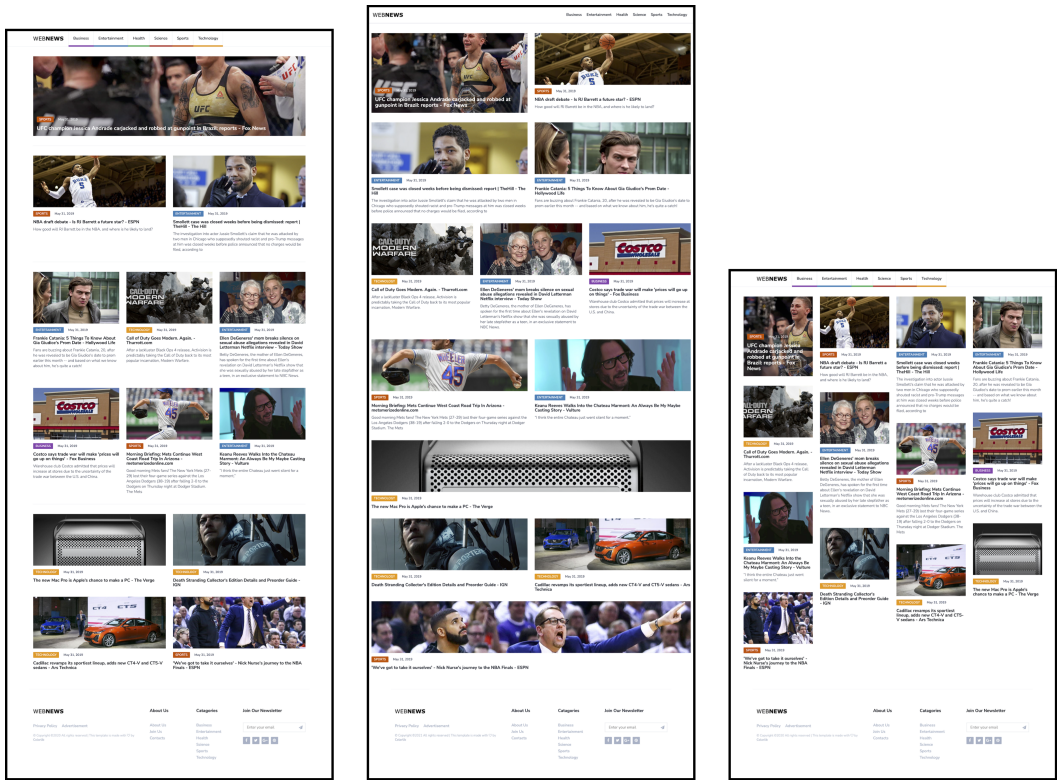

Fig. 8. Laptop views of an example website created via the three distinct layout-design methods: *Original* (left), *Optimized* (middle), and *Masonry* (right).

4.2.4 Data Analysis. For testing H1 and H2, we used multilevel regression with rating as the dependent variable and with layout-design method and screen size as fixed effects. We added participant as a random intercept. We used the R packages `lme4`, `lmerTest`, and `lsmeans` to run the analyses. For H1, we used *post hoc* pairwise-comparison statistical tests (the Tukey method), but our computations for the more detailed analysis of H2 involved only omnibus testing, accompanied by visual inspection of the differences – on account of the large number of possible pairwise comparisons.

4.2.5 Results. The grand mean ratings for the *Original*, *Optimized*, and *Masonry* method are, respectively, 0.84 ($SD = 0.69$), 0.78 ($SD = 0.67$), and 0.47 ($SD = 0.38$). The effect of layout-design method on ratings (H1) is statistically significant, $F(2,\ 935) = 33.1,\ p < .001$. In the *post hoc* comparisons, the difference between *Masonry* and *Original* is $d = 0.34,\ t(935) = 7.6,\ p < .001$, and that between *Masonry* and *Optimized* is $d = 0.28,\ t(935) = 6.4,\ p < .001$. The difference between *Original* and *Optimized* is not statistically significant.

The grand mean ratings for the individual layout-design methods, by screen size, are listed in Table 1, with Figure 9 providing more detailed visualization. The interaction effect between screen size and layout-design method (H2) is statistically significant, $F(6,\ 935) = 11.2,\ p < .001$.

Table 1. Mean end-user rating (and SD), by screen size, with the scale -3 (extremely low quality) to 3 (extremely high quality).

Layout-design method	Screen size			
	Mobile	Tablet	Laptop	Desktop
Original	-0.13 (1.23)	0.81 (0.87)	1.30 (1.26)	1.38 (0.80)
Optimized	-0.13 (1.23)	0.69 (0.95)	1.19 (0.72)	1.36 (0.84)
Masonry	-0.09 (1.18)	0.69 (0.94)	0.71 (0.77)	0.59 (0.91)

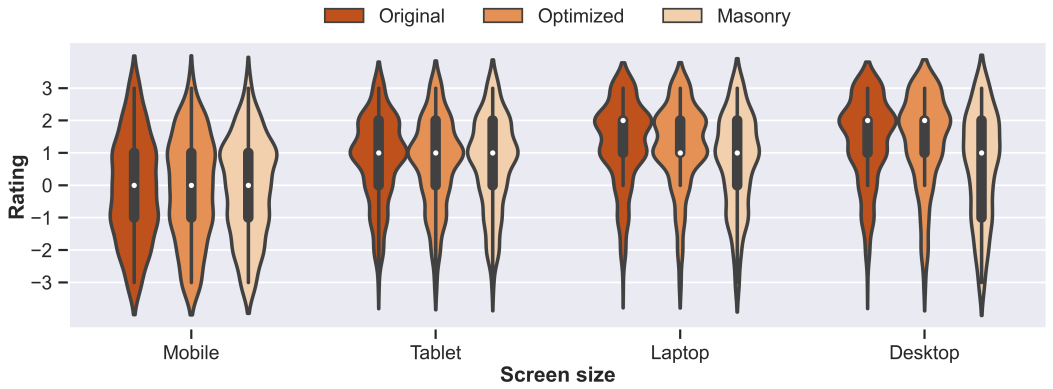

Fig. 9. Results from the survey of end users.

4.3 Study 2: Perceived Quality by Designers

With the second study, the aim was to evaluate how *designers* perceive the quality of various web designs. The methods and data-analysis procedures were similar to those in Study 1 except that participants reported themselves to be proficient in responsive design and/or user interface (UI) design.

4.3.1 Participants. Again, we handled recruitment via Prolific. We used the same prescreening criteria and the same compensation as in Study 1, with the exception that eligibility was contingent on being proficient in responsive design and/or UI design and not having taken part in our previous study.

In total, 100 individuals completed the study, of whom 17 failed attention checks and an additional 19 did not meet the criteria (i.e., they had, at best, beginner-level knowledge of both responsive design and UI design). After their exclusion from further analyses, the final sample consisted of 64 participants (52 male, 10 female, 2 other) aged between 18 and 58 years ($M = 26.2$, $SD = 8.2$) with a Prolific score of 95/100 or higher. Most participants were IT professionals, and they worked in various roles, such as web developer, graphic designer, or software engineer. Their (self-reported) professional experience in responsive/UI design ranged from zero to 15 years ($M = 2.4$, $SD = 2.8$), and most reported having intermediate-level or advanced skills in UI design, responsive design, or both (see Table 2).

Table 2. Participants' level of skill in UI design and responsive design.

Skill level	UI design	Responsive design
Expert	3	1
Advanced	17	20
Intermediate	43	35
Beginner	1	8
No knowledge	0	0

Table 3. Mean designer rating (and SD) for each screen size, on a scale of -3 to 3 (for extremely low to extremely high quality).

Layout-design method	Screen size			
	Mobile	Tablet	Laptop	Desktop
Original	0.50 (1.12)	1.06 (0.71)	1.20 (0.81)	1.31 (0.76)
Optimized	0.41 (1.15)	0.91 (0.79)	1.18 (0.81)	1.20 (0.82)
Masonry	0.53 (1.13)	0.89 (0.70)	0.72 (0.82)	0.43 (1.08)

4.3.2 Procedure and Design. The material used and the study procedure and design were similar to what we used for Study 1, with two exceptions: (i) we collected additional demographic information (profession, level of skill in responsive design, UI design skill level, and years of professional experience) from the participants, and (ii) the data come from, in total, 3072 trials (64 participants $\times$ 3 layout-design methods $\times$ 4 websites $\times$ 4 screen sizes) without attention checks.

4.3.3 Results. The *Original*, *Optimized*, and *Masonry* layout-design method had a grand mean rating of 1.02 ($SD = 0.36$), 0.93 ($SD = 0.37$), and 0.64 ($SD = 0.21$), respectively. Thus, the effect of layout-design method on ratings (H1) is statistically significant, $F(2,\ 693) = 24.7,\ p < .001$. In the *post hoc* comparisons, the difference between *Masonry* and *Original* is $d = 0.39,\ t(693) = 6.7,\ p < .001$, and that between *Masonry* and *Optimized* is $d = 0.30,\ t(693) = 5.1,\ p < .001$. No statistically significant difference was found between *Original* and *Optimized*.

The grand mean ratings for the three layout-design methods are listed, by screen size, in Table 3, with a more detailed graphical view provided by Figure 10. The interaction effect between screen size and layout-design method (H2) is statistically significant, $F(6,\ 693) = 8.8,\ p < .001$.

5 DISCUSSION

Our integer programming -based approach generates responsive and personalized web layouts that were viewed favorably by human evaluators. The ratings given both by end users (in Study 1) and by designers (in Study 2) for the optimized designs were comparable to the subjects' ratings for the original designs and higher than those they gave for a popular baseline method of layout design. Furthermore, our results highlight the impact of screen size. With smaller screens, the difference between the methods in the web designs' perceived quality was smaller than with larger screens. It is possible that less variability appears with smaller screen sizes and that only larger ones can bring

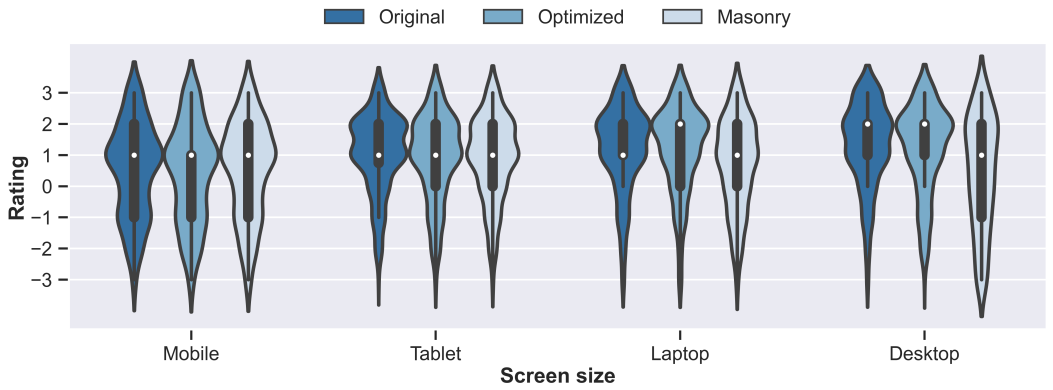

Fig. 10. Results from the survey of designers.

out differences among methods. In future, researchers could attempt to disentangle the effects of screen size vs. layout method.

We can identify several possible steps for efforts to improve the approach. Firstly, our experiments involved a fairly simple, one-item metric for perceived quality. Given the amount of research into ways of measuring perceived usability and aesthetics of websites [24], through various of these it should be possible to shed greater light on specific design choices' precise impact on how the website gets perceived visually. This would not only aid in assessing the method but also steer the objectives' design toward fuller coverage of visual aesthetics. Secondly, we tested our method with a representative but rather limited set of websites; more variety, especially in terms of the types of sites presented, is required for further testing of such methods.

Finally, we used only full-page screenshots of the web designs and asked participants to rate their quality. Having subjects actually interact with the designs via assigned tasks and then collecting both objective and subjective usability feedback would yield richer evaluation and comparison of various layout-design methods. While it is known that usability may affect perceived aesthetics, the details of this effect are less clear – for instance, how perceptions may change as time passes [37].

We can point to several avenues of research whereby our work's applicability could be expanded. For example, the current implementation is limited to server-side rendered web pages only. While many platforms (e.g., WordPress[12]) and JavaScript UI libraries (e.g., React[13] and Vue.js[14]) use and support server-side rendering, support for more dynamic, client-side rendering of web pages would further improve our approach's compatibility with a broad range of website types. One way of realizing this support could entail using MutationObserver [46] to track DOM changes and applying layout adaptation only after client-side rendering is complete. Also, incorporating the latest advancements in the field of adaptive user interfaces would enhance the adaptation strategy so as to consider both end-user costs and benefits [35].

One shortcoming of our approach is using heuristics within elements but combinatorial optimization for the grid layout. Though the results are of high quality, they could be further improved by optimizing both, jointly. Recent work has looked at hierarchical optimization approaches for web pages [33]; however, these approaches are still unable to deal with realistic problem sizes. Another challenge to be tackled is addressing how to preserve relationships between UI elements.

[12]WordPress, https://wordpress.com/
[13]React, https://reactjs.org/
[14]Vue.js, https://vuejs.org/

For example, "OK" and "Cancel" should be adjacent to each other. While such relationships can be conveniently expressed to the optimizer as constraints, how to identify all such vital relationships in the first place is less obvious. One way forward might be to study data-driven techniques, for deriving such constraints from datasets (e.g., [13]).

6 CONCLUSION

We have presented a promising method for computationally generating responsive web layouts, a challenging computational problem that we believe is hard to address with data-driven methods. Integer programming presents the key benefit that certain relatively universal human factors can be ensured in the absence of any training data. Moreover, when said data for a given user (or user group) *is* available, it can be exploited for further personalization of the layout. The proposed approach is flexible, in the sense that new constraints and objectives can be plugged in and controlled by the website's owner. Although the use of integer programming for layout problems has seen advances [28], applications for responsive web design have remained missing because of the additional requirement for consistency among the designs generated. Moreover, the state of the art of web engineering has lacked techniques for adapting computed responsive layouts at runtime. The user studies reported upon here suggest that our approach can automatically create layouts that, if not on par with those created by designers, are at least very close to humans' and that clearly surpass those created by a popular technique (Masonry) on the market. To conclude, we consider the methods presented here a step toward a future wherein designers and developers can be relieved of much of the manual burden involved in creating responsive websites.

ACKNOWLEDGMENTS

This work was funded by Technology Industries of Finland (Grant No. 700047, "Self-Optimizing Web Pages") and the Academy of Finland (Grant No. 328813, "Human Automata").

REFERENCES

[1] Shaun Anderson. 2020. What are the Best Screen Sizes for Responsive Web Design? https://www.hobo-web.co.uk/best-screen-size/

[2] Federico Bellucci, Giuseppe Ghiani, Fabio Paternò, and Claudio Porta. 2012. Automatic Reverse Engineering of Interactive Dynamic Web Applications to Support Adaptation across Platforms. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces* (Lisbon, Portugal) *(IUI '12)*. Association for Computing Machinery, New York, NY, USA, 217–226. https://doi.org/10.1145/2166966.2167004

[3] Gilbert Louis Bernstein and Scott Klemmer. 2014. Towards Responsive Retargeting of Existing Websites. In *Proceedings of the Adjunct Publication of the 27th Annual ACM Symposium on User Interface Software and Technology* (Honolulu, HI, USA) *(UIST '14 Adjunct)*. Association for Computing Machinery, New York, NY, USA, 119–120. https://doi.org/10.1145/2658779.2658805

[4] Robert Bringhurst. 2004. *The elements of typographic style.* Hartley & Marks, Vancouver, Canada.

[5] Peter Brusilovsky. 2003. From Adaptive Hypermedia to the Adaptive Web. In *Mensch & Computer 2003: Interaktion in Bewegung*, Gerd Szwillus and Jürgen Ziegler (Eds.). Vieweg+Teubner Verlag, Wiesbaden, Germany, 21–24. https://doi.org/10.1007/978-3-322-80058-9_3

[6] Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. 2007. *The adaptive web: Methods and strategies of web personalization.* Vol. 4321. Springer Science & Business Media, Berlin, Germany. https://doi.org/10.1007/978-3-540-72079-9

[7] Andrea Bunt, Giuseppe Carenini, and Cristina Conati. 2007. Adaptive Content Presentation for the Web. In *The adaptive web: Methods and strategies of web personalization*, Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl (Eds.). Springer, Berlin, Germany, 409–432. https://doi.org/10.1007/978-3-540-72079-9_13

[8] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. 2003. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (2003), 289–308. https://doi.org/10.1016/S0953-5438(03)00010-9

[9] Luca Chittaro. 2016. Tailoring Web Pages for Persuasion on Prevention Topics: Message Framing, Color Priming, and Gender. In *Persuasive technology*, Alexander Meschtscherjakov, Boris De Ruyter, Verena Fuchsberger, Martin Murer,

and Manfred Tscheligi (Eds.). Springer International, Cham, Switzerland, 3–14. https://doi.org/10.1007/978-3-319-31510-2_1

[10] Cisco. 2019. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022. https://s3.amazonaws.com/media.mediapost.com/uploads/CiscoForecast.pdf

[11] 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

[12] Paul De Bra. 2017. Challenges in User Modeling and Personalization. *IEEE Intelligent Systems* 32, 5 (2017), 76–80. https://doi.org/10.1109/MIS.2017.3711638

[13] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (Québec City, QC, Canada) *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 845–854. https://doi.org/10.1145/3126594.3126651

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

[15] 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* (Seattle, WA, USA) *(UIST '05)*. Association for Computing Machinery, New York, NY, USA, 173–182. https://doi.org/10.1145/1095034.1095063

[16] 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* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 2197–2206. https://doi.org/10.1145/1978942.1979262

[17] 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, Cham, Switzerland, 19–26. https://doi.org/10.1007/978-3-030-50578-3_2

[18] Zhen Liu, Anand Ranganathan, and Anton Riabov. 2007. Modeling Web Services Using Semantic Graph Transformations to aid Automatic Composition. In *IEEE International Conference on Web Services* (Salt Lake City, UT, USA) *(ICWS '07)*. IEEE, 78–85. https://doi.org/10.1109/ICWS.2007.129

[19] Christof Lutteroth, Robert Strandh, and Gerald Weber. 2008. Domain Specific High-Level Constraints for User Interface Layout. *Constraints* 13, 3 (2008), 307–342. https://doi.org/10.1007/s10601-008-9043-2

[20] Ethan Marcotte. 2010. Responsive Web Design. https://alistapart.com/article/responsive-web-design/

[21] Jennifer Mazzei. 2012. Responsive Web Design. https://www.businessnhmagazine.com/article/responsive-web-design

[22] Aliaksei Miniukovich, Michele Scaltritti, Simone Sulpizio, and Antonella De Angeli. 2019. Guideline-Based Evaluation of Web Readability. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland, UK) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300738

[23] Bamshad Mobasher, Robert Cooley, and Jaideep Srivastava. 2000. Automatic Personalization Based on Web Usage Mining. *Commun. ACM* 43, 8 (2000), 142–151. https://doi.org/10.1145/345124.345169

[24] Morten Moshagen and Meinald T. Thielsch. 2010. Facets of Visual Aesthetics. *International Journal of Human–Computer Studies* 68, 10 (2010), 689–709. https://doi.org/10.1016/j.ijhcs.2010.05.006

[25] Michael Nebeling, Fabrice Matulic, Lucas Streit, and Moira C. Norrie. 2011. Adaptive Layout Template for Effective Web Content Presentation in Large-Screen Contexts. In *Proceedings of the 11th ACM Symposium on Document Engineering* (Mountain View, CA, USA) *(DocEng '11)*. Association for Computing Machinery, New York, NY, USA, 219–228. https://doi.org/10.1145/2034691.2034737

[26] Michael Nebeling and Moira C. Norrie. 2013. Responsive Design and Development: Methods, Technologies and Current Issues. In *Web engineering*, Florian Daniel, Peter Dolog, and Qing Li (Eds.). Springer, Berlin, Germany, 510–513. https://doi.org/10.1007/978-3-642-39200-9_47

[27] Antti Oulasvirta. 2019. It's Time to Rediscover HCI Models. *Interactions* 26, 4 (2019), 52–56. https://doi.org/10.1145/3330340

[28] 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. https://doi.org/10.1109/JPROC.2020.2969687

[29] Stefan Palan and Christian Schitter. 2018. Prolific.ac – A Subject Pool for Online Experiments. *Journal of Behavioral and Experimental Finance* 17 (2018), 22–27. https://doi.org/10.1016/j.jbef.2017.12.004

[30] Eyal Peer, Laura Brandimarte, Sonam Samat, and Alessandro Acquisti. 2017. Beyond the Turk: Alternative Platforms for Crowdsourcing Behavioral Research. *Journal of Experimental Social Psychology* 70 (2017), 153–163. https://doi.org/10.1016/j.jesp.2017.01.006

[31] Mike Perkowitz and Oren Etzioni. 1998. Adaptive Web Sites: Automatically Synthesizing Web Pages. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence* (Madison, WI, USA) *(AAAI '98 / IAAI '98)*. American Association for Artificial Intelligence, USA, 727–732.

[32] Ruth Rosenholtz, Yuanzhen Li, Jonathan Mansfield, and Zhenlan Jin. 2005. Feature Congestion: A Measure of Display Clutter. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Portland, OR, USA) *(CHI '05)*. Association for Computing Machinery, New York, NY, USA, 761–770. https://doi.org/10.1145/1054972.1055078

[33] Nishant Sinha and Rezwana Karim. 2015. Responsive Designs in a Snap. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE '15)*. Association for Computing Machinery, New York, NY, USA, 544–554. https://doi.org/10.1145/2786805.2786808

[34] StatCounter. 2021. StatCounter Screen Resolution Stats Worldwide. https://gs.statcounter.com/screen-resolution-stats

[35] Kashyap Todi, Gilles Bailly, Luis A. Leiva, and Antti Oulasvirta. 2021. Adapting User Interfaces with Model-Based Reinforcement Learning. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3411764.3445497

[36] Kashyap Todi, Jussi Jokinen, Kris Luyten, and Antti Oulasvirta. 2019. Individualising Graphical Layouts with Predictive Visual Search Models. *ACM Transactions on Interactive Intelligent Systems* 10, 1, Article 9 (2019), 24 pages. https://doi.org/10.1145/3241381

[37] Alexandre N. Tuch, Sandra P. Roth, Kasper Hornbæk, Klaus Opwis, and Javier A. Bargas-Avila. 2012. Is Beautiful Really Usable? Toward Understanding the Relation between Usability, Aesthetics, and Affect in HCI. *Computers in Human Behavior* 28, 5 (2012), 1596–1607. https://doi.org/10.1016/j.chb.2012.03.024

[38] Jean Vanderdonckt. 2008. Model-Driven Engineering of User Interfaces: Promises, Successes, Failures, and Challenges. In *Proceedings of the 5th Annual Romanian Conference on Human–Computer Interaction* (Iaşi, Romania) *(ROCHI '08, Vol. 8)*. Matrix ROM, Bucharest, Romania, 32–41.

[39] Rares Vernica and Niranjan Damera Venkata. 2015. AERO: An Extensible Framework for Adaptive Web Layout Synthesis. In *Proceedings of the 2015 ACM Symposium on Document Engineering* (Lausanne, Switzerland) *(DocEng '15)*. Association for Computing Machinery, New York, NY, USA, 187–190. https://doi.org/10.1145/2682571.2797084

[40] W3C. 2016. CSSOM View Module W3C Working Draft, 17 March 2016. https://www.w3.org/TR/cssom-view-1/.

[41] W3C. 2018. CSS Flexible Box Layout Module Level 1 W3C Candidate Recommendation, 19 November 2018. https://www.w3.org/TR/css-flexbox-1/.

[42] W3C. 2018. Web Content Accessibility Guides (WCAG) 2.1 W3C Recommendation 05 June 2018. https://www.w3.org/TR/WCAG21/.

[43] W3C. 2020. CSS Grid Layout Module Level 1 W3C Candidate Recommendation Draft, 18 December 2020. https://www.w3.org/TR/css-grid-1/.

[44] 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* (Dunedin, New Zealand) *(CHINZ '12)*. Association for Computing Machinery, New York, NY, USA, 72–79. https://doi.org/10.1145/2379256.2379268

[45] Clemens Zeidler, Gerald Weber, Wolfgang Stuerzlinger, and Christof Lutteroth. 2017. Automatic Generation of User Interface Layouts for Alternative Screen Orientations. In *Human–Computer Interaction – INTERACT 2017*, Regina Bernhaupt, Girish Dalvi, Anirudha Joshi, Devanuj K. Balkrishan, Jacki O'Neill, and Marco Winckler (Eds.). Springer International, Cham, Switzerland, 13–35. https://doi.org/10.1007/978-3-319-67744-6_2

[46] Alexander Zlatkov. 2018. How JavaScript Works: Tracking Changes in the DOM Using MutationObserver. https://blog.sessionstack.com/how-javascript-works-tracking-changes-in-the-dom-using-mutationobserver-86adc7446401

Received February 2021; accepted April 2021