Sketchplore: Sketch and Explore with a Layout Optimiser

Kashyap Todi[1,2]
kashyap.todi@uhasselt.be

Daryl Weir[1]
daryl.weir@aalto.fi

Antti Oulasvirta[1]
antti.oulasvirta@aalto.fi

[1]Aalto University
Helsinki, Finland

[2]Hasselt University - tUL - iMinds
Diepenbeek, Belgium

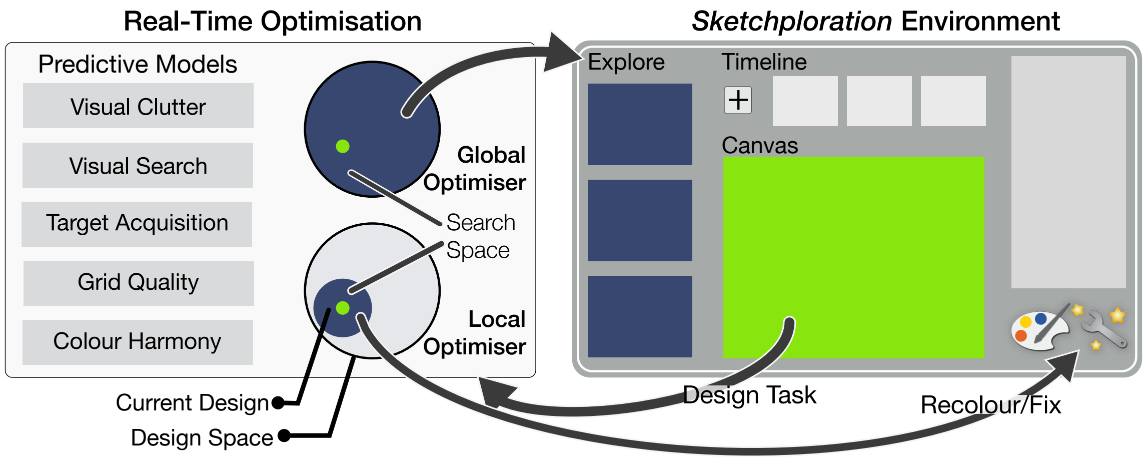


Figure 1: *Sketchplorer* is an interactive layout sketching tool supported by real-time model-based optimisation. The tool is designed to facilitate the creative and problem-solving aspects of sketching without requiring extensive input. While a designer is sketching, a design task is automatically inferred. The optimiser uses predictive models to make suggestions for local and global changes that improve usability and aesthetics. Suggestions appear on the side, and never override the designer's work.

ABSTRACT

This paper studies a novel concept for integrating real-time design optimisation to a sketching tool. Although optimisation methods can attack very complex design problems, their insistence on precise objectives and a point optimum is a poor fit with sketching practices. *Sketchplorer* is a multi-touch sketching tool that uses a real-time layout optimiser. It automatically infers the designer's task to search for both local improvements to the current design and global (radical) alternatives. Using predictive models of sensorimotor performance and perception, these suggestions steer the designer toward more usable and aesthetic layouts without overriding the designer or demanding extensive input.

Author Keywords
Sketching; Model-based optimisation; Visual Layouts

ACM Classification Keywords
H.5.2. Information Interfaces and Presentation: User Interfaces

INTRODUCTION

This paper is motivated by the observation that optimisation methods have great untapped potential in design tools. We focus on the activity of *sketching layouts*, in which a designer places, colours, organises, and defines elements on a canvas. From a combinatorial perspective, the design of layouts is notoriously hard. For a canvas of 1024$\times$768 pixels, divided into a 24$\times$32 grid, as in the tool presented here, there are 158,400 one-element layouts and a whopping 10^{41} eight-element layouts. Although algorithms may not be able to find the optimal solution in such large search spaces, they can "parallelise" search, and find candidate solutions and suggest them to designers. This could help designers in exploration, who are known to be limited to a handful of designs per iteration [8]. Also, algorithms can complement designers by exploring design spaces neutrally without being constrained by past experiences, to produce designs that the designer might not otherwise conceive. Employing an optimiser might also improve the quality of designs for end-users (see [12, 33, 45]).

However, several hard research challenges emerge. First, layout design is a complex, multi-objective task addressing not only usability but also aesthetic qualities [15, 44]. Presently no algorithmic approach exists that can address both. Second, optimisation typically takes a long time, due to combinatorial complexity, and no solution has been shown for fast-paced, iterative design of layouts. Third, although optimisation methods can attack very complex design problems, their insistence on *precise* inputs contradicts sketching. To better fit with design practice, optimisers should radically reduce the effort for defining tasks (see e.g., [2]). Crucially, they should not attempt to override design thinking. Instead, final decisions should be left to designers, who possess knowledge that computers might not.

This paper investigates *'sketchploration'*, a novel concept to exploit interactive optimisation methods in design tools and in particular for layout sketching. We approach sketching as a *problem-solving* activity and a tool for *visual thinking* [6, 30]. The goal is to enable access to results of real-time optimisation, yet impose as few control requirements on the designer as possible, in order to support the natural flow of sketching. For this end, Sketchplorer, illustrated in Figure 1, relaxes typical requirements for controlling an optimiser in a way that respects the design process. Unlike with previous solutions (e.g., [2, 12]), no additional input is required from the designer. As the designer sketches, the system infers the designer's task automatically. The optimiser simultaneously searches for local improvements (small changes), and explores global alternatives (large changes). Importantly, it deploys several predictive models of user performance and perception adapted from literature. This allows it to make informed suggestions that "pull" the designer toward usable and aesthetic designs. Sketchplorer does not override the designers' sketch, but presents optimised designs as glanceable suggestions. Technically, sketchploration extends *model-based interface optimisation* [2, 12, 45] to real-time design exploration under ill-specified and changing design goals.

This paper presents the first investigation of the concept, focusing on *interactive layouts* familiar from GUIs, web pages, menus, and dialogs. It complements existing work by showing how an interactive optimiser can be integrated with fast-paced and unconstrained early-stage sketching. Our implementation presently supports 10 common types of elements and hierarchical (nested) placement. But concept could be implemented for any sketching tool that affords extending the workspace and communication with a server. The technical contributions include:

1. *Design principles* for integrating a model-based interface optimiser to a sketching tool.
2. *Extension of model-based interface optimisation* to interactive layouts by: 1) Formulation of (unconstrained) layout sketching as an optimisation problem, allowing for addressing the positions, sizes, and colours of elements. 2) A novel, theoretically informed objective function addressing five aspects important in layout use: perception of clutter, visual search performance, pointing performance, grid quality, and colour harmony.
3. *A dynamic optimisation approach* that 1) infers the designer's task from the current layout, 2) simultaneously explores and exploits in the design space and 3) reacts rapidly to changes in the current sketch.

We report results from two empirical evaluations—with end-users and with trained designers. We conclude that interactive model-based optimisation warrants more attention.

PRIOR ART: SKETCHING TOOLS AND UI OPTIMISATION

Our goal is to support sketching with computational methods. In this work, we investigate sketching as a tool for visual thinking and problem-solving, and do not delve into freehand drawing and rendering aspects of sketches. When understood as problem-solving, sketching is characterised by its quick, ambiguous, and uncertain nature [13, 42]. The goal of designers is not a point design, but rather exploration. Sketching unfolds as an iterative process of idea-generation, refinement (exploitation), and redefinition. Designers entertain multiple hypotheses and may backtrack to previous designs. Details to a sketch are added gradually, and they can be changed at any stage. Idea-generation in sketching has recognised limits and biases [8]. These properties make sketching both a challenge and an opportunity for computational support.

Sketching Tools and Interaction Techniques

Research on computational support for sketching originates from Sutherland's Sketchpad [37], which highlighted the benefits of a digital medium. We refer to a recent review [17] and here point out a few main trends in research.

First, improved recognition technologies have brought pen-and-paper like techniques to sketching [41]. Second, several interaction techniques have been proposed to enhance the drawing of shapes in sketching (e.g., [1]). Third, some approaches have looked at integration with other activities in design, and a better support for going back and forth between designs. SILK [23] and DENIM [25] explored several such techniques. Fourth, sketching has been extended from 2D spatial displays. For example, [19] enabled sketching for animations and dynamic authoring of illustrations. Fifth, design heuristics have been implemented in sketching tools. Examples include colour palettes [29], template-based sketching [18], and design guidelines. While these ensure that designs meet certain standards, they allow neither exploring designs nor refining them for some objectives.

Heuristic and Data-Driven Methods for Layout Generation

Heuristic approaches to layout generation have explored balance, consistency, or the golden cut (e.g. [11, 27]). Although heuristics can produce results that are visually appealing and resemble real designs, they do not predict effects on end-users. They lack means for conflict resolution. To our knowledge, there are no heuristic approaches that solve both spatial and visual aspects of layout.

Data-driven approaches such as Webzeitgeist [22] mine a large number of designs and can produce new ones for designer given inputs. Although colour, size, position, and grouping can be addressed, the approach results in basically "mimicry" of existing designs. It does not offer a principled way of setting objectives for goals like usability and aesthetics. They offer no guarantee that the outcomes are good beyond visual appeal.

Some work has been done on automatic generation and improvement of static graphical media such as posters, flyers, or slides. [31] used an energy-based model considering aesthetic heuristics, such as alignment, balance, and flow. DesignScape

[32] is a tool for assisting novice designers in creating graphical media. While our work, at first glance, resembles DesignScape, the assumptions and underlying mechanisms of the two systems are very different. DesignScape does not consider interactive layouts, which form the basis of GUIs. Additionally, the tool is meant to improve final-stage designs, based on realistic content. It is not designed for the uncertain and ambiguous nature of initial-stage sketching, where the actual content is still subject to change. With regard to underlying optimisation mechanisms, DesignScape uses a reduced design space which does not include element colour, and makes simplifying assumptions about aesthetics rather than using validated predicitive models from psychology.

Finally, there has been extensive work on automating layout generation based on constraint satisfaction. Interfaces are specified as a set of semantic and spatial constraints, and solver programs generate possible designs for given screen sizes. [26] reviews these techniques. However, the constraints still have to be constructed by a designer, and moreover the generated layouts have no guarantees about aesthetic quality or interaction performance.

Model-based Optimisation

Model-based user interface optimisation uses combinatorial optimisation to automate interface generation. The idea is to represent a design problem and design knowledge (e.g., user simulations, models, heuristics) as an objective function for a search algorithm that iteratively improves designs for the stated objectives. Unlike heuristic approaches, model-based optimisation relies on theories and predictive models of how users interact or perceive a layout and an algorithm that searches the design space systematically. The most well-known layout problem is the letter assignment problem. Fitts' law can be used together with bigram data on transition frequency to find keyboard layouts that minimise expected finger travel distance [24]. This idea has been extended to widget layouts in SUPPLE using branch-and-bound [12].

The first *interactive* design tool following model-based optimisation was MenuOptimizer [2]. It uses a model of menu search together with a consistency heuristic to optimise hierarchical menus for application. It integrated several types of support to the QtDesigner development environment. It introduced an interactive optimiser that proposed global and local changes like moving a menu item to improve user performance. However, it was designed for "point optimisation" and contained an overwhelming amount of controls and visualisations and insisted on problem specification for the optimiser. The authors concluded that designers mostly used global suggestions (suggestions for the whole menu system).

WALKTHROUGH AND DESIGN OVERVIEW

We had four objectives for integrating computational support to a design tool:

1. Support quick and ambiguous sketching, and leave room for uncertainty, by providing capabilities to defer the task of specifying details at any stage of the design process.
2. Allow fluent shifts between exploration and refinement.
3. Provide support for multiple hypotheses, and give an overview of progress, by providing a non-destructive, and editable, timeline of previous alternate designs.
4. Minimise user actions not related to the design activity itself by inferring details whenever possible.
5. Eliminate ambiguity, in designers' minds, while perceiving optimised results by communicating with the designer in a timely and predictable manner.

Walkthrough: Designing a Blog Page

This walkthrough illustrates the use of Sketchplorer from a designer's perspective.

Sketching the initial layout: Sketchplorer initially presents the designer with an empty canvas. The designer starts sketching by creating a structure for her design (Figure 2a). Drawn elements can be moved around, or resized, at any time. She ambiguously sketches out boxes, serving as proxies for the elements of her blog page. Sketchplorer dynamically infers hierarchy of elements, and does not require the designer to explicitly specify ordering or grouping. It starts computing both local and global suggestions in the background.

Refining and adding details: An *inspector* panel, similar to most commercial sketching tools, sits on the right-edge of the display, and can be pulled out at any time. This can be used to specify details, such as the element type, colour, and importance (usage probability) of objects on the canvas (Figure 2b). The designer now adds details to some of the elements. For instance, she indicates that her blog page has a heading and a paragraph element, and marks them as being of high importance for the optimiser. She also specifies that the image (site logo) is of low importance. Satisfied with the first version of the design, she taps the save ('+') button. This adds the current design to the designer's *timeline*, and provides a preview. The designs here can be retrieved, and edited, at any later time. This enables the designer to see the evolution in designs, borrow previous ideas, and select feasible alternatives.

Fine-tuning and local changes: While the designer sketches and refines, the system continuously streams the description of the current design to the optimiser. The local optimiser uses the current design as a starting point to suggest fixes and provide recolour options. Pulsating icons in the inspector panel appear when *fix* and *recolour* options are available (Figure 2c). The designer refers to these suggestions. She realises that by using the recolouring suggestion, she can make the paragraph of text stand out. She chooses one of the recoloured layouts, and continues working on her sketch.

Exploring new designs: By abstracting from the current design, the global optimiser retrieves unique designs, and returns them to the designer. An *explore* panel, residing on the left edge of the display (Figure 2d), is periodically updated with these designs. The designer browses through the list of alternatives, and finds two interesting alternatives. She adds the first to her list of saved designs, and drags the second onto the canvas, to continue working on it.

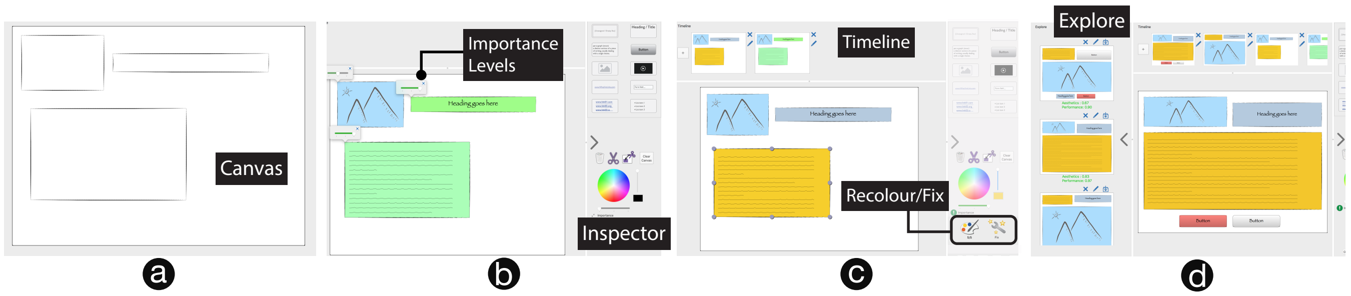


Figure 2: Overview of Sketchplorer. A large canvas is surrounded by the various features. Designers can (a) ambiguously sketch layout designs, and (b) refine and add details. They have immediate access to a timeline of saved designs. They can improve designs by (c) using *fix* and *recolour* suggestions, or (d) exploring globally optimised designs.

In a short duration, the designer's collection of *saved layouts* is populated with several feasible alternatives—some sketched by the designer, and the others with the aid of the optimiser. While the above phases appear to be linear, in practice, sketching and exploration phases are intertwined.

Overview of Interactions

Sketchplorer is designed for a multitouch environment using a large display, and uses touch gestures for all controls.

Sketching and refinement: Sketchplorer allows designers to either sketch ambiguous bounding boxes for layout elements, or pick out a specific element type and draw it on the canvas. It takes care to accurately order every element on the canvas, without requiring the user to explicitly *bring-to-front* or *send-to-back*. Each time an element is changed, so as to change the hierarchy, the ordering is dynamically adapted. Hierarchical groups of elements can be selected and manipulated at the same time. This inferred hierarchy is also essential for the internal representation of a layout in the optimiser. The colour picker allows designers to select the hue–saturation combination, and adjust the brightness. Double-tapping on an element reveals an in-place pop-over, and allows adding details without having to move to the inspector. Individual elements' importance can be adjusted in the inspector panel. Alternatively, an overlay can be enabled, that displays the importance of each element, and allows batch adjustments (Figure 2b).

Saved versions and timeline: Benefits of interactive timelines and alternate versions have been emphasised in HCI literature [38]. In Sketchplorer, the current design can be added to the timeline at any time. The timeline provides an overview of all saved designs, and any alternative design, or intermediate sketch, can be dragged back to canvas. This allows designers to compare designs, have an overview of the evolution of their designs, and iterate over sketches.

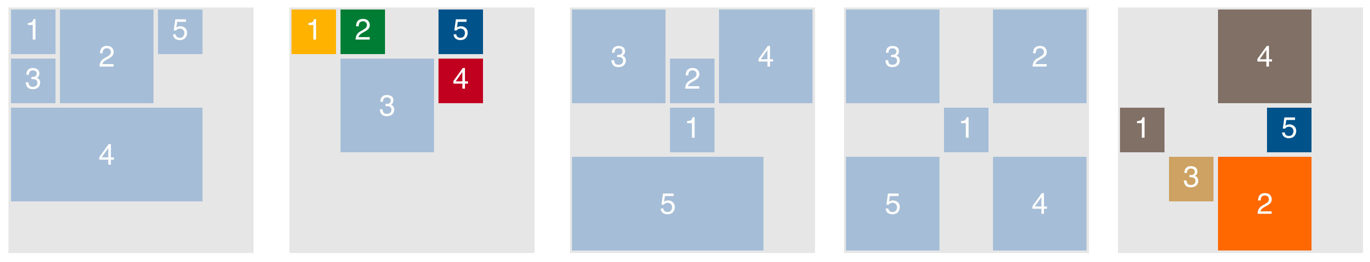
Integration with the optimiser: To the designer, the optimisation appears as a two-pronged approach, consisting of local and global optimisation. The local optimiser listens in on every change in the design, including changes in sizes and positions of elements. It creates fine-tuned designs, that maintain the overall composition of the original sketch, but improve certain aspects of it. Recolouring suggestions maintain the sizing and placement of elements, and offer harmonious recolouring suggestions, which also improve aesthetics and performance. In contrast, the global optimiser listens exclusively to changes in design tasks, and acts upon them. It abstracts away from exact details, allowing it to explore the entire design space, and generate unique designs. It performs exploration in real-time and periodically returns improved results that are immediately displayed to the designer in an expandable *explore* panel. The optimiser focuses on creating unique and improved solutions, not on refining or polishing a solution to make them perfect.

PREDICTIVE MODELS FOR INTERACTIVE LAYOUTS

Sketchplorer is the first user interface optimiser using models of human performance and perception to optimise both spatial and colour aspects of interactive layouts. We chose models that cover both aesthetic features, such as symmetry, and sensorimotor performance measures, such as target selection time. The models have also been validated in empirical studies and shown to align with user preferences and performance. In this way, we aim to produce layouts that are aesthetically pleasing and usable in a predictable way. However, note that the choice of models in the system is flexible. If there is evidence for another model being better than those we use currently, it is very simple to add this without changing the workflow of Sketchplorer.

Overview: The Colour Patches Task

In order to examine the effect of optimising for a model as an objective, and learn how the models affect optimisation, we created the *colour patches task*. Here, the optimiser's goal is to assign five rectangular elements to a 5×5 grid. Each element is a block or patch with a single colour. We assume that each patch is interactive, with a certain probability of being targeted. We sample these probabilities from a Zipf distribution. These probabilities are used to weight the average visual search and target acquisition times using a formula given later in the paper. To produce candidate designs, we traverse the elements in a random order and pick a random colour, size and position for each. The colours are chosen from Kelly's set of perceptually distinct colours [20], and the sizes are chosen randomly from three fixed sizes. For each objective, we evaluate many random designs to find the one with the lowest objective value, then perform small changes to search for local improvements in the design.



(a) Clutter (b) Visual search (c) Target acquisition (d) Grid quality (e) Colour harmony

Figure 3: Results for the colour patches task. These five layouts are outcomes when optimising for a *single objective at a time*. Sketchplorer combines these objectives into multi-objective search. Numbers refer to a ranking based on frequency of use.

Figure 3 shows the results. Elements are numbered by their usage probability, with 1 being the most common. The results show that each objective focuses on either spatial or colouring aspects, but may ignore or contradict the others. In the following sections, we will detail these results.

Visual Clutter

We use the Rosenholtz model [34] to minimise visual clutter. The idea is that as more objects are added to a display, it becomes more difficult to place a new object that it is perceptually unique and easy to identify. The layout elements have some distribution of visual features. As the number of elements grows, the feature distribution occupies more of the available space. This model has been shown to correlate with user perception of clutter, a correlate of aesthetic preference.

Given vectors of the features for each object on the display, we compute the mean and covariance $\mathbf{\Sigma}$ of these vectors. In our implementation, we consider only colour in our feature vector. The clutter of the display is then defined simply as the determinant of the covariance matrix, $|\mathbf{\Sigma}|$. This can be thought of as the volume of the 1 s.d. covariance ellipsoid. We take the inverse of the clutter score for the objective function.

Figure 3(a) shows the colour patch results for this model. Each item has the same colour, which makes sense as this choice minimises the volume of the colour covariance.

Visual Search

We implement the Kieras-Hornof model of visual attention [21] to maximise visual search performance by enhancing the perceptual uniqueness of commonly searched elements.

Prediction consists of two parts: (1) a set of availability functions determine whether the features of a target are perceivable from the user's current eye location; and (2) a simple GOMS-style algorithm estimates the time in milliseconds required to locate the target.The availability functions are based on the eccentricity from the current eye location e and angular size s of the target. Additive Gaussian random noise with variance $\sigma^2 = vs$ proportional to the size of the target is assumed. For each feature, a threshold t is computed as $t = ae^2 + be + c$ and the probability that the feature is perceivable is given by:

$$P(\text{available}) = 1 - \Phi\left(\frac{t-s}{\sigma^2}\right), \quad (1)$$

where $\Phi(x)$ is the c.d.f. of the standard normal. We further define $P(\text{available}) = 1$ when $e < 1°$, since targets in the fovea are always perceived.

We use parameter values from the original paper, which were shown to accurately predict visual search time for data from an older study [40]. Note that our implementation only uses colour as an availability feature, since the authors found size and shape could only reliably be perceived for targets close to the fixation location.

For memory effects we use a simple logarithmic model based on the number of previous acquisitions. We optimise assuming the user is an expert and has visually searched the interface 1000 times in total. However, this assumption can be easily changed for example to account for novices.

Figure 3(b) shows the visual search colour patches. Each item has a different colour to make it visually unique, and all items are clustered in the top left of the grid, where the simulated search begins. More important items are closer to the left corner. Note that this objective conflicts with the clutter metric, which rewards layouts where everything is the same colour. Thus, there is a tradeoff between aesthetics and visual search performance that can be controlled by adjusting the weights.

Target Acquisition

Following previous work on performance-optimisation of layouts [24, 45], we deploy a variant of Fitts' law [28] as our model of target acquisition. Fitts' law favours transitions between large elements at minimum distances, and estimates the upper bound of expert pointing performance. Including this model allows us to optimise for efficient selection of elements, rather than purely for aesthetics. T is the time it takes for an end-effector to reach a target from a given distance D. For target width W, T is given by:

$$T = \sum_{r \in R} t_r = \sum_{r \in R} \left[a + b \log_2\left(\frac{D}{W} + 1\right)\right] \quad (2)$$

where R is the set of responses when navigating to the target.

When optimising, we assume that users interact with the interface elements using touch. We use values of a and b from a recent study [10]. To compute D, we assume that the starting point of the finger is the centre of the display. We compute W according to the angle of approach. Fitts' law parameters for other input devices, such as the mouse, are readily available, and the optimiser can easily be adapted for these devices.

Figure 3(c) shows the colour patches for target acquistion. The most important element is centrally placed, close to the assumed starting position. The other elements are arranged around the center and have larger sizes so that they can be acquired without moving the finger far from the center.

Grid Quality

We use the Balinsky symmetry metric [3] as a measure of grid quality. It calculates the distance to the closest symmetrical layout. It was shown to correlate with aesthetic preferences.

We start with the layout vertex set $s = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, rescaled such that the x-axis coincides with the axis of symmetry and the y-axis contains the centre of mass of the x_i. This is mapped to the complex plane to obtain a set $z_i = x_i + Iy_i$. These points are horizontally symmetrical if they consist only of real values and complex conjugate pairs. A fundamental result in complex analysis states that this is equivalent to the stipulation that all coefficients a_j of the polynomial

$$P_n(z) = \prod_{j=1}^{n}(z - z_j) = \sum_{j=0}^{n} a_j z^j \tag{3}$$

are real. Thus, the asymmetry for a layout is obtained by constructing this polynomial and averaging the size of the imaginary parts of the coefficients. Vertical symmetry is scored in the same way, after a simple coordinate transformation. Our overall metric is an even weighting of these two scores.

Figure 3(d) shows the colour patches for this metric. As expected, the result is symmetric in both axes.

Colour Harmony

Harmonic colours are sets of colours that combine to provide a pleasant visual perception. Harmony is determined by relative position in colour space. There is no universal definition for a harmonic set. Here, we use the templates proposed in [7]. These consist of one or two sectors of the hue wheel, with given angular sizes. These templates can be arbitrarily rotated to create new sets.

The distance of a layout X from a template T is given by

$$D(X, T) = \sum_{e \in X} DH(e, T) S(e), \tag{4}$$

where $DH(e, T)$ is the arc-length distance between the hue of element e and the hue of the closest sector border in T, and $S(e)$ is the saturation channel of e. If a colour falls inside one of the sectors of T, $DH(e, T)$ is identically zero.

In order to rapidly evaluate many layouts, we precreate a set C of 78 harmonic sets by rotating the templates in fixed increments. We define the colour harmony H by:

$$H(X) = \min_{T \in C} D(X, T) \tag{5}$$

Figure 3(e) shows the colour patches with the best match to one of the harmonic sets. Patches 1–4 have similar hues but different saturations, and patch 5 has a hue on the opposite side of the colour wheel. Together these colours form a harmonic set with the Y-template defined in [7]. Note that the placement and ordering of the patches is somewhat chaotic, since this model does not consider those factors.

Scope and Limitations

These models allow us to deal with the positions, size, alignment and colour of layout elements. An optimiser is able to search for layouts that improve for user performance, but also for aesthetic qualities. If implemented alone, each objective drives a layouts to an unbalanced design favouring one aspect. A multi-objective optimiser allows finding good compromises. A limitation is that these models do not allow us to deal with semantic aspects of layouts, such as naming or grouping of elements. These decisions are currently left to the designer, but could be incorporated in future versions if models for these aspects are found.

DYNAMIC LAYOUT OPTIMISATION DURING SKETCHING

Layout design is a high-dimensional, NP-hard problem [9, 36]. Search is computationally expensive, as multiple objective values are calculated. Yet, when integrated with fast-paced sketching, only brief computation times are allowed. Further, the design problem is also under-specified, because the designer may not have time to specify point objectives and assumptions for each sketch.

Our approach builds on three ideas: First, we relax the requirement to specify a design task to an optimiser. Instead, we infer the "implicit" design task assumed in the sketch currently edited. The internal representation of the *design task*, in its simplest form, is comprised of the different layout elements placed on the canvas, and their 'importance' values, if specified. Using this, We can search alternatives without asking the designer anything. Second, we relax the assumption that the goal is to find the optimum design. Instead, we run several optimisers simultaneously that *explore* the design space with different assumptions. We pick a few diverse design ideas and pass them to the designer, and let his/her choices guide the search. Third, to accelerate search, we make use of the observation that there are far fewer (abstract) *design tasks* than there are (concrete) layouts. Using this observation, we pre-train an "associative memory" that has a good starting point for many design tasks that might occur. When Sketchplorer is running, it maps the present sketch to the closest seed in the associative memory to offer a good solution very quickly. By contrast, MenuOptimizer assumed a point-optimisation goal and enforced a complete reset of the optimiser's state when a single element was changed [2].

Definition: Layout Design Task

Layout design is similar to the letter assignment problem from keyboard optimisation [5] and to the *layout facility problem* (LFP), where the task is to place machines with fixed/varying size in a workshop [9, 36]. Layout design in HCI, as we address it here, is a special case, where the goal is to find a non-overlapping planar orthogonal arrangement of n coloured rectangular elements so as to minimise the cost function (see below). There are two differences to LFP: 1) elements can be coloured; and 2) the cost function is more complex. We make two further additions to the task. First, we define the task to involve *element types*, each with unique size constraints. Second, we allow groups. The elements can be nested as long as their bounding box edges do not overlap.

Internally, our optimiser represents a layout as a 2D array of pointers to information objects which store the colours, usage probabilities, etc. of the elements. We use an efficient maximal empty rectangles algorithm [39] to find places in this array where we can move or grow elements to modify the layout. Additionally, each element also has an internal array representing the positions of any child elements relative to itself. We treat the canvas in a recursive loop from the highest to the lowest level of this hierarchy.

Objective Function

We define a multi-objective task where we seek to minimise a weighted combination of the outputs of the five models:

$$U = \sum_{i=1}^{5} w_i S_i, \tag{6}$$

where the weights w_i sum to 1 and the individual objectives S_i are normalised to the range $[0, 1]$.

In addition, we introduce two optional canvas-level constraints. We compute *alignment* following an approach similar to [4], in which we count the number of 'alignment lines', or object edge positions, weighted by the average size of object edges lying on those lines. We compute *packing efficiency* as the proportion of the minimum bounding rectangle for the layout elements which is empty, averaged with the proportion of the overall canvas which is empty. These help the optimiser improve quicker in the grid quality dimension. They align elements horizontally and vertically and "pack" them. However, since they aggressively push search toward harmonic layouts, it may drive search to a local minima. We address this by introducing a filtering stage where we compare results for solution quality and diversity (see below).

Inferring the Design Task

We extract the *design task* by analysing the present layout being edited by the designer. We map a layout to a design task by 1) counting the number of occurrences for each element type, and 2) making a distinction between high and low importance items for each element type.

Dynamic Optimisation

During sketching, the optimiser system runs several optimisers simultaneously, each exploring the design space differently, and dedicated to a functionality in the Sketchplorer workspace. An overview is given in Table 1. The optimisers deploy two search heuristics.

Variable Neighbourhood Search (VNS) [14] is a strategy combining a neighbourhood-based meta-heuristic to a hill-climbing algorithm. This strategy was chosen to ensure good exploration of the design space, since it is able to increase the search radius if the search gets stuck. It allows larger and larger changes to be made in each iteration. In our case,

Sketchplorer Feature	Duration (s)	Search method	Neighbourhood size and type	Objective function
Recolour	1-10 s	VNS	Small; only colour changes	Colour only
Fix	1-10 s	VNS	Small; only spatial changes	All
Explore	1-10 s	AM	Large; only spatial changes	All
Explore	10-120 s	VNS	Medium; only spatial changes	All

VNS: Variable Neighbourhood Search, AM: Associative Memory

Table 1: Sketchplorer uses a multi-threaded optimisation strategy that both explores and exploits. To support dynamic changes in the task, two methods are used in parallel: Variable Neighbourhood Search and Associative Memory.

we run a steepest descent with different pre-set neighbourhood sizes. A neighbourhood is defined as a change to the colour, position, or size of an element. Small neighbourhood means that one iteration can only produce one change to a layout. Large neighbourhood means that multiple changes are allowed. The optimisers use VNS with small (e.g. $1, 2, 3$) or medium radii (e.g. 4 to 10). Figure 4 shows good and failed designs produced in about 4 minutes on a laptop computer. Since Sketchplorer is targeted towards skilled designers, we take a mixed-initiative approach [16], and rely on the designer to identify good designs, and filter out unappealing or illogical ones.

Associative Memory (AM) [43] is executed in parallel. AM is a memory-based learning approach to dynamic optimisation problems. The idea is to populate the memory with good solutions offline, and map the current design task to its closest-matching task in AM, and use that as a starting point for search. We implemented *layout remapping* by searching AM for layouts with design tasks that were supersets of the current design, ordering the elements in each by usage probability, then searching for matched type pairs. We trained our AM for 6,600 randomly generated design tasks. The live optimiser loaded the AM in advance in order to respond quickly (within few seconds).

Filtering and Diversification of Results

Since the parallel searchers explore different parts of the design space, and we cannot show all results to the designer, we pool the results and filter them using a bi-objective pareto front criterion. We define this as the weighted (and normalised) sum of the objective score and *inter-layout distance*. Inter-layout distance is a score from 0 to 1 where 0 means identical layouts and 1 means that all elements have different locations. This diversifies search results and prevents identical designs from being shown to the designer.

SYSTEM IMPLEMENTATION

The tool is implemented in Objective-C. Multitouch input is detected using PQ Labs multitouch SDK, and interactions are recognised using customised gesture recognition. The optimisers are written in Python, and use parallelisation for speed-ups. When designers make changes to the current design, layout files (in JSON format) are sent from the design

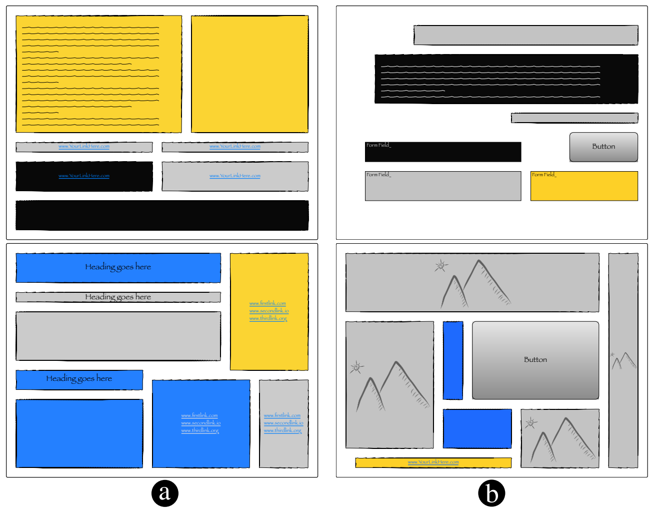


Figure 4: Examples of (a) successful and (b) unsuccessful global designs from the Associative Memory, *before* real-time optimisation. Designs were computed from randomised starting points, in 4 minutes each.

tool to the optimisers asynchronously. The optimisers run on dedicated laptops (we use Macbook Pros), and all devices communicate using ThoMoNetworking. Local vs. global design changes are identified using filename tags. To avoid outdated retrievals, the optimisers kill existing threads when a new file is received. Generated designs are immediately pushed back to the design tool.

STUDY 1: END-USER EVALUATION

Our first study addresses optimisation of the design of the Windows Phone home screen. This study involves no designer in the loop. The goal is evaluate our optimisation approach by testing its outputs with *end-users*. As the case we chose the Windows Phone home screen, a complex design task that could not have been addressed with previous approaches. The task allows multiple icon sizes, colours, and positions over three pages. We compare an optimised design against the baseline in a study where users are asked to select applications from the home screen. As dependent variables we look at selection time and measures of perceived aesthetics. As our baseline, we implemented the factory default of a Lumia 930 phone. Apps which were not preinstalled on the phone were added in a random order at the end of the default menu, as in an unpersonalised order-of-installation layout. Comparing against a real, commercial design provides a hard baseline for the layout optimiser.

Optimisation Task

The inputs to the optimiser are a list of 55 apps, together with usage probabilities from the LiveLab database [35]. We sampled the primary colours from the app icons manually. When generating layouts in the optimiser, we allowed as many 8×6 grid pages as necessary, and each icon was sized at 1×1, 2×2, or 2×4 grid cells. For some apps, the icon is a transparency, coloured according to system-wide accent colour. Others have icons with developer-chosen colours. For the system-coloured icons, the optimiser was free to choose from a set of 20 perceptually distinct colours [20].

Optimisation follows a different approach than in the live, dynamic optimisation task. We here performed multiple restarts of a random search procedure with different objective weights. For each weight set, we generated and evaluated 10,000 random layouts, then performed 3000 iterations of local search around each of the best three designs.

The menu with the best score over all weight sets was chosen as the design to evaluate with users. The models predicted that our optimised design would be better than the baseline in terms of visual search (score 0.106 vs. 0.117, a difference of ~270ms), selection (0.108 vs 0.104, ~90ms), and grid quality (0.005 vs. 0.013), similar in colour harmony (both 0.015), and worse in terms of clutter (0.028 vs 0.020).

Participants

We recruited 20 student participants (4 male), aged 20 to 36 (mean 26.7, s.d. 5.2). They had 2 to 8 years of experience with smartphones (mean 4.3, s.d. 1.8). Participants were required to not have prior experience with Windows Phone devices. Two participants were left handed. Participants were compensated with a cinema ticket.

Apparatus, Procedure, and Experimental Design

We implemented a logging application to display the menus and collect performance information on a Nexus 5 smartphone running Android 4.4.3.

Participants performed 275 trials with each menu. In each trial, they were shown a textual stimulus with the target name. After tapping to dismiss the stimulus, the first page of the current menu was shown. Participants had to navigate to and tap the appropriate icon. A hint button was available if they forgot the stimulus. The times of all taps and swipes were logged, along with any incorrect selections. The different apps were shown as stimuli between 1 and 25 times, depending on their usage probability. We used a smoothed version of the distribution used to train the optimiser, in order to increase the number of applications with more than 1 selection.

Participants performed the experiment while seated in a quiet room. They were asked to hold the phone in their non-dominant hand and tap with the index finger of their dominant hand. Order of conditions was counterbalanced.

After this, participants were shown images of the full layouts and asked to rate them from 1 to 5 in terms of overall aesthetic quality, use of colour, composition logic, and symmetry. They were also asked to indicate a preferred layout.

Results

As our principal performance metric, we use the average selection time over all trials, measured from the tap to dismiss the stimulus to the tap on the correct icon. We filtered out trials with selection errors or hint requests.

Figure 5 shows our results and the first page of each of the designs. Over all trials, there is no significant difference be-

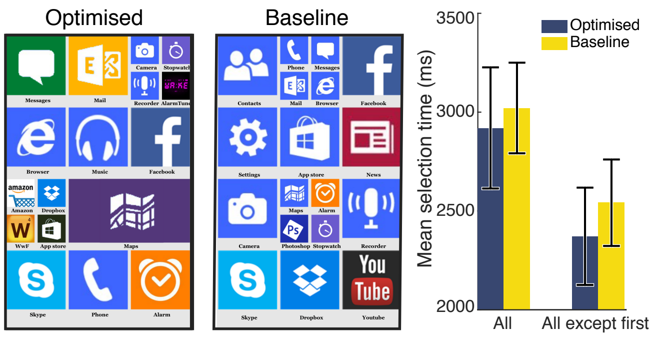


Figure 5: *End-User Study.* Left: Homescreens of the optimised and baseline designs. Right: Average selection times (95% CIs). The optimised design was significantly faster when the first selection for each app was excluded.

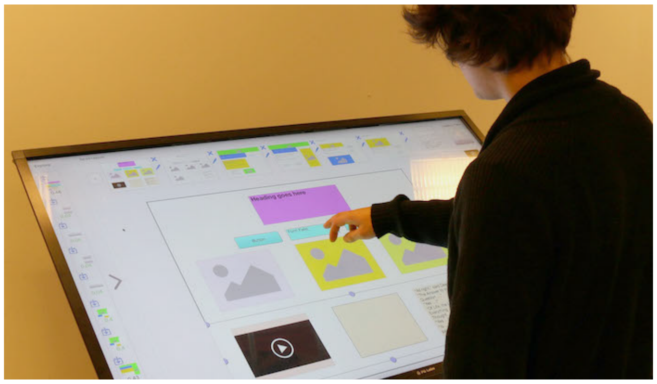


Figure 6: *Designer Study.* We evaluated Sketchplorer with 10 trained designers. Participants completed design tasks while using the different features supported by the design tool.

tween the average selection times between the two menus (paired t-test, $t(19) = -0.95, p > 0.05$).

However, when the first selection for each app in each condition is removed from consideration, the difference in selection time between the menus is significant (paired t-test, $t(19) = -2.67, p < 0.05$). This removal is justified given that users were mostly familiarising with the menus during the first selections. We also analysed only the latter half of trials for each app, to see if the learning benefit continued to grow. However, the results when doing this were very similar. This suggests that the majority of the learning takes place on the first selection. This result matches with our models that predict expert performance rather than that of novices.

For the aesthetic ratings, we found no significant differences between the overall ratings and logic of composition scores across the designs (Wilcoxon signed rank test, $Z = -1.80, -1.79, p > 0.05$). However, users rated the use of colour in the optimised design significantly higher than the baseline ($Z = 2.38, p < 0.05$), but the level of symmetry significantly lower ($Z = -2.65, p < 0.05$).

Half (10) of the 20 users expressed a preference for the optimised design over the baseline. Several participants noted that they disliked one or both designs because the menus did not prioritise the apps they themselves used. This reflects the app usage dataset, which is several years old.

Summary

To summarise, the optimiser was able to produce a significant performance benefit over a realistic baseline design. The observed differences were largely predicted by our models.

STUDY 2: DESIGN STUDY WITH A LIVE SYSTEM

To evaluate *Sketchplorer* as an integrated tool, we conducted a design study with trained designers. We aimed to evaluate whether Sketchplorer is effective in supporting creativity and problem-solving in sketching. We were interested in designers' insight and impressions, and not on validation of produced designs, as this had been investigated in the previous study. We did not direct them to use the optimiser's suggestions but were interested in seeing if this would happen spontaneously.

Study Design

We recruited a total of 10 participants (2 female), age ranging from 22 to 40 years (mean 29.4). All participants, except one, had an educational or professional background in design, and all of them had some experience using digital design tools. Participants were compensated with two cinema tickets.

All design tasks were performed on a 55-inch (140 cm) 4K display (3840 × 2160), with a PQ Labs G5S multitouch overlay. The display was tilted to a comfortable angle, and participants performed tasks in standing position (Figure 6). The design tool ran on a Macbook Pro (OS X 10.10). In addition, two Macbook Pros were used for the optimisers.

Procedure: After initial training, in which participants created simple designs and explored the tool, they were given a 'design brief' for the main study task. Similar to the example in the walkthrough, they were asked to create designs for a blog page. There were no strict requirements, and participants could freely decide on the exact elements, and their details. They were asked to create a few different designs, and could choose to make as many as they wished to, in a time frame of 30 minutes. Participants were free to save designs that were sketched with or without the aid of the optimiser. At the end of the task, they assigned ratings (1--5) to their saved designs, using a custom 'viewer' application. Importantly, the viewer did not reveal whether a given design had been entirely sketched by the participant, or if it was optimiser-assisted. Finally, we gathered further information through a questionnaire and a semi-structured interview.

Results

The study aimed at gauging whether Sketchplorer allowed designers to sketch freely, and their usage of different features the tool provided. On reviewing participants' list of saved designs, we found that 8 out of 10 participants had at least one optimiser-aided design in their saved list. Additionally, participant 10 stated to have borrowed an idea from the optimiser

ID	N(Saved)	N(Recolour)	N(Fix)	N(Global)
1	3	0	0	3
2	5	0	0	2
3	5	3	1	0
4	5	0	2	2
5	4	1	0	1
6	3	0	0	0
7	4	0	3	0
8	8	0	0	5
9	10	2	3	4
10	1	0	0	1

N(Saved) : Total number of saved layouts
N(Recolour) : Saved layouts with recolour-suggestions
N(Fix) : Saved layouts with fix-suggestions
N(Global) : Saved layouts with global-suggestions

Table 2: Summary of features used by each participant in Study 2. 9 out of 10 designers took advantage of suggestions offered by the optimiser, to achieve their final designs.

suggestions, but recreated it manually in his own sketch. On average, participants perceived optimiser-assisted designs to be equally good as designs they sketched alone. Table 2 summarises the optimiser-features used by the participants, in the saved versions.

Responses to the questionnaire and interview session were encouraging for Sketchplorer. Apart from minor technical glitches, participants commented that they could sketch out their ideas quickly and freely, and the multitouch interactions were easy and straightforward. The ability to have a large touch-friendly canvas was appreciated. All participants stated that they would certainly prefer a visual timeline in such design tools, over traditional save-dialogs, and such a feature would be useful in future tools.

With regards to the optimiser-related features, participants especially liked the *explore* option. They found the suggestions to be distinctly different, useful, and indicated that it would help them while designing. One commented, *"I kind of like how the suggestions turn my approach upside down, and I can get hints from there"*. There was a split between participants while rating the usefulness of *fix* and *recolour* during the given design task. A few participants found some of the *recolour* suggestions *"so 90s"* or *flashy*, and wished that the suggestions would be more subtle. One participant commented that they would rather explore new and different ideas, than 'fix' existing ones. However, participants also indicated that recolouring and fixing would help them in future design activities.

DISCUSSION

This paper has studied a new concept interactive design optimisation, backed by predictive models, and integrated to a sketching tool. By inferring design tasks implicitly, and by using a combination of exploration and exploitation, a layout optimiser can complement the sketching activities of a designer in real time, and enable her to explore larger design spaces without having to divert their attention to the optimisation process. During sketchploration, the designer–optimiser system is continuously both sketching (or, in optimiser terms, exploiting) and exploring. Crucially, the optimiser is not suggesting just anything for the designer. The predictive models of the optimiser try to "pull" the designer towards usable and aesthetic designs. A designer can reject designs that are unsatisfactory for some reasons that the optimiser may not include in its objective function. This way, the designer and the optimiser can iteratively approach a region of good designs without communicating an objective function. We are not aware of a similar approach in the past.

Results from two studies provide first evidence for the concept. Study 1 exposed end-users to a visual layout generated by the optimiser, comparing it against a commercial baseline design. The results were favourable, the optimised design was significantly better in average selection time, when measured after the first selection of an app. Colour harmony was rated higher, but the layout less harmonic. It is intriguing to note that the obtained data are mostly in alignment with the predictions made by the models.

Study 2 gauged Sketchplorer's ability to provide designers with a sketching environment conducive to sketchploration, and gathered insights about the different features. Most designers in our study added some of the optimisers' designs to their saved list of designs, used the features in their process, and rated the optimiser's outputs high. They in particular appreciated the explore feature. Given that the participants had educational and even professional background in design, we consider this a notable achievement.

Overall, we see sketchploration as a promising concept; one that can lead to not only empowering designers, but also to systematically improving usability and aesthetics, and raising the bar of design.

While our work lays the foundation for sketchploration, it also uncovers many challenges and opens up opportunities for future efforts. First, while this paper covers visual aspects of designs, it does not capture semantics or the dynamicity of interactions that interfaces afford. To further improve exploratory results, integrating these aspects is desirable. Second, our optimisation techniques are highly dependent on well-validated models; further development of accurate predictive models is a key factor to improvements in results. Third, although the optimiser's results were mostly good, we note that its performance was limited to designs with ten or fewer elements. Layout optimisation for HCI is a combinatorially challenging task that cannot be directly solved with standard methods. Finally, it can be beneficial to allow deeper exploration into design spaces, and provide designers with detailed insights of the entire space. In optimisation terms, the optimisation landscape could be visualised to allow more informed choices.

ACKNOWLEDGEMENTS

The project has received funding from the Academy of Finland project COMPUTED and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 637991).
Among others, we would like to thank Andreas Karrenbauer and Gilles Bailly for their suggestions and comments, and Olli Savisaari and Perttu Lähteenlahti for their assistance.

REFERENCES

1. Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. 2008. ILoveSketch: As-natural-as-possible Sketching System for Creating 3D Curve Models. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology (UIST '08)*. ACM, New York, NY, USA, 151–160. DOI: http://dx.doi.org/10.1145/1449715.1449740

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

3. Helen Y. Balinsky. 2006. Evaluating interface aesthetics: measure of symmetry. In *Electronic Imaging 2006*. International Society for Optics and Photonics, 607–608.

4. Helen Y. Balinsky, Anthony J. Wiley, and Matthew C. Roberts. 2009. Aesthetic Measure of Alignment and Regularity. In *Proceedings of the 9th ACM Symposium on Document Engineering (DocEng '09)*. ACM, New York, NY, USA, 56–65. DOI: http://dx.doi.org/10.1145/1600193.1600207

5. Rainer E. Burkhard and J. Offerman. 1977. Entwurf von schreibmaschinentastaturen mittels quadratischer zuordnungsprobleme. *Operations Res* 21 (1977), B121–B132.

6. Bill Buxton. 2007. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

7. Daniel Cohen-Or, Olga Sorkine, Ran Gal, Tommer Leyvand, and Ying-Qing Xu. 2006. Color Harmonization. In *ACM SIGGRAPH 2006 Papers (SIGGRAPH '06)*. ACM, New York, NY, USA, 624–630. DOI: http://dx.doi.org/10.1145/1179352.1141933

8. Nigel Cross. 2004. Expertise in design: an overview. *Design studies* 25, 5 (2004), 427–441.

9. Amine Drira, Henri Pierreval, and Sonia Hajri-Gabouj. 2007. Facility layout problems: A survey. *Annual Reviews in Control* 31, 2 (2007), 255–267.

10. Karim El Batran and Mark D. Dunlop. 2014. Enhancing KLM (Keystroke-level Model) to Fit Touch Screen Mobile Devices. In *Proceedings of the 16th International Conference on Human-computer Interaction with Mobile Devices &#38; Services (MobileHCI '14)*. ACM, New York, NY, USA, 283–286. DOI: http://dx.doi.org/10.1145/2628363.2628385

11. Steven K. Feiner. 1988. A Grid-based Approach to Automating Display Layout. In *Proceedings on Graphics Interface '88*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 192–197. http://dl.acm.org/citation.cfm?id=102313.102339

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

13. Mark D. Gross and Ellen Yi-Luen Do. 1996. Ambiguous Intentions: A Paper-like Interface for Creative Design. In *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology (UIST '96)*. ACM, New York, NY, USA, 183–192. DOI: http://dx.doi.org/10.1145/237091.237119

14. Pierre Hansen and Nenad Mladenović. 2001. Variable neighborhood search: Principles and applications. *European journal of operational research* 130, 3 (2001), 449–467.

15. Marc Hassenzahl. 2008. The Interplay of Beauty, Goodness, and Usability in Interactive Products. *Hum.-Comput. Interact.* 19, 4 (Dec. 2008), 319–349. DOI: http://dx.doi.org/10.1207/s15327051hci1904_2

16. Eric Horvitz. 1999. Principles of Mixed-initiative User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*. ACM, New York, NY, USA, 159–166. DOI: http://dx.doi.org/10.1145/302979.303030

17. Gabe Johnson, Mark D. Gross, Jason Hong, and Ellen Yi-Luen Do. 2009. Computational Support for Sketching in Design: A Review. *Found. Trends Hum.-Comput. Interact.* 2, 1 (Jan. 2009), 1–93. DOI: http://dx.doi.org/10.1561/1100000013

18. Levent Burak Kara, Chris M. D'Eramo, and Kenji Shimada. 2006. Pen-based Styling Design of 3D Geometry Using Concept Sketches and Template Models. In *Proceedings of the 2006 ACM Symposium on Solid and Physical Modeling (SPM '06)*. ACM, New York, NY, USA, 149–160. DOI: http://dx.doi.org/10.1145/1128888.1128909

19. Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. 2014. Kitty: Sketching Dynamic and Interactive Illustrations. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, New York, NY, USA, 395–405. DOI: http://dx.doi.org/10.1145/2642918.2647375

20. Kenneth L. Kelly. 1965. Twenty-two colors of maximum contrast. *Color Engineering* 3, 26 (1965), 26–27.

21. David E. Kieras and Anthony J. Hornof. 2014. Towards Accurate and Practical Predictive Models of Active-vision-based Visual Search. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 3875–3884. DOI: http://dx.doi.org/10.1145/2556288.2557324

22. Ranjitha Kumar, Arvind Satyanarayan, Cesar Torres, Maxine Lim, Salman Ahmad, Scott R. Klemmer, and Jerry O. Talton. 2013. Webzeitgeist: Design Mining the Web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 3083–3092. DOI: http://dx.doi.org/10.1145/2470654.2466420

23. James A. Landay and Brad A. Myers. 1995. Interactive Sketching for the Early Stages of User Interface Design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 43–50. DOI: http://dx.doi.org/10.1145/223904.223910

24. Lissa Light and Peter Anderson. 1993. Designing better keyboards via simulated annealing. (1993).

25. James Lin, Mark W. Newman, Jason I. Hong, and James A. Landay. 2000. DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site Design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '00)*. ACM, New York, NY, USA, 510–517. DOI: http://dx.doi.org/10.1145/332040.332486

26. Simon Lok and Steven Feiner. 2001. A survey of automated layout techniques for information presentations. *Proceedings of SmartGraphics* 2001 (2001).

27. Simon Lok, Steven Feiner, and Gary Ngai. 2004. Evaluation of Visual Balance for Automated Layout. In *Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI '04)*. ACM, New York, NY, USA, 101–108. DOI: http://dx.doi.org/10.1145/964442.964462

28. Scott I. MacKenzie. 1992. Fitts' law as a research and design tool in human-computer interaction. *Human–Computer Interaction* 7, 1 (1992), 91–139.

29. Barbara J. Meier, Anne Morgan Spalter, and David B. Karelitz. 2004. Interactive Color Palette Tools. *IEEE Comput. Graph. Appl.* 24, 3 (May 2004), 64–72. DOI: http://dx.doi.org/10.1109/MCG.2004.1297012

30. Mark W. Newman and James A. Landay. 2000. Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice. In *Proceedings of the 3rd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques (DIS '00)*. ACM, New York, NY, USA, 263–274. DOI: http://dx.doi.org/10.1145/347642.347758

31. Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2014. Learning Layouts for Single-PageGraphic Designs. *Visualization and Computer Graphics, IEEE Transactions on* 20, 8 (2014), 1200–1213.

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

33. Antti Oulasvirta, Anna Reichel, Wenbin Li, Yan Zhang, Myroslav Bachynskyi, Keith Vertanen, and Per Ola Kristensson. 2013. Improving Two-thumb Text Entry on Touchscreen Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 2765–2774. DOI: http://dx.doi.org/10.1145/2470654.2481383

34. Ruth Rosenholtz, Yuanzhen Li, and Lisa Nakano. 2007. Measuring visual clutter. *Journal of vision* 7, 2 (2007), 17.

35. Clayton Shepard, Ahmad Rahmati, Chad Tossell, Lin Zhong, and Phillip Kortum. 2011. LiveLab: Measuring Wireless Networks and Smartphone Users in the Field. *SIGMETRICS Perform. Eval. Rev.* 38, 3 (Jan. 2011), 15–20. DOI: http://dx.doi.org/10.1145/1925019.1925023

36. Surya P. Singh and Renduchintala RK Sharma. 2006. A review of different approaches to the facility layout problems. *The International Journal of Advanced Manufacturing Technology* 30, 5-6 (2006), 425–433.

37. Ivan E. Sutherland. 1963. Sketchpad: A Man-machine Graphical Communication System. In *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference (AFIPS '63 (Spring))*. ACM, New York, NY, USA, 329–346. DOI: http://dx.doi.org/10.1145/1461551.1461591

38. Michael Terry, Elizabeth D. Mynatt, Kumiyo Nakakoji, and Yasuhiro Yamamoto. 2004. Variation in Element and Action: Supporting Simultaneous Development of Alternative Solutions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 711–718. DOI: http://dx.doi.org/10.1145/985692.985782

39. Marc Van Droogenbroeck and Sébastien Piérard. 2011. Object Descriptors Based on a List of Rectangles: Method and Algorithm. In *Proceedings of the 10th International Conference on Mathematical Morphology and Its Applications to Image and Signal Processing (ISMM'11)*. Springer-Verlag, Berlin, Heidelberg, 155–165. http://dl.acm.org/citation.cfm?id=2023043.2023061

40. L.G. Williams. 1966. *A Study of Visual Search Using Eye Movement Recordings*. Technical Report. DTIC Document.

41. Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. 2007. Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*. ACM, New York, NY, USA, 159–168. DOI: http://dx.doi.org/10.1145/1294211.1294238

42. Yin Yin Wong. 1992. Rough and Ready Prototypes: Lessons from Graphic Design. In *Posters and Short Talks of the 1992 SIGCHI Conference on Human Factors in Computing Systems (CHI '92)*. ACM, New York, NY, USA, 83–84. DOI: http://dx.doi.org/10.1145/1125021.1125094

43. Shengxiang Yang and Xin Yao. 2008. Population-based incremental learning with associative memory for dynamic environments. *Evolutionary Computation, IEEE Transactions on* 12, 5 (2008), 542–561.

44. Yeonsoo Yang and Scott R. Klemmer. 2009. Aesthetics Matter: Leveraging Design Heuristics to Synthesize Visually Satisfying Handheld Interfaces. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems (CHI EA '09)*. ACM, New York, NY, USA, 4183–4188. DOI: http://dx.doi.org/10.1145/1520340.1520637

45. Shumin Zhai, Michael Hunter, and Barton A. Smith. 2002. Performance Optimization of Virtual Keyboards. *Human–Computer Interaction* 17, 2-3 (2002), 229–269.