Pacific Graphics 2021
E. Eisemann, K. Singh, and F.-L Zhang
(Guest Editors)

*Volume 40 (2021), Number 7*

# Modeling Visual Containment for Web Page Layout Optimization

K. Kikuchi[1], M. Otani[2], K. Yamaguchi[2], and E. Simo-Serra[1]

[1]Waseda University, Japan [2]CyberAgent, Japan

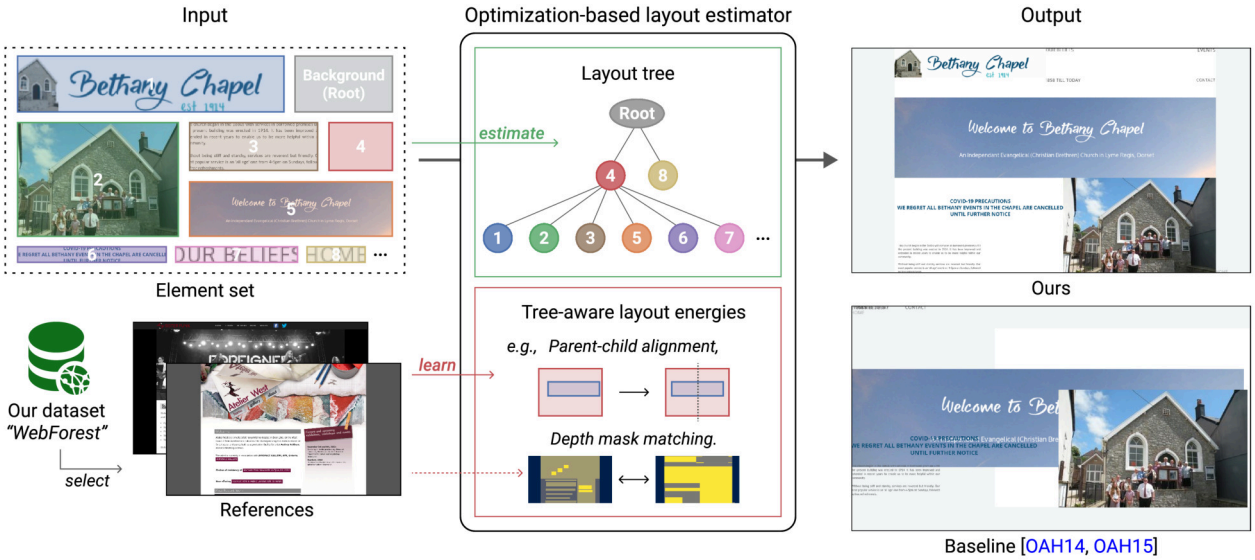



**Figure 1:** *Given a set of elements and reference designs, our method automatically generates a plausible web page layout. Our method first estimates a* layout tree *representing visual containment, and then optimizes the layout with learned tree-aware energies. While the existing method fails to visually organize the elements, our method is successful through hierarchical layout parameterization via layout tree.*

**Abstract**

*Web pages have become fundamental in conveying information for companies and individuals, yet designing web page layouts remains a challenging task for inexperienced individuals despite web builders and templates. Visual containment, in which elements are grouped together and placed inside container elements, is an efficient design strategy for organizing elements in a limited display, and is widely implemented in most web page designs. Yet, visual containment has not been explicitly addressed in the research on generating layouts from scratch, which may be due to the lack of hierarchical structure. In this work, we represent such visual containment as a layout tree, and formulate the layout design task as a hierarchical optimization problem. We first estimate the layout tree from a given a set of elements, which is then used to compute tree-aware energies corresponding to various desirable design properties such as alignment or spacing. Using an optimization approach also allows our method to naturally incorporate user intentions and create an interactive web design application. We obtain a dataset of diverse and popular real-world web designs to optimize and evaluate various aspects of our method. Experimental results show that our method generates better quality layouts compared to the baseline method.*

**CCS Concepts**

• ***Human-centered computing*** → *Interaction design process and methods;* • ***Applied computing*** → *Computer-aided design;*

## 1. Introduction

The World Wide Web has become one of the major backbones of modern society with nearly 200 million active websites [Net20]. This has led to web design and aesthetics to play a fundamental role in visual communication [Tho07]. The design of a website affects users' perception and behavior [FGO09], leading to an in-

creased investment in web design. This huge and steady demand of websites has promoted the development of web builders, *e.g.*, Dreamweaver, Wix.com, and Webflow, which have significantly lowered the entry bar to creating websites. However, web design is still difficult for non-technical users who lack knowledge on design principles and guidelines [Skl11, Wil15]. Given that the use of examples is common in practice [LSK*10], we focus on assisting the web design process, especially the web page layout, by utilizing reference designs.

Visual containment is an efficient design strategy for organizing elements in a limited display, and the *container* is implemented in many web frameworks as a basic layout element [Boo21, Web21]. Containers are used to group elements that play a similar role, *e.g.*, menu buttons or gallery images. Web pages that have elements with similar roles distributed across different container elements can be impractical and aesthetically unpleasing. Our work incorporates the concept of visual containment directly by using a tree hierarchy of elements which we denote as the *layout tree*.

Our approach consists of posing the task of graphic design layout as an optimization problem that determines the position and size of all elements that form the layout. We model the visual containment with a *layout tree* that represents it as parent-child relationships, which can be estimated from a set of elements. Generating layouts through optimization rather than inference, our approach enables flexible interactive design that follows users' intentions and expectations by changing optimization parameters and adding objective functions on demand. We build upon the previous works [OAH14, OAH15], which model important design properties such as alignment, symmetry, and spacing as energy terms to be minimized. In particular, we modify those terms to be tree-aware, and introduce new terms such as alignment to parent and matching to reference masks. We also employ the modern derivative-free algorithm [Han16], which greatly speeds up the optimization process. A high-level overview of our approach is shown in Fig. 1.

We collected a dataset of real-world web pages, which we call *WebForest*, in order to train and evaluate the methods. The data was collected by crawling popular websites based on traffic analysis. For each web page, we not only store the position, HTML text, and hierarchy of the elements, but also manipulate the visibility to capture an occlusion-free image of each element, making it unique among commonly used datasets.

We designed an experiment to enable automatic evaluation of layout generation with references, and the proposed method shows quantitatively and qualitatively better results than the baseline. The proposed method also performed better in perceptual evaluation by user voting. To evaluate the method in a more practical scenario, we developed an interactive design tool that incorporates our method, and obtained encouraging results through online experiments.

In summary, our key contributions are as follows:

- An optimization-based hierarchical layout model focused on visual containment.
- A method to compute a layout tree from a set of elements.
- A new dataset for studying the layout with visual containment problem.
- In-depth evaluation of the proposed approach comparing to the existing approaches [OAH14, OAH15].

## 2. Related Work

### 2.1. Graphic Layout Generation

Early work on layout generation has been seen in the application in on-screen articles [JLS*03], and more recently there has been research targeting layouts in various media such as comics [CCL12], single-page graphic design [OAH14, OAH15], magazine covers [YMX*16], scientific posters [QFY*19], mobile UIs [SWO*20], and web pages [PCLC16, DTSO20, LNDO20]. Pang *et al.* [PCLC16] propose a method for optimizing web design so that the user attention follows the path given by the designer. O'Donovan *et al.* [OAH14] tailor energy functions for single-page graphic designs such as advertisements, flyers, and posters. Their energy enforces several design guidelines, including alignment, symmetry, and white space. They later use a simplified energy model to build an interactive system [OAH15]. Dayama *et al.* [DTSO20] formulate the grid layout problem as mixed integer linear programming, and develop a wireframing tool with real-time layout suggestions.

Recent deep learning approaches [LXZ*19] have been shown to approximate data distributions well without domain-specific knowledge with sufficient training data, but it is difficult to reflect the user's intention under interaction. Lee *et al.* [LJE*20] and Li *et al.* [LYZ*20] learn user intents with conditioning on user constraints, even though their methods do not guarantee the constraints are satisfied. Zheng *et al.* [ZQCL19] report a method for generating a variety of magazine layouts considering the content of images and text. Also, Li *et al.* [LAZ*20] concurrently performed a layout completion task using a tree-based transformer.

Another important line of research is the retargeting of web pages for responsive design. AERO [VV15], given contents, searches for the suitable one from the pre-defined HTML templates, including those for PC and mobile screens. Sinha and Karim [SK15] formalize responsive design as a constraint repair problem for the hierarchical representation of web pages, and built a recommendation tool called *DECOR*. More recently, C-RWD [LZS*21] automatically converts existing web designs into those suitable for different screen widths based on integer programming and grid layout. These studies require a completed design or hierarchy, and cannot generate a design from scratch.

In this work, we consider visual containment in layout generation. Although a limited form of containment appears in the literature (*e.g.*, text over image in [ZQCL19] or relation inference in [LJE*20]), we go further and explicitly model the container-containee relationships among visual elements, and propose a layout tree estimation approach. We adapt energy models of the existing work [OAH14, OAH15] to explicitly utilize visual containment and tree structures. Our results show that hierarchical modeling is essential in effectively generating complex real-world web designs.

### 2.2. Exemplar-based GUI Design

Web design often leverages existing designs to produce new pages. For efficient navigation of inspiring designs, Ritchie *et al.* [RKK11] build a style-based web design navigator named *d.tour*. Gallery D.C. [CFX*19] accumulates millions of web designs to build a GUI component gallery.

For data-driven GUI design, d.mix [HWCK07] samples Web APIs on various external sites to build a personalized mashup view. Pix2code [Bel18] generates a code that can be rendered to reproduce a given GUI screenshot. FaceOff is a data-driven system that combines a set of templates to create a single harmonious GUI template [ZHM19]. Kumar *et al*. [KTAK11] address the DOM node matching problem for style transfer from a reference web page. Kumar *et al*. [KST*13] later built a scalable data collection framework for web design, in order to provide insights to designers.

Our approach shares the spirit of exemplar-based design in that we take reference designs to construct positional energy terms and train a specialized layout model with them.

### 2.3. Interactive Design

Providing useful feedback to the designer is another important task. Todi *et al*. [TWO16] propose an interactive layout optimization system that starts from a sketch. Swearngin *et al*. [SWO*20] design constraints at a high level of abstraction, and propose a system that supports rapid exploration of UI alternatives through a constraint solver. GUIComp [LKH*20] is an authoring tool that gives various real-time feedback to the current design. Aalto Interface Metrics [ODPK*18] is an online tool that quantitatively evaluates web designs, and the designers can easily find shortcomings in their designs. Zhao *et al*. [ZCL18] model the compatibility of font against surrounding contexts in the web design.

In this work, we built an interactive tool similar to [OAH15] for user study and incorporated our model as a layout recommender.

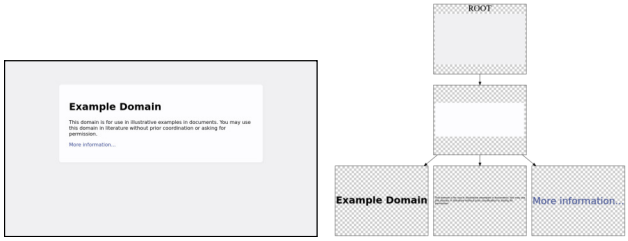
## 3. Dataset

In order to train our model and perform in-depth evaluation, we have constructed a large-scale dataset of real-world web pages with a focus on visual containment that we call the *WebForest dataset*. They have been obtained by crawling a wide variety of web pages based on the popularity scores. Having been designed for the modelling of visual containment, unlike existing datasets, our dataset provides images and metadata for each element, the layout trees, read-orders, and semantic labels for each web page.

We compare existing datasets with ours in Table 1. For datasets such as Rico [DHF*17], where only the whole image is provided, it may be possible to obtain images for each element as we do, through image inpainting techniques. They will, however, likely introduce artifacts that damage the perceptual quality and become difficult to evaluate the layout quality by user voting.

### 3.1. Web Page Crawling

We build a dataset consisting of diverse real-world web pages by using a traffic analysis tool. In particular, we use the API provided by Alexa Internet [Int21] to pick the top pages of up to 20 popular websites in each of the 1,848 subcategories including Top/Science/Academic_Departments, Top/Regional/North_America/Health, and Top/Business/Industrial_Goods_and_Services/Bearings, to name a few, and obtain 26,409 pages. We crawl the HTML text for each page, and the position, size, and rendered image for each element.




**(a)** *Screenshot* **(b)** *Layout tree*

**Figure 2:** *An example web page from* `https://example.com/`*. The parent-child relationships in the layout tree (b) represent visual containment of the elements on the rendered screen (a).*

We filter out inappropriate pages such as 404 errors and cases where reconstruction of a page screenshot from crawled data yields a higher error. We also filter out too simple pages with less than five elements, and too complex pages with more than 50 elements, such as those having many decorative elements. After filtering, we obtained 4,521 pages in total, and split them into 4,122 for training and validation and 399 for testing, ensuring that the distributions of the number of elements in a page are roughly the same.

Obtaining rendered images of all elements in a page without occlusion is not supported by modern browsers. In order to overcome this limitation, we rely on the Selenium WebDriver [Sel21] which allows automating visibility control and screenshot capturing. In particular, we set the browser's default background color to transparent, and use JavaScript to make all elements except the target one transparent. We can then capture a screenshot and crop the non-transparent region to obtain the isolated element. Note that despite the development efforts, about 60% of the total pages fail because the design changes while taking a screenshot of each element, which is due to the dynamics caused by JavaScript, such as carousels. This may be solved by disabling JavaScript functions related to dynamics, such as overriding the setInterval function with a function that does nothing, but we leave it for future work.

### 3.2. Layout Tree

The layout tree represents the visual containment of the elements in a web page, *i.e*., the children of a parent element will all be visually contained inside the parent. Web pages are designed using a Document Object Model (DOM) tree which seems like could play the role of a layout tree, however, the DOM tree has many spurious nodes and many different node types play similar roles, making it unsuited for modeling visual containment. For this purpose, we convert a DOM tree into a tractable layout tree, similar to the Bento algorithm [KTAK11]. First, we keep only those elements where any of the descendants is visible. We then remove elements that are too small or completely overlapped, and merge pairs of overlapping elements if the Intersection over Union (IoU) is large or they are painted in exactly the same color. If all the descendant elements are inline text, we also merge them into the ancestor element. We then rearrange the tree so that the parent-child relationships correspond to the visual containment on the page. Finally, we remove the functional intermediate nodes so that all elements have visual components. An example page and its layout tree are shown in Fig. 2.

**Table 1:** *Comparison with public datasets used in graphic layout research. We note that the existing datasets are not suitable when considering visual containment as they do not provide hierarchy or appropriate images which could perceptually degrade the rearranged design.*

| Dataset | # Samples | Annotation |
|---|---|---|
| Rico [DHF*17] | 72.2 k | Screenshots, view hierarchies, semantic labels, and layouts. *No images for each element.* |
| Magazine [ZQCL19] | 3.9 k | Cropped and partially masked images, keywords, and layouts. *Not all images being pixel-perfect and no hierarchical structures.* |
| WebForest (Ours) | 4.5 k | Image and position for all elements, layout trees, ordering, and semantic labels |

**Table 2:** *Overview of the energy terms comprising our model, which can be roughly grouped into 7 different categories.*

| Category (Num.) | Description |
|---|---|
| alignment (15) | coarse align. of sibling/parent-child, fine align., and align. consistency. |
| scale (14) | size mean enlargement, size variance reduction, and matching size order to metadata. |
| spacing (7) | element spacing, white space, edge margin. |
| symmetry (4) | vert. & horiz. symmetry and asymmetry. |
| position (2) | matching to reference masks. |
| overlap (1) | avoiding element overlap. |
| ordering (1) | matching read-order to metadata. |

### 3.3. Metadata

We extract and simplify metadata from web pages to use them as simple hints in our later experiments. We first specify an element's role in the web page. Since HTML tags do not often represent the actual element role, we assign the semantic label to an element according to its properties by using heuristic rules. The label set used in our work contains six labels: text, button, input, graphic, image, and container. Detailed labeling rules can be found in the supplemental material.

We then specify the positional order and the rough size of the elements, named read-order and importance, respectively, which are necessary to make the generated layout obey the user's expectations. We compute these metadata from the position and size of the elements in the ground-truth layout. We assign the read-order to the elements in top-left to bottom-right order. The values of importance used in our work contains five types: 0 (x-small), 1 (small), 2 (medium), 3 (large), and 4 (x-large). We defined the importance metadata separately for text and non-text elements, and determined their values by performing k-means clustering, using the CSS font size attribute for text elements and area for non-text elements. Finally, since the text element can have multiple lines, we approximated its number of lines from the height and font size attribute.

## 4. Approach

### 4.1. Overview

We define a layout problem as an optimization over *layout parameters* $\mathbf{X}$ that specify the position and size of each element. In particular, we consider a *layout tree* $\mathbf{T}$ that represents the visual containment among elements, and assume we are given the *configuration* $C$ consisting of semantic labels and read-order of elements and so on. The objective is to solve for the best layout parameters $\mathbf{X}$.

We formulate the layout optimization into a two-step process. First we solve for the best layout tree from a set of elements, then solve for the optimal layout:

$$\arg\min_{\mathbf{X}} \mathsf{E}_X(\mathbf{X}, C, \arg\min_{\mathbf{T}} \mathsf{E}_T) \,, \tag{1}$$

where $\mathsf{E}_X$ and $\mathsf{E}_T$ are energy functions for the design layout and the layout tree, respectively.

The energy term $\mathsf{E}_X$ is composed of the weighted sum of 44 different components corresponding to various aspects of graphic design such as alignment, or symmetry and can be written by:

$$\mathsf{E}_X(\mathbf{X}, C, \mathbf{T}) = \sum_k w_k \mathsf{E}_k(\mathbf{X}, C, \mathbf{T}) \,, \tag{2}$$

where $w_k$ is the $k$-th weight and $\mathsf{E}_k$ the $k$-th energy function. An overview of our energy terms is summarized in Table 2. We learn the weights $w_k$ from a small set of reference designs.

In the rest of this section, we first describe the layout parameterization defined by a layout tree (§4.2) and an optimization-based method for estimating the layout tree (§4.3). We then explain the tree-aware layout energy terms derived from various design heuristics (§4.4). Lastly, we describe a method for solving Eq. (1) by stochastic optimization (§4.5) and learning the energy weights in Eq. (2) by inverse optimization (§4.6).

### 4.2. Layout Parameterization

For each web page we consider a fixed number $N$ of elements and we model each element with three parameters corresponding to horizontal position, vertical position, and element height. Unlike previous work [OAH14], parameters are not defined to be absolute, but relative to the parent container element, *e.g.*, a horizontal position of 0 corresponds to the left-most side of the parent container, while a value of 1 corresponds to the right-most side of the parent container. The height is also normalized such that a value of 1 corresponds to the parent height. We directly enforce the visual containment in our parameterization by restricting the position and height inside the parent container. The parameters of the entire layout $\mathbf{X}$ consist of the parameters of all the elements concatenated together, thus $\dim(\mathbf{X}) = 3N$. Note that for simplicity, we assume that the aspect ratio is provided for each element. Alternative elements can be handled with additional indicator parameters [OAH14].

**ALGORITHM 1:** Tree optimization using simulated annealing

**Input:** tree energy model $\mathsf{E}_{\mathbf{T}}$, initial temperature $t_{\text{init}}$, final temperature $t_{\text{fin}}$, # iterations $N_{\text{iter}}$, number to keep $N_{\text{keep}}$
**Output:** optimized feasible tree $\mathbf{T}^*$

```
t ← t_init
Δt ← (t_init − t_fin)/N_iter
T_curr ← create_flattened_tree()
H ← {T_curr}
for i = 1 to N_iter do
    T_next ← create_neighbor_tree(T_curr)
    if E_T(T_next) ≤ E_T(T_curr) then
        p ← 1
    else
        p ← exp((E_T(T_curr) − E_T(T_next))/t)
    end
    if random() < p then
        T_curr ← T_next
        H ← H ∪ {T_curr}
        if N_keep < |H| then
            H ← discard_worst_tree(E_T, H)
        end
    end
    t ← t − Δt
end
while 0 < |H| do
    T* ← get_best_tree(E_T, H)
    if not violate_any_bounds(T*) then
        return T*
    else
        H ← H \ {T*}
    end
end
```

### 4.3. Layout Tree Estimation

We define layout tree as a rooted tree $\mathbf{T}$, where each node in the layout tree corresponds to an element in the layout. The layout tree must be a valid tree and its parent-child relationship must be plausible. We consider layout tree estimation as the optimization problem to search for a tree that minimizes tree energy model $\mathsf{E}_{\mathbf{T}}$ as follows:

$$\mathsf{E}_{\mathbf{T}}(\mathbf{T}) = \alpha_{\text{anc}}\mathsf{E}_{\text{anc}} + \alpha_{\text{sib}}\mathsf{E}_{\text{sib}} + \alpha_{\text{leaf}}\mathsf{E}_{\text{leaf}} \tag{3}$$

where $\mathsf{E}_{\text{anc}}$, $\mathsf{E}_{\text{sib}}$, $\mathsf{E}_{\text{leaf}}$ are the anchor, sibling, and leaf energies, respectively, and $\alpha_{\text{anc}}$, $\alpha_{\text{sib}}$, and $\alpha_{\text{leaf}}$ are weights learnt from data. In our approach, we restrict the search space to only valid trees and evaluate the plausibility of the layout tree by the energy model.

#### 4.3.1. Tree Energy

For efficient optimization, we use a matrix representation of the layout tree $\mathbf{T} = \{\mathbf{A}, \mathbf{S}, \mathbf{L}\}$. In particular we define the ancestor matrix $\mathbf{A} \in \{0,1\}^{N\times N}$ where $\mathbf{A}_{i,j}$ is set to 1 if the $i$-th element is an ancestor of the $j$-th element in the tree or 0 otherwise. Similarly, we define the sibling matrix $\mathbf{S} \in \{0,1\}^{N\times N}$ where $\mathbf{S}_{i,j}$ is 1 if the $i$-th and $j$-th elements are siblings, 0 otherwise. Finally, we define the leaf vector $\mathbf{L} \in \{0,1\}^{N}$ where $\mathbf{L}_i$ is 1 if the $i$-th element is a leaf or 0 otherwise.

We calculate the probabilities of an element being a leaf, a pair of elements being siblings, and a pair of elements being parent-child by using binary classifiers with random forests [Bre01] that take element features as an input. For each element we compute a 14-dimensional feature vector that uses simple features such as the aspect ratio and mean RGB values, and a complete list is provided in the supplemental material. In the case of predicting whether or not elements are siblings or have a parent-child relationship, we simply concatenate the features as the input for the random forest. We train three random forest models using the training and validation split of our dataset, with the ground-truth properties as positive and all other possibilities as negative samples. We calibrate the classifiers to treat their output as a probability [NMC05]. We report that F1 scores for the binary classification of the tree properties are 0.334, 0.580, and 0.966 for ancestor, sibling, and leaf, respectively.

**Table 3:** *Quantitative evaluation of the tree estimation approach. We evaluate using the F1 scores of ancestors, siblings, and leaves. The estimated trees get better F1 scores than the flattened trees.*

| Layout tree | $F_{\text{anc}} \uparrow$ | $F_{\text{sib}} \uparrow$ | $F_{\text{leaf}} \uparrow$ |
|---|---|---|---|
| Flattened | 0.000 | 0.574 | 0.915 |
| Estimated | **0.426** | **0.622** | **0.963** |

We then obtain the probably matrix of elements being siblings $\bar{\mathbf{S}} \in [0,1]^{N\times N}$, the probably matrix of elements being parent-child $\bar{\mathbf{A}} \in [0,1]^{N\times N}$, and the probably vector of elements being leaves $\bar{\mathbf{L}} \in [0,1]^{N}$. These are defined analogously to $\mathbf{S}$, $\mathbf{A}$, and $\mathbf{L}$, with the only difference being that they are probability values instead of binary 0 or 1 values. We consider energies that increase the average of the estimated probabilities for each tree property, which can be written as:

$$\begin{aligned} \mathsf{E}_{\text{anc}}(\mathbf{A},\bar{\mathbf{A}}) = \mathsf{E}_D(\mathbf{A},\bar{\mathbf{A}}) \qquad \mathsf{E}_{\text{sib}}(\mathbf{S},\bar{\mathbf{S}}) = \mathsf{E}_D(\mathbf{S},\bar{\mathbf{S}}) \\ \mathsf{E}_{\text{leaf}}(\mathbf{L},\bar{\mathbf{L}}) = 2\mathsf{E}_D(\mathbf{L},\bar{\mathbf{L}}) - 1\,, \end{aligned} \tag{4}$$

where

$$\mathsf{E}_D(\mathbf{X},\mathbf{Y}) = 1 - \frac{1}{2N}\left(\langle \mathbf{X},\mathbf{Y}\rangle + \langle (1-\mathbf{X}),(1-\mathbf{Y})\rangle\right)\,, \tag{5}$$

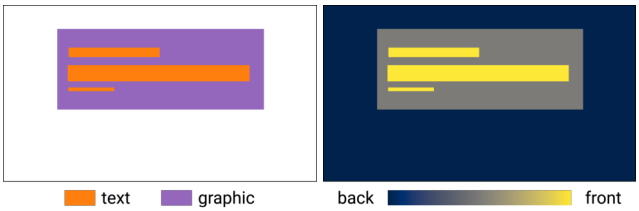
with $\langle \mathbf{X},\mathbf{Y}\rangle = \sum_{i,j}\mathbf{X}_{ij}\mathbf{Y}_{ij}$ being the inner product of matrices or vectors.

#### 4.3.2. Tree Optimization

We write the tree optimization problem as:

$$\begin{aligned} \underset{\mathbf{A},\mathbf{S},\mathbf{L}}{\arg\min} \quad & \alpha_{\text{anc}}\mathsf{E}_{\text{anc}}(\mathbf{A},\bar{\mathbf{A}}) + \alpha_{\text{sib}}\mathsf{E}_{\text{sib}}(\mathbf{S},\bar{\mathbf{S}}) + \alpha_{\text{leaf}}\mathsf{E}_{\text{leaf}}(\mathbf{L},\bar{\mathbf{L}}) \\ \text{subject to} \quad & \{\mathbf{A},\mathbf{S},\mathbf{L}\} \text{ being a valid tree.} \end{aligned} \tag{6}$$

Searching for the layout tree that minimizes Eq. (6) is a difficult optimization problem due to its huge search space. The tree optimization problem has been actively studied in the discipline of phylogenetics and is known to be NP-hard [NSvHM14]. We optimize with simulated annealing [KGV83], which is a stochastic optimization method in which a temperature parameter controls how much change is allowed at each iteration. The initial tree is a flattened tree in which each element is a child of the root element, and thus is a valid tree. We use a strategy of linearly decreasing the temperature, which allows the algorithm to accept changes to higher energies with high probability in the early stage, and thus avoid falling into local solutions. At the end of the optimization, the algorithm rarely




**(a)** *Label mask* **(b)** *Depth mask*

**Figure 3:** *Example of label and depth masks for the same web page as in Fig. 2.*

accepts changes to higher energies. We randomly use two proposal strategies to modify the current tree at each iteration: swapping the two elements, and making one element a child of another. We note that both these operations ensure that the current solution is a valid tree and there is no additional need to enforce any other constraints.

We perform optimization for 100,000 iterations, and it takes less than 10 seconds on a machine with a Intel(R) Xeon(R) Platinum 8260 CPU (2.40GHz). The weights $\alpha_{anc}$, $\alpha_{sib}$, and $\alpha_{leaf}$ of Eq. (6) are tuned with the training and validation split using Optuna [ASY*19]. We summarize the entire algorithm in Algorithm 1. We compare the optimized tree with the initial flat tree in Table 3, where we can see the optimization significantly outperforms the naive flattening strategy and mostly improves the independent prediction results. For details on the F1 score calculation, please refer to the supplemental material.

### 4.4. Layout Energy Model

We build our model upon the work of [OAH14], modify most of the energy terms to make use of the layout tree, in addition to add new terms that are suited to the visual containment model. We note that each of the terms has an independent non-negative weight that is learned from reference designs, thus it is common to define energy terms that are contrary to each other and let the model learn which term is important for a given web page. Below we give a high-level overview of the energy terms, for the details, please refer to the supplemental material.

#### 4.4.1. Label and Depth Mask

We extend on previous work that only consider text and graphic labels, and assign each element as being one of six labels as explained in §3.3. We use these labels to construct a label mask, where for each pixel in the rendered screen, we compute whether or not a pixel contains any element of a label for each label. In a similar approach, we also construct a depth mask using the levels of the layout tree like z-values. For each pixel in the mask, we use the the maximum level of the corresponding elements and normalize it as the root element being 0 and the leaf elements being 1. These mask representations are used in the energy terms described in the following subsections, and an example is shown in Fig. 3.

#### 4.4.2. Alignment

In general, alignment of elements leads to aesthetically pleasing layouts. Instead of limiting the alignment to arbitrary element pairs, we focus on exploiting the layout tree and look at alignment between siblings and parent-children. We consider six possible alignment types: Left, X-center, Right, Top, Y-center, and Bottom. We first define a coarse alignment terms that encourages adjacent sibling elements to be aligned to each other, and one that encourages the alignment of parents and their children for each of the six alignment types. This gives a total of 12 different terms. We additionally add fine alignment terms for both horizontal and vertical alignment that penalize misalignment if nearly aligned in order to encourage full alignment. Finally, we encourage elements to share the same type of alignment with a group alignment term. This term penalizes the presence of many different alignment types and thus encourages a more uniformed alignment for the design layout.

#### 4.4.3. Symmetry

Symmetry can play an important role in graphic design. For this purpose, we implement a simple check for both symmetry and asymmetry, depending on how the terms are weighted, our mehtod can optimize towards either symmetry or asymmetry. We implement the terms by using the depth mask, which is flipped either horizontally or vertically, with the energy term measuring how similar the flipped version is to the original version. Thus, this term is a global term that considers the symmetry of the entire layout.

#### 4.4.4. Spacing

The amount of spacing is very important on the visual impact of a design, and we evaluate it using seven terms. The first two terms guide the elements to increase the proportion of the white space area within the entire canvas or parent element. The another term promotes a larger average distance between elements, and another term encourages the elements to be spread throughout the page. We also add two terms that gather the elements to the center, and implement it by promoting larger margins between the outermost element and the edge of the entire canvas or parent element. The last term facilitates uniform vertical spacing of adjacent text elements.

#### 4.4.5. Scale

In general, an element should be large enough to be seen, but not too large to be aesthetically unpleasant. Our model has per-label energy terms that encourage larger sizes of content elements, and per-label energy terms that suppress the variance in scale of elements with the same label We also have two terms that encourage the scale of the elements to correlate with the importance metadata: one for text elements, and one for non-text elements. This is usually called *emphasis* in the literature [OAH14, SWO*20].

#### 4.4.6. Position

Given that web pages can have very complex layouts, it is difficult to reflect the positional trends in the reference designs with simple statistics for each label as in previous work [OAH14]. Instead, we opt to represent the position of elements as a mask and evaluate the consistency of the masks between the reference and current design. With the idea that the mask should cover the reference mask with minimal over or under coverage, we borrow the concept of the F1 score to design two matching terms: one for the label masks, and one for the depth masks.

#### 4.4.7. Overlap and Ordering

Under the assumption that sibling elements should not overlap each other, we design an energy term to penalize the amount of overlap between siblings. Additionally, to achieve the reading order of the elements as intended by the user, we encourage the elements to be arranged according to the read-order metadata.

### 4.5. Optimization

While the previous work [OAH14] relies on simulated annealing to optimize a layout by changing elements basically one-by-one, we found that while this only works for a limited number of elements, and it performs very poorly when dealing with large amounts of elements in the web pages of our dataset. We opt to use the CMA-ES [Han16] algorithm, which is an efficient derivative-free optimization algorithm. With CMA-ES, the candidate solutions are modeled as a multivariate normal distribution where the covariance matrix represents the pairwise dependencies between variables. The covariance matrix is updated progressively based on the values evaluated by the objective function for sampled candidates. We set the maximum number of iterations to 1,000, and run eight optimizers in parallel and select the minimum. We measured the time taken to optimize the web pages in our test set, and the median time was 85.5 seconds on a machine with a Intel(R) Xeon(R) Platinum 8260 CPU (2.40GHz).

### 4.6. Learning Model Parameters

Our model consists of 44 energy terms, and each term has a weight parameter that can be learned. Following the existing studies [OAH14,OAH15], we learn the weight parameters $\mathbf{w}$ by Nonlinear Inverse Optimization (NIO) [LHP05]. We search for weights $\mathbf{w}$ in Eq. (2) that minimize the following equation $G$, given a set of layout problems $(\bar{\mathbf{X}},\bar{C},\bar{\mathbf{T}}) \in \mathcal{X}$ as reference:

$$G(\mathbf{w}) = \frac{1}{|\mathcal{X}|} \sum_{(\bar{\mathbf{X}},\bar{C},\bar{\mathbf{T}})\in\mathcal{X}} \left( \sum_k w_k E_k(\bar{\mathbf{X}},\bar{C},\bar{\mathbf{T}}) - \min_{\mathbf{X}} \sum_k w_k E_k(\mathbf{X},\bar{C},\bar{\mathbf{T}}) \right) \tag{7}$$

We note that in general, we use the estimated tree $\bar{\mathbf{T}} = \arg\min_{\mathbf{T}} \mathsf{E}_T$ unless otherwise specified.

We then minimize $G$ using gradient descent with line search. With the optimal layout $\mathbf{X}^\star$ obtained by the optimization explained in the previous section, the gradient of $G$ can be approximated as:

$$\frac{dG(\mathbf{w})}{d\mathbf{w}} \approx \frac{1}{|\mathcal{X}|} \sum_{(\bar{\mathbf{X}},\bar{C},\bar{\mathbf{T}})\in\mathcal{X}} \left( \frac{\partial E_X(\bar{\mathbf{X}},\bar{C},\bar{\mathbf{T}})}{\partial \mathbf{w}} - \frac{\partial E_X(\mathbf{X}^\star,\bar{C},\bar{\mathbf{T}})}{\partial \mathbf{w}} \right) \tag{8}$$

We reparameterize $\mathbf{w}_k = \exp(\beta_k)$ to force the weight parameters to be non-negative, and the actual optimization is done for $\beta_k$.

## 5. Automatic Evaluation of Layout Generation

To validate our method, we design an automatic evaluation experiment by simulating how a user creates designs with our layout model. We summarize below the simulated user behavior.

1. *Ideation.* The user ideates a desired design in his or her mind. We call that design as *target design*, and we select a real web page from our dataset and consider it as target design.
2. *Reference search.* The user finds reference designs that are similar to the design in his or her mind. We imitate this behavior by retrieving similar designs to the target design and use them as references $\mathcal{X}$ when optimizing the weight parameters $\mathbf{w}$.
3. *Generation.* The user sets the desirable metadata, and our method automatically generates a design. In the experiments, we compute the metadata $C$ from the ground-truth target design.
4. *Evaluation.* The user evaluates the generated design to see if it is good enough. We evaluate the generated design by measuring the difference with the target design using several metrics.

**Table 4:** *Reconstructive correctness of automatic layout optimization. The correctness is computed based on IoU $d_{\text{IoU}}$, position error $d_{\text{pos}}$, and scale error $d_{\text{scale}}$. The layouts optimized by our model outperform the baseline layouts in all the metrics.*

| Method | $d_{\text{IoU}} \uparrow$ | $d_{\text{pos}} \downarrow$ | $d_{\text{scale}} \downarrow$ |
|---|---|---|---|
| LLSPGD | 0.080 | 0.472 | 2.384 |
| Ours | **0.091** | **0.448** | **2.152** |
| Ours (oracle) | 0.330 | 0.235 | 1.622 |

### 5.1. Reference Search

We use the web pages in the test set of our dataset for the target design and the pages in the train-validation set for a pool for searching references. We train an autoencoder on layouts and use the bottleneck feature of the autoencoder to compute the similarity between layouts, as in the design search used in the literature [DHF*17,LCS*18]. Our encoder, which consists of seven convolutional layers, takes a concatenation of a label mask and a depth mask as input. To retrieve references, we sort the candidate designs by Euclidean distance of the bottleneck features, and preferentially select those with similar label types that appear in the target design.

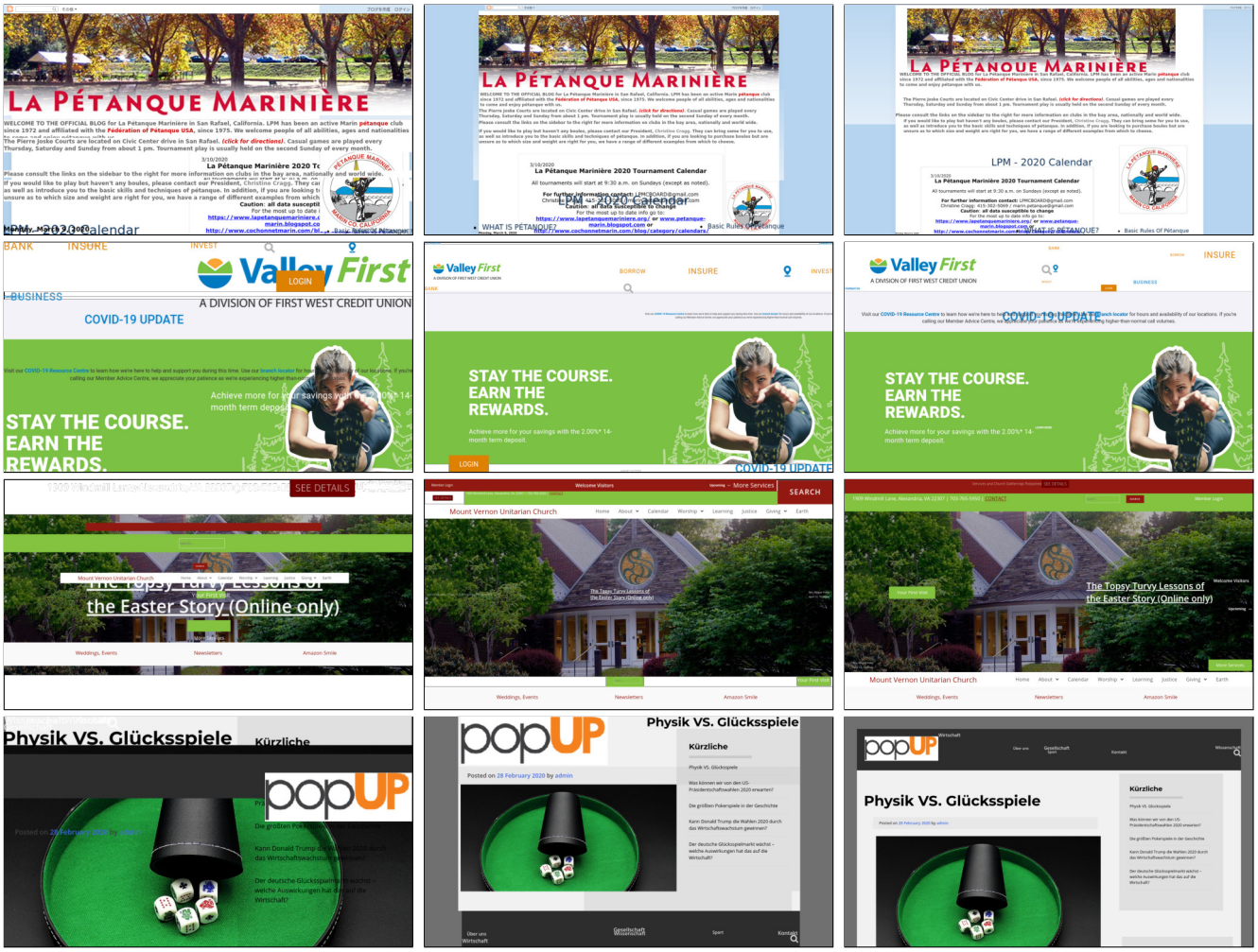
### 5.2. Evaluation Metrics

We evaluate generated layouts with the reconstructive correctness metrics: IoU $d_{\text{IoU}}$, position error $d_{\text{pos}}$, and scale error $d_{\text{scale}}$. IoU is computed with the intersection over union of the elements between the generated layout and target layout, position and scale metrics are the average errors normalized by the page size. For full details, please refer to the supplemental material. We evaluated all pages in the test set using the above metrics and report the mean values.

### 5.3. Comparison with Existing Approach

We compare our layout model with the approach of [OAH14] while applying the simplification and speed-up techniques used in [OAH15, O'D15], which we denote as LLSPGD. Other existing methods are difficult to compare fairly with ours because they have different problem settings, for example, the method in [DTSO20] assumes no overlap and uses a single layout model with no training capability. We implement LLSPGD using layout parameterization with a flat tree, and use the same optimization and learning methods and reference designs to make the comparison as fair as possible.

We show the generated results by ours and LLSPGD in Fig. 4,



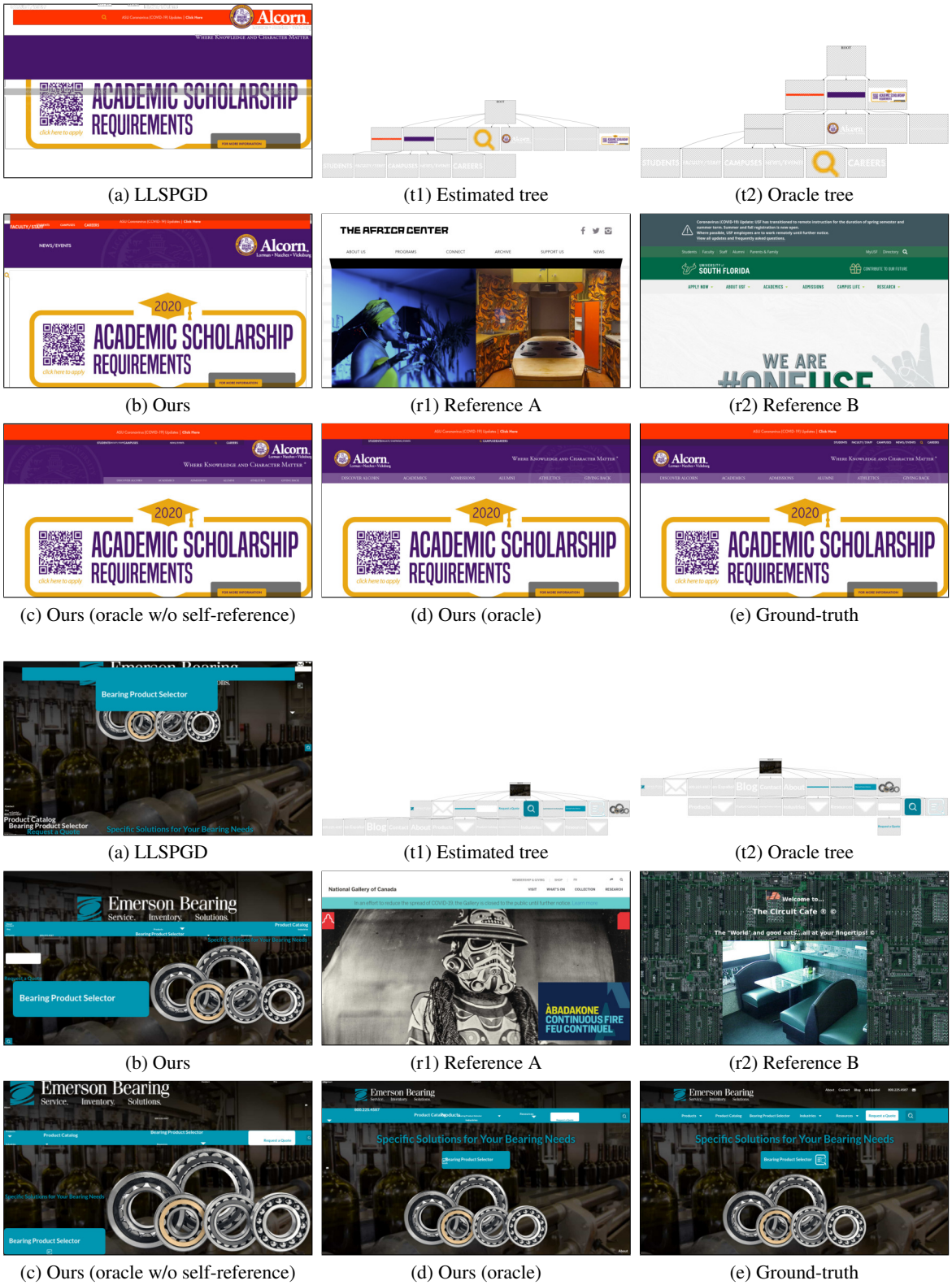

LLSPGD | Ours | Ours (oracle)

**Figure 4:** *Comparison with the baseline LLSPGD method [OAH14, OAH15]. LLSPGD uses the mean and variance of element positions per label to model positional trends, which can capture simple cases but not complex ones. On the other hand, our method successfully captures complex positional trends through the mask representations. Our method is also successful in organizing elements by modeling visual containment. Our method in the last row is a failure case of element ordering due to a critical tree estimation error (the gray container should be placed at the top as a header).*

and Fig. 5 revealing more detailed settings. We observe that the layouts of LLSPGD often suffer from inadequate overlaps of elements. Although LLSPGD uses an overlap avoidance term, it is designed for simple layouts with little or no overlap, whereas the web page layouts often have overlap. Our method, on the other hand, avoids this issue by computing only the overlap penalty between sibling elements. The examples demonstrate the importance of visual containment especially for designing layouts with many elements. Quantitative results are shown in Table 4, where we can see that the layouts optimized by our method reconstruct the target layouts better than those by the baseline in all the metrics. Ours (oracle) is the result considered to be the upper bound of our method, which uses ground-truth for layout trees and references, and shows that using more accurate trees and references can lead to a significant improvement. Further ablation study reveals how the use of the layout tree and our energy model contributes to the performance, see the supplemental material for details. We note, however, that given the multi-modal nature of the problem, the metrics based on a single ground-truth may not be adequate, and we complement them with a user study below.

**Table 5:** *User voting result for the optimized layout. The layout optimized by our method received a larger number of votes.*

| | # Votes | |
|---|---|---|
| Method | Quality | Similarity |
| LLSPGD | 124 | 152 |
| Ours | **275** | **247** |




(a) LLSPGD (t1) Estimated tree (t2) Oracle tree

(b) Ours (r1) Reference A (r2) Reference B

(c) Ours (oracle w/o self-reference) (d) Ours (oracle) (e) Ground-truth

(a) LLSPGD (t1) Estimated tree (t2) Oracle tree

(b) Ours (r1) Reference A (r2) Reference B

(c) Ours (oracle w/o self-reference) (d) Ours (oracle) (e) Ground-truth

**Figure 5:** *Qualitative comparison with LLSPGD. We use the same two reference designs (r1 and r2) for both LLSPGD and our method. Our method (b) produces better results than LLSPGD (a). The failure of our method (d) using the oracle tree and self-reference to reconstruct fine details may be due to the use of low resolution masks to reduce computational cost.*

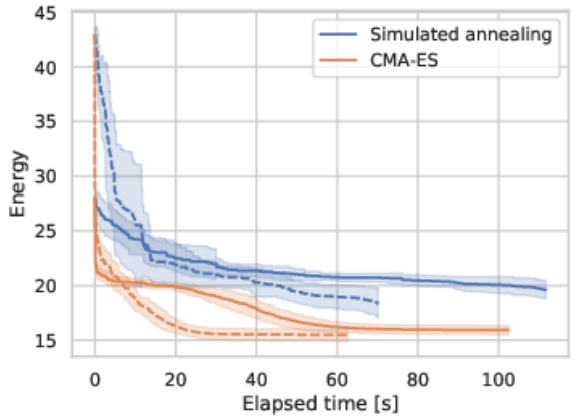



**Figure 6:** *Comparison of optimization methods. We measured the elapsed time and energy values for two layout problems (shown by line style). We performed 10 trials for each condition, and show the mean values by line and the standard deviations by range.*

We conducted a pairwise comparison based on user perception using Amazon Mechanical Turk (MTurk). The MTurk workers are displayed with a pair of designs by ours and the baseline and select one of the designs according to the question. The questions are about quality - *"Which one is better?"*, or similarity - *"Which one is more similar to the references?"*. The displayed positions of the designs change randomly, and only when the question is about similarity, the reference designs are displayed together. We issued 798 tasks (two questions for 399 configurations in the test split), paid ¢2 for a vote, and 43 workers voted (the number of votes per worker varied from 1 to 104). Table 5 summarizes the results. The layout optimized by our method received a larger number of votes. Using the Pearson's chi-square test, we found a significant difference in the number of votes for both questions about quality ($p$ = 4e-14) and similarity ($p$ = 2e-06).

### 5.4. Effects of Optimization Algorithms

Our method employs CMA-ES instead of simulated annealing used in the previous work. Fig. 6 shows the benefits of CMA-ES compared to simulated annealing in layout optimization. The comparison demonstrates that CMA-ES achieves better convergence speed and obtains a better minima than simulated annealing, which samples only one candidate solution from the designed proposal distribution for each iteration. CMA-ES is more efficient because it samples multiple candidate solutions from the normal distribution per iteration. Exploiting multiple candidates further allows us to find better solutions in shorter time.

## 6. Interactive Evaluation via Online User Study

We next compare our method with the baseline LLSPGD method in an interactive scenario. For this purpose, we developed a web-based design tool and conducted an online user study using Amazon Mechanical Turk (MTurk). The tool interface can be seen in Fig. 7.

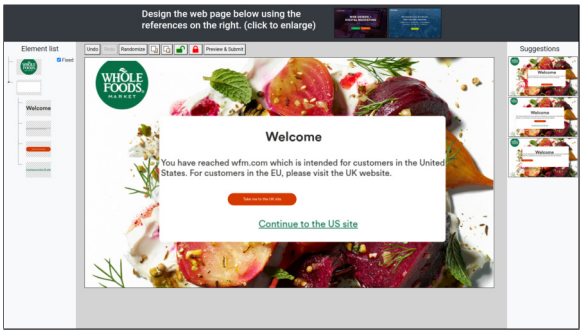



**Figure 7:** *Our interactive design tool. The right panel shows layout suggestions based on the current canvas state, and the left panel shows the current layout tree.*

### 6.1. Interactive Design Tool

We developed a tool for interactive experiments. Referring to the previous work [OAH15], we implemented the following features:

- *Layout editing.* Moving and scaling of elements, and manipulation of their overlapping order.
- *Tree editing.* The layout tree is shown as a tree view, and is synchronized with the current layout. The user can edit the tree by manipulating the elements on either the tree view or canvas.
- *Layout suggestions.* Using the current state, the stochastic optimizers with the pre-trained model run in parallel, and the resulting layouts are displayed as suggestions. The suggestions can be zoomed in by mouse over and applied to the canvas by click.

We also added some features to control the suggestions implicitly or explicitly to follow the user's expectations and intentions.

- *Element fixing.* The user can fix either or both the position and size of elements. The corresponding parameters of fixed elements are excluded from layout optimization.
- *Temporary grouping.* When the user selects multiple elements, the optimizer preserves relative positions and sizes among them.
- *Local exploration.* To prevent the suggestions from changing significantly from the current layout, we add extra terms to penalize quadratically for both movement and scaling of the elements.

Note that since the layout tree is available in the interactive setting, we use it instead of the optimized tree in Eq. (1). The working behavior of our tool can be found in the supplemental video.

### 6.2. Setup

We conducted an online user study using MTurk. We instructed the workers to assume the following scenario: *"Suppose your friend asks you to create a one-page web site and provides you with two reference designs. Your goal is to design a website following the style of the reference designs."* We also presented a demonstration video to show them how to use the interface. Each worker was assigned tasks to create five layouts for different designs, and randomly assigned either the proposed method or the LLSPGD method. In the case of the LLSPGD method, we fix the layout tree to flat and disable its editing. The layout is initialized with the layout optimization method. The worker then revises the design using

the provided tool and submits a design when the worker is satisfied with the result. After completing the design task, we asked them to answer a survey about the usability and suggestion. We use the completion time and the completed design to compare the methods.

#### 6.2.1. Results

59 workers participated in our experiment, and their ages ranged from 19 to 61 years old. $4 for a task. 31 workers used LLSPGD and the remaining 28 workers used our method. In the early experiments, the results were not stored correctly, so there were fewer valid results. 128 designs were created using LLSPGD and 116 designs were created using our method. We report only the high-level results, and the details are provided in the supplemental material.

The results of the survey show that our tool was perceived as easy to use, and many workers found the task enjoyable. Regarding the suggestions, roughly half of the workers responded positively, but it was difficult to clearly claim the differences in response trends based on the backend methods. This result may suggest that users' expectations of layout suggestions are high, and that there is a need to control their expectations and further improve the method.

We conducted a pairwise comparison in the same way as in Section 5.3. We selected the first 100 completed designs for both methods and collected five votes for each comparison. We used the Pearson's chi-square test and do not found a significant difference in the number of votes for both questions about quality ($p = 0.53$) and similarity ($p = 0.33$), which is to be expected as the users are allowed to take time to edit until they are satisfied with the results.

We then compare the two sets of completion time. The average completion time using the proposed method roughly 38% shorter than that using the LLSPGD method (3.7 and 5.1 minutes, respectively). This result indicates that the proposed method was able to omit more necessary editing through the layout initialization. Using the Mann-Whitney U test (two-sided), we found there to not be a statistically significant difference in the completion time ($p = 0.12$). This is likely due to noisy network issues and the fact that workers can freely pause the task execution. It is likely that more experiments in more controlled settings might be able to give more significance to the result.

## 7. Limitations and Discussion

Modelling visual containment as a layout tree and integrating it into the optimization is able to improve the quality of the results while enabling processing of design layouts with a large number of elements, which was not possible with previous approaches. For applying our method to other graphic layout problems, a hierarchical structure is required to train the tree energies. In mobile app layouts, for example, the view hierarchy could be used. If such structure is not available, it should be possible to use some heuristic rules to create a layout tree instead of obtaining one by estimation. One of the drawbacks of our method comes from the pipeline, *i.e.*, a failure in tree optimization makes it difficult to succeed in later layout optimization, and we believe that the joint optimization of tree and layout is worth investigating. Also, with a tree-based approach like ours, it is difficult to define the layout structure when an element overlaps two or more elements at the same time. Such cases are rarely found in web pages, but to adapt the similar concept to other domains, a directed acyclic graph-based approach, where overlaps are represented by edges, may be a reasonable modification.

While our method can support any screen width thanks to the relative parameterization, the inability of handling alternative elements and the necessity of placing all the given elements can be obstacles in applying our method to a responsive design scenario, such as converting a PC design to a mobile one. An extension to these limitations could be to add indicator variables for pre-defined alternatives (*e.g.*, text elements with different line breaks) and visibilities of elements to the optimization target. Further improvements may require a tractable modeling of the web rendering process.

Additionally, while the approach is significantly faster and more efficient than the existing works, learning styles from few references by inverse optimization is still too computationally expensive, and handcrafting the basis energies can be a drag on the representation capability of the model. Our collected dataset allows us to study deep learning based approaches, and we believe that a promising research direction for layout generation is to introduce a few-shot learning scheme [WYKN20].

Our experiments also show that the dataset plays a critical role in the results, with better reference designs providing a large increase in performance. To efficiently search for better reference designs that users are looking for in their minds, adopting a method that interactively searches over an approximated data distribution [CKL*20] may be an interesting line of research.

## Acknowledgement

This work is partially supported by Waseda University Leading Graduate Program for Embodiment Informatics.

## References

[ASY*19] AKIBA T., SANO S., YANASE T., OHTA T., KOYAMA M.: Optuna: A next-generation hyperparameter optimization framework. In *ACM SIGKDD* (2019), pp. 2623–2631. 6

[Bel18] BELTRAMELLI T.: Pix2code: Generating code from a graphical user interface screenshot. In *CHI EICS* (2018), pp. 1–6. 3

[Boo21] BOOTSTRAP: Overview · bootstrap, 2021. URL: https://getbootstrap.com/docs/4.1/layout/overview/#containers. 2

[Bre01] BREIMAN L.: Random forests. *Machine learning 45*, 1 (2001), 5–32. 5

[CCL12] CAO Y., CHAN A. B., LAU R. W. H.: Automatic stylistic manga layout. *ACM TOG 31*, 6 (2012). 2

[CFX*19] CHEN C., FENG S., XING Z., LIU L., ZHAO S., WANG J.: Gallery d.c.: Design search and knowledge discovery through auto-created gui component gallery. *ACM HCI 3*, CSCW (2019). 2

[CKL*20] CHIU C.-H., KOYAMA Y., LAI Y.-C., IGARASHI T., YUE Y.: Human-in-the-loop differential subspace search in high-dimensional latent space. *ACM TOG 39*, 4 (2020). 11

[DHF*17] DEKA B., HUANG Z., FRANZEN C., HIBSCHMAN J., AFERGAN D., LI Y., NICHOLS J., KUMAR R.: Rico: A mobile app dataset for building data-driven design applications. In *ACM UIST* (2017), p. 845–854. 3, 4, 7

[DTSO20] DAYAMA N. R., TODI K., SAARELAINEN T., OULASVIRTA A.: Grids: Interactive layout design with integer programming. In *ACM SIGCHI* (2020), p. 1–13. 2, 7

[FGO09] FLAVIAN C., GURREA R., ORÚS C.: Web design: a key factor for the website success. *J. Syst. Inf. Technol. 11*, 2 (2009). 1

[Han16] HANSEN N.: The CMA evolution strategy: A tutorial. *CoRR abs/1604.00772* (2016). 2, 7

[HWCK07] HARTMANN B., WU L., COLLINS K., KLEMMER S. R.: Programming by a sample: Rapidly creating web applications with d.mix. In *ACM UIST* (2007), p. 241–250. 3

[Int21] INTERNET A.: Keyword research, competitive analysis, & website ranking | alexa, 2021. URL: https://www.alexa.com/. 3

[JLS*03] JACOBS C., LI W., SCHRIER E., BARGERON D., SALESIN D.: Adaptive grid-based document layout. In *ACM SIGGRAPH* (2003), p. 838–847. 2

[KGV83] KIRKPATRICK S., GELATT C. D., VECCHI M. P.: Optimization by simulated annealing. *Science 220*, 4598 (1983), 671–680. 5

[KST*13] KUMAR R., SATYANARAYAN A., TORRES C., LIM M., AHMAD S., KLEMMER S. R., TALTON J. O.: Webzeitgeist: Design mining the web. In *ACM SIGCHI* (2013), p. 3083–3092. 3

[KTAK11] KUMAR R., TALTON J. O., AHMAD S., KLEMMER S. R.: Bricolage: Example-based retargeting for web design. In *ACM SIGCHI* (2011), p. 2197–2206. 3

[LAZ*20] LI Y., AMELOT J., ZHOU X., BENGIO S., SI S.: Auto completion of user interface layout design using transformer-based tree decoders. *CoRR abs/2001.05308* (2020). 2

[LCS*18] LIU T. F., CRAFT M., SITU J., YUMER E., MECH R., KUMAR R.: Learning design semantics for mobile apps. In *ACM UIST* (2018), p. 569–579. 7

[LHP05] LIU C. K., HERTZMANN A., POPOVIĆ Z.: Learning physics-based motion style with nonlinear inverse optimization. *ACM TOG 24*, 3 (2005), 1071–1081. 7

[LJE*20] LEE H.-Y., JIANG L., ESSA I., LE P. B., GONG H., YANG M.-H., YANG W.: Neural design network: Graphic layout generation with constraints. In *ECCV* (2020), pp. 491–506. 2

[LKH*20] LEE C., KIM S., HAN D., YANG H., PARK Y.-W., KWON B. C., KO S.: Guicomp: A gui design assistant with real-time, multi-faceted feedback. In *ACM SIGCHI* (2020), p. 1–13. 3

[LNDO20] LAINE M., NAKAJIMA A., DAYAMA N., OULASVIRTA A.: Layout as a service (laas): A service platform for self-optimizing web layouts. In *ICWE* (2020), pp. 19–26. 2

[LSK*10] LEE B., SRIVASTAVA S., KUMAR R., BRAFMAN R., KLEMMER S. R.: Designing with interactive example galleries. In *ACM SIGCHI* (2010), p. 2257–2266. 2

[LXZ*19] LI J., XU T., ZHANG J., HERTZMANN A., YANG J.: LayoutGAN: Generating graphic layouts with wireframe discriminator. In *ICLR* (2019). 2

[LYZ*20] LI J., YANG J., ZHANG J., LIU C., WANG C., XU T.: Attribute-conditioned layout gan for automatic graphic design. *IEEE TVCG Early Access* (2020). 2

[LZS*21] LAINE M., ZHANG Y., SANTALA S., JOKINEN J. P. P., OULASVIRTA A.: Responsive and personalized web layouts with integer programming. In *ACM HCI* (2021), vol. 5. 2

[Net20] NETCRAFT: December 2020 Web Server Survey. https://news.netcraft.com/archives/2020/12/22/december-2020-web-server-survey.html, 2020. [Online: accessed 25-January-2021]. 1

[NMC05] NICULESCU-MIZIL A., CARUANA R.: Predicting good probabilities with supervised learning. In *ICML* (2005), pp. 625–632. 5

[NSvHM14] NGUYEN L.-T., SCHMIDT H. A., VON HAESELER A., MINH B. Q.: IQ-TREE: A fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Mol. Biol. Evol. 32*, 1 (2014), 268–274. 5

[OAH14] O'DONOVAN P., AGARWALA A., HERTZMANN A.: Learning layouts for single-page graphic designs. *IEEE TVCG 20*, 8 (2014), 1200–1213. 2, 4, 6, 7, 8

[OAH15] O'DONOVAN P., AGARWALA A., HERTZMANN A.: Designscape: Design with interactive layout suggestions. In *ACM SIGCHI* (2015), p. 1221–1224. 2, 3, 7, 8, 10

[O'D15] O'DONOVAN P.: *Learning Design: Aesthetic Models for Color, Layout and Typography*. PhD thesis, University of Toronto, 2015. 7

[ODPK*18] OULASVIRTA A., DE PASCALE S., KOCH J., LANGERAK T., JOKINEN J., TODI K., LAINE M., KRISTHOMBUGE M., ZHU Y., MINIUKOVICH A., PALMAS G., WEINKAUF T.: Aalto interface metrics (aim): A service and codebase for computational gui evaluation. In *ACM UIST* (2018), pp. 16–19. 3

[PCLC16] PANG X., CAO Y., LAU R. W. H., CHAN A. B.: Directing user attention via visual flow on web designs. *ACM TOG 35*, 6 (2016). 2

[QFY*19] QIANG Y.-T., FU Y.-W., YU X., GUO Y.-W., ZHOU Z.-H., SIGAL L.: Learning to generate posters of scientific papers by probabilistic graphical models. *Journal of Comput. Sci. and Technol. 34*, 1 (2019). 2

[RKK11] RITCHIE D., KEJRIWAL A. A., KLEMMER S. R.: D.tour: Style-based exploration of design example galleries. In *ACM UIST* (2011), p. 165–174. 2

[Sel21] SELENIUM: Seleniumhq browser automation, 2021. URL: https://www.selenium.dev/. 3

[SK15] SINHA N., KARIM R.: Responsive designs in a snap. In *ACM FSE* (2015), p. 544–554. 2

[Skl11] SKLAR J.: *Principles of web design: the web technologies series*. 2011. 2

[SWO*20] SWEARNGIN A., WANG C., OLESON A., FOGARTY J., KO A. J.: Scout: Rapid exploration of interface layout alternatives through high-level design constraints. In *ACM SIGCHI* (2020), p. 1–13. 2, 3, 6

[Tho07] THORLACIUS L.: The role of aesthetics in web design. *Nordicom Review 28*, 1 (2007), 63–76. 1

[TWO16] TODI K., WEIR D., OULASVIRTA A.: Sketchplore: Sketch and explore with a layout optimiser. In *ACM DIS* (2016), p. 543–555. 3

[VV15] VERNICA R., VENKATA N. D.: Aero: An extensible framework for adaptive web layout synthesis. In *ACM DocEng* (2015), p. 187–190. 2

[Web21] WEBFLOW: Container | webflow university, 2021. URL: https://university.webflow.com/lesson/container. 2

[Wil15] WILLIAMS R.: *The non-designer's design book: Design and typographic principles for the visual novice*. 2015. 2

[WYKN20] WANG Y., YAO Q., KWOK J. T., NI L. M.: Generalizing from a few examples: A survey on few-shot learning. *ACM Comput. Surv. 53*, 3 (2020). 11

[YMX*16] YANG X., MEI T., XU Y.-Q., RUI Y., LI S.: Automatic generation of visual-textual presentation layout. *ACM TOMM 12*, 2 (2016). 2

[ZCL18] ZHAO N., CAO Y., LAU R. W.: Modeling fonts in context: Font prediction on web designs. *Comput. Graph. Forum 37*, 7 (2018), 385–395. 3

[ZHM19] ZHENG S., HU Z., MA Y.: Faceoff: Assisting the manifestation design of web graphical user interface. In *ACM WSDM* (2019), p. 774–777. 3

[ZQCL19] ZHENG X., QIAO X., CAO Y., LAU R. W. H.: Content-aware generative modeling of graphic design layouts. *ACM TOG 38*, 4 (2019). 2, 4