Grid-based Genetic Operators for Graphical Layout Generation

MORTEZA SHIRIPOUR, Aalto University, Finland
NIRAJ RAMESH DAYAMA, Aalto University, Finland
ANTTI OULASVIRTA, Aalto University, Finland

Graphical user interfaces (GUIs) have gained primacy among the means of interacting with computing systems, thanks to the way they leverage human perceptual and motor capabilities. However, the design of GUIs has mostly been a manual activity. To design a GUI, the designer must select its visual, spatial, textual, and interaction properties such that the combination strikes a balance among the relevant human factors. While emerging computational-design techniques have addressed some problems related to grid layouts, no general approach has been proposed that can also produce good and complete results covering color-related decisions and other nonlinear design objectives. Evolutionary algorithms are promising and demonstrate good handling of similar problems in other conditions, genetic operators, depending on how they are designed. But even these approaches struggle with elements' overlap and hence produce too many infeasible candidate solutions. This paper presents a new approach based on grid-based genetic operators demonstrated in a non-dominated sorting genetic algorithm (NSGA-III) setting. The operators use grid lines for element positions in a novel manner to satisfy overlap-related constraints and intrinsically improve the alignment of elements. This approach can be used for crossovers and mutations. Its core benefit is that all the solutions generated satisfy the no-overlap requirement and represent well-formed layouts. The new operators permit using genetic algorithms for increasingly realistic task instances, responding to more design objectives than could be considered before. Specifically, we address grid quality, alignment, selection time, clutter minimization, saliency control, color harmony, and grouping of elements.

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

Additional Key Words and Phrases: Grid-based genetic operators, Many-objective optimization, Graphical layout problem, User interfaces, Gestalt law, Fitts' law

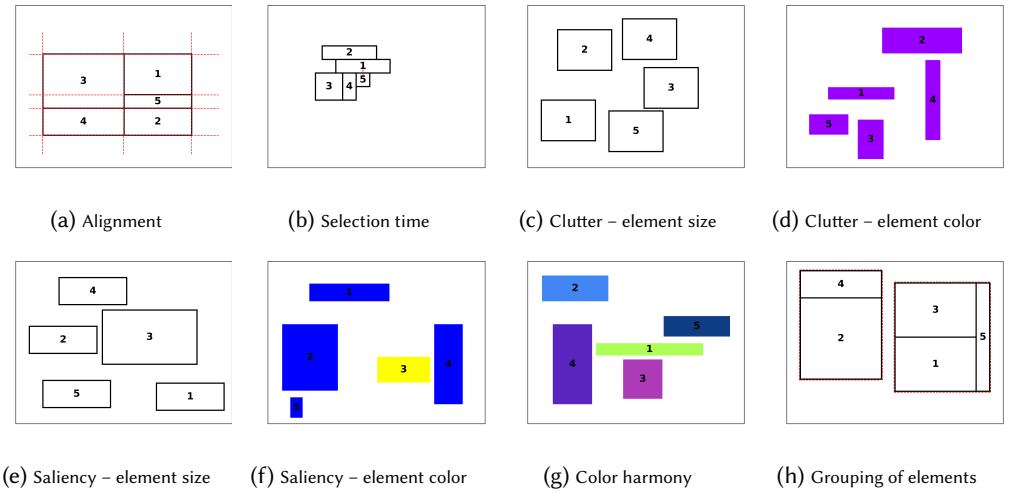
ACM Reference Format:
Morteza Shiripour, Niraj Ramesh Dayama, and Antti Oulasvirta. 2021. Grid-based Genetic Operators for Graphical Layout Generation. *Proc. ACM Hum.-Comput. Interact.* 5, EICS, Article 208 (June 2021), 30 pages. https://doi.org/10.1145/3461730

1 INTRODUCTION

The graphical user interface, or GUI, has become the most prevalent user interface for computing systems. The process of designing GUIs remains a time-consuming, challenging, and mostly manual activity. The difficulty arises from the large design spaces and from the presence of multiple objectives [55, 62]. This large design space is due to the combinatorial complexity of problem. For example, consider a canvas with 1024×768 pixels, then divide the canvas into 32×24 pixels.

Authors' addresses: Morteza Shiripour, shiripour.morteza@aalto.fi, Aalto University, PO Box 15400, Espoo, Finland; Niraj Ramesh Dayama, Aalto University, PO Box 15400, Espoo, Finland, niraj.dayama@aalto.fi; Antti Oulasvirta, Aalto University, PO Box 15400, Espoo, Finland, antti.oulasvirta@aalto.fi.

2573-0142/2021/6-ART208 $15.00
https://doi.org/10.1145/3461730



(a) Alignment (b) Selection time (c) Clutter – element size (d) Clutter – element color

(e) Saliency – element size (f) Saliency – element color (g) Color harmony (h) Grouping of elements

Fig. 1. An illustration of design objectives in GUI design. Optimizing for any one of these separately does not yield good results. Our work permits more efficient use of genetic algorithms to produce GUI designs for any combination of such nonlinear and linear objectives.

Consequently, there are about 2.6e+14 different combinations of positions for only 5 elements. If other features of the elements, such as size, color and grouping are required, the number of different solutions will explode. Computational methods could assist designers with the creative process by producing designs or sketches automatically. While combinatorial optimization offers a natural representation for the design problem, results have been limited to well-isolated parts of the general problem, such as widget layouts, keyboards and keypads, grid layouts, and tiled information displays [72]. What is missing is an optimization method that would allow dealing with the complexity of the large design space, multiple objective functions, and with good results.

While GUI problem is related to known problems in operations research and optimization, it also has some unique nuances. The closest comparable example is the facility layout problems (FLPs) [83]; here, the choices of facility sizes and positions optimize certain objective functions (e.g., for minimization of material-handling costs) [42]. The facilities' coloring is not among the decision variables and objectives of FLP; however, in the domain of GUI design, colors play a crucial role [19, 21, 91]. They emphasize important information, guide attention, influence visual appeal, and help users identify subgroups [65]. Colors are typically computed with nonlinear objectives, and their values can only be determined in the context of other colors and mutual distances on a display (e.g., in terms of saliency and clutter objectives).

In this paper, the graphical layout problem is addressed as a matter of 1) organizing graphical elements on a canvas – including choosing their positions and sizes – and 2) coloring them in a visually appealing, use-supporting manner. Consider a news page online. The design task consists of placing given elements – such as images, videos, headlines, text content, navigation support, and advertisements – on a canvas and deciding on their sizes. Also, these elements should be *grouped* into larger entities, to afford perceptual understanding of the page's structure. Finally, their colors and possibly other visual properties should be chosen. We approach this as a many-objective optimization problem wherein the objective functions may be anything that assigns a numerical score to a design (the literature provides a good review of objective functions, which range from linear functions to nonlinear ones and simulator-based models [72]). Flexible "plug and play" for

objectives is crucial to practical deployment, wherein designers may want to adjust the emphasis among objectives case-specifically. Formulated thus, our problem is an extension of the general FLP. Our approach can be used for a variety of audiences. For example, it could help interaction designers to explore more solutions when sketching and release resources from lower-level editing [20]. It could also help end-users by personalizing or customizing UI layouts [34, 73, 96]. Here, we consider the eight design criteria in Figure 1, an example beyond the reach of previous work. This paper focuses on static user interfaces (UIs). Dynamically changing UIs are out of the scope of the paper.

Because multiple nonlinear objectives are involved, nature-inspired algorithms [94], especially genetic algorithms, offer a promising approach. They can solve many-objective problems, and, in contrast to some classical optimization methods [28], all objectives are evaluated simultaneously. This helps the designer identify not merely a single optimal design but several on the Pareto front and, thereby, make choices that better consider the tradeoffs involved.

Notwithstanding the success of evolutionary algorithms for similar problems elsewhere, **genetic operators** (including mutation and crossover operators) struggle with overlap of elements so yield too many infeasible candidate solutions. Our work extends genetic operators, via a grid-based concept, to deal with graphical layouts more efficiently. These operators render a non-dominated sorting genetic algorithm (NSGA-III) [25] more efficient, to obtain approximated Pareto-front solutions. Though the grid operators we propose are useful more generally, we picked NSGA-III for its value as a known, powerful evolutionary algorithm for many-objective problems [12, 59, 78]. It has demonstrated its efficiency in tackling diverse problems: economical X-bar control chart design [87], hydro-thermal-wind scheduling [95], software refactoring [68], feature selection for intrusion-detection systems [99], and others.

This paper presents **grid-based operators** to facilitate the use of genetic algorithms, particularly with regard to the overlap constraint. Generating non-overlapping layouts helps the optimizer not waste computation time on identifying which solutions satisfy this constraint. As we show, this yields a significant boost to their performance that is of great practical value.

Our concept of grid operators is based on the idea of **alignment**. One of the main objectives in GUI design is to align elements along grid lines. These lines are used in design to form a layout, create balance, organize the elements, and keep the layout flexible for future changes [63]. Our grid-based operators offer three benefits. Primarily, the rest of the algorithm can work as it is; thanks to the grid approach, it need not consider these constraints. Next, as we show, they expedites convergence. Lastly, they automatically reduce unnecessary white space among elements, so the resulting layouts look more compact and appealing.

In summary, our main contribution lies in grid-based operators that significantly improve the efficiency of genetic algorithms (NSGA-III in our case) in tackling the GUI design problem. This improves the general crossover and mutation operators by means of the concept of grid lines. We demonstrate the approach with design problems that encompass more objective functions than grappled with previously.

The discussion proceeds as follows: Four main categories of literature are identified and described, in Section 2, after which Section 3 expresses the problem definition and the objective functions. Our proposed grid operators and a methodology overview are presented in Section 4; then, Section 5 discusses the computational performance of the algorithm and illustrates a concrete application problem. The paper concludes with a summary of contributions and suggestions for future work.

2 RELATED WORK

We discuss the literature related to GUI design in terms of four classes: computational design of graphical user interfaces, related layout problems (facility layouts especially), many-objective optimization methods, and operators in genetic algorithms.

2.1 Computational Design of GUIs

Existing literature on computational design of GUIs covers two major approaches. Firstly, interaction techniques have been proposed to facilitate editing of layouts in design tools. Bier and Stone [8] suggested snap-dragging, which places GUI elements on grid lines automatically with the aid of a ruler-and-compass heuristic while designing the global layout remains the designer's task. Imagine [69] is an interactive GA tool for generating the style of HTML sheets. Gajos and Weld [34] used a branch-and-bound (B&B) optimization technique to generate personalized a single layout for the end-user via their SUPPLE system. Jiang et al. [48] decreased the computational time of B&B algorithm with heuristic preprocessing. Swearngin et al. [86] suggested an exploration algorithm with high-level design constraints which helps designers to quickly find alternative designs. Frisch et al. [33] presented interactive grids and multi-touch alignment guides, and Xu et al. [92] described an alignment method for decreasing the ambiguity of elements. Masson et al. [66] proposed an evolutionary system to improve creativity designs, and Yanagida et al. [93] used a flexible widget layout to personalize GUIs. Recently, neural networks have been applied, such as content-aware deep generative model (Zheng et al. [97]), generative adversarial networks (Li et al. [60, 61]), and neural design networks (Lee et al. [58]), to generate layouts.

In the context of menu layout design, Bouzit et al. [9] employed Bertin's eight visual variables [7] to interactively help the designer to explore the design space via choosing different values for these visual variables. Goubko and Danilenko [37] presented a mathematical hierarchical menu optimization based on the navigation time. Then, Goubko and Varnavsky [36] suggested user's preference share instead of navigation time criterion to optimize their menu design. These approaches address only part of the full layout-problem space.

Algorithmic methods, in turn, produce full layouts through application of grids. All papers but one have ignored coloring, however. A recent review of combinatorial-optimization-based methods for GUI design does not cite a single approach that proves able to deal with objectives that designers typically need to address in GUI design – including layout and coloring [72]. Recently, Dayama et al. [23] proposed a method based on mixed integer programming. Their model considers preferential positioning of elements and uses a space-spanning [89] approach to find diverse solutions. The lack of a comprehensive optimization system implies that interactive methods are needed, to add those decisions not covered. One exception is found in the work of Todi et al. [88], who suggested an interactive optimizer for a "sketch-and-explore" approach. Their method, using a multi-touch sketching tool, can generate layouts in real time by using five predictive models and a variable-neighborhood-search-based optimizer. From a designer-sketched wireframe, the tool computes alternative layouts to aid in the ideation process. However, the authors acknowledged significant computational issues with larger layout sizes, as the number of elements grows beyond 10. To the best of our knowledge, the literature has not identified any computational method for modeling this problem whereby multiple graphical layouts are automatically generated for a set of several objectives that includes color.

2.2 Related Layout Problems

The most relevant problem in the mathematical-optimization field is the unequal-area facility layout problem (UA-FLP). The UA-FLP was first formulated as a quadratic assignment problem (QAP) [57]

wherein the goal is to minimize the total cost of material-handling [4]. It is an NP-hard problem [35], and exact algorithms can find the optimal design for only small and medium-sized instances in reasonable time [74]. For example, Castillo et al. [10] proposed a cutting-plane method to solve a one-objective UA-FLP for up to nine facilities. In another study, Chae and Regan [11] reached optimal values for up to 12 facilities. Several heuristic and meta-heuristic approaches have been presented [50, 56, 76]. A recent survey by Hosseini-Nasab et al. [42] provides further details of these.

The GUI design problem is distinct from the UA-FLP in three main respects: Adding coloring requires adding nonlinear objectives – e.g., optimizing for color harmony and saliency. Secondly, aligning each element with all others is different from deciding on element locations one by one, in a local neighborhood, or pairwise. Finally, the GUI problem is characteristically a many-objective problem.

2.3 Approaches to Many-Objective Problems

In most real-world design problems, the decision-making involves more than one objective, and, rather than one optimal solution, there is a set of optimal solutions: the Pareto front. In other words, improving the value for one objective leads to worse results in relation to at least one other objective. We discuss two general approaches to problems with up to three objectives (multi-objective models): classical and evolutionary [18]. Among the former, weighted-sum and ϵ-constraint methods are some of the most popular [28]. Methods in this class have four shortcomings: 1) multiple runs are required for finding the Pareto front; 2) in cases of non-convex problems, one cannot obtain the full Pareto front; 3) these methods are sensitive to the Pareto front's shape (e.g., concave or disconnected); and 4) nonlinear problems require linear approximation techniques [53]. Since the problem discussed in this paper is nonlinear and many-objective, classical methods are not ideal.

Heuristic and meta-heuristic algorithms (e.g., evolutionary methods) emerged as an alternative to classical algorithms, for overcoming these disadvantages [98]. They are easier to implement, have broad applicability, and can be robust to dynamic changes [30]. For multi-objective problems, often-successful algorithms such as multi-objective particle swarm optimization [79], multiple-objective ant colony optimization [2], and non-dominated sorting genetic algorithm II [26] have been suggested. However, they are recognized as scaling up poorly: as the number of objectives rises, the efficiency of these algorithms decreases, and they cannot find a diverse set of Pareto-front solutions [29, 45]. Therefore, a new generation of algorithms has been presented for problems with more than three objectives, known as **many-objective** problems [59].

This new generation of algorithms is capable of approximating the diversity of Pareto front solutions through introducing new approaches such as relaxed domination, diversity approach, or reference-point approach [59]. The interested reader is referred to a recent review [24] for detailed description. In this paper, we address eight distinct objectives (discussed in Section 3) and propose an extension to genetic algorithms for better handling of many-objective cases for layout problems. We have implemented our grid operators for the **reference-point algorithm** NSGA-III [25] (see Section 4). As far as we are aware, only **multi**-objective problems have previously been studied for FLPs.

2.4 Operators for Genetic Algorithms

Crossover and mutation are two kinds of operator applied to generate new solutions for genetic algorithms (GAs). Their main role is exploration/exploitation of the search space. In each generation, they are applied to find new solutions and identify better nearby solutions [44]. Scholars have suggested several crossover and mutation operators, depending on the application [1, 40, 51, 52, 71, 85]. In most cases, these operators are heuristics and their performance is restricted to specific

applications. For example, Albayrak and Allahverdi [1] developed a mutation operator that they named "Greedy Sub Tour Mutation" (GSTM) for the traveling salesman problem. Relative to simple GA operators, their GSTM yields error-value reductions between 59.42% and 79.51%. Swarup and Yamashiro [85] discussed the unit commitment problem and proposed specific operators to deal with its time-dependency constraints. In practical use, such modification steps are required if one is to improve the efficiency of GA operators in generating better solutions [39].

In response to general GA operators' inefficiency in relation to the overlap constraint, we propose a grid-based modification to them. To deal with this issue, our approach forms grid lines at the beginning and end of the parent's elements. The grid lines are used for finding the empty slots on the canvas, and the operator then moves/adjusts the position/size of the swapped genes to occupy one of these slots. Hence, our grid-based operators satisfy the constraint of non-overlap. Subsection 4.2 provides detail-level discussion of the steps followed.

2.5 Summary of Connections with Prior Work

Our work breaks new ground in three respects. Firstly, it extends research on layout problems from facility layouts to graphical layouts, a domain distinguished by the number and nonlinearity of the objective functions, its coloring decisions, and the grid structure. Secondly, ours appears to be the first attempt at a formal definition of the many-objective graphical layout problem. Finally, the novel grid operators we propose permit more efficient solutions from genetic algorithms.

3 PROBLEM DEFINITION AND OBJECTIVE FUNCTIONS

This section provides a mathematical definition 1) of the decision problem and 2) of sufficient objectives for computational generation of full graphical layouts. The per-objective results in Figure 1 show that optimizing for a single objective function does not yield a high-quality layout. Finding a desirable tradeoff among the various objectives is an essential aspect of design. Therefore, we cast the task as a many-objective optimization problem. Further on, we present a GA-based method for approximating the Pareto front, in Section 4.

Verbal description of the decision variables lays the ground for their mathematical formulation. The problem comprises three sets of decision variables, for the position, size, and color of each element. Position articulates the beginning location of the rectangular elements on the canvas, with x_i and y_i representing the starting distance of element i from the given x-coordinate and y-coordinate, respectively. Size refers to the width and height of each element, honoring its specific size limitations. Its decision variables for element i are w_i and l_i. Finally, three decision variables are used to find a color for each graphical element. This is considered in the *RGB* color space, with decision variables of R_i for element i's red channel, G_i for green, and B_i for blue. Let us take a given set of rectangular graphical elements N and a task of placing each element $i \in N$ on a fixed 2D canvas such that there is no overlap among elements, overflow, or empty space.

Using these decision variables, we can formulate our objective functions. A multi-objective optimization problem can be formulated generally as

$$\begin{aligned} \min \quad & \boldsymbol{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \cdots, f_M(\boldsymbol{x})) \\ \text{s.t.} \quad & \boldsymbol{x} \in \Omega, \end{aligned} \tag{1}$$

where M is the number of objectives, x is the feasible set of decision variables, and Ω is the set of all boundaries and constraints.

The eight objectives considered here cover fundamental aspects of visual attention (saliency), perceptual fluency (grid lines and clutter), motor control (selection time), and visual appeal (color harmony). Obviously, these objectives, discussed in the literature [72], are not the only ones possible;

we used them to test the efficiency of grid-based operators and demonstrate that they yield a high-quality layout without additional human input. Therefore, the designer can consider this approach to be a plug-and-play framework – new objectives can be added and old ones removed as the case dictates. Also, our approach is compatible with various screen resolutions.

Below, we will use a running example to illustrate the objectives. The example Web page has a canvas size of 800×600 and rectangular elements with dimensions of 50 to 600 pixels (see Table 9 in the appendix).

3.1 Grid Lines and Alignment

The term "grid quality" refers to the internal alignment of elements with each other. Several studies have discussed this metric [27, 33, 43]. Our method uses the start and the end position of each element to find the number of grid lines, so the sum of unique horizontal and vertical lines gives the total number of alignment lines. This objective can be formulated as

$$\min \sum_i U(v_i) + U(h_i), \tag{2}$$

where vertex locations v_i and h_i indicate the positions of the vertical and horizontal lines associated with element i, respectively. Note that each element has two vertical line positions and two horizontal line positions, and their positions could overlap with the beginning or end of other elements. Then, the function U returns the number of unique values for each vector. For this objective, the total number of these unique lines must be minimized.

The optimal result for this objective, given the element-size limitations shown in Table 9, is displayed in Figure 1a. This pane shows three vertical lines and four horizontal lines, which makes the value for this objective 7, with all elements perfectly aligned and within the set size limits.

3.2 Selection Time

The objective denoted as "selection time" describes the pairwise relationship between elements. The main aim is to reduce the time for moving the cursor point from the current element to the target one. Thus, the user finds related elements more easily. Hence, the transition time is calculated via Fitts' law [64], according to which the time for moving between two elements' center points can be computed via the formulation

$$T = a + b \log_2 \left(\frac{D}{W} + 1 \right), \tag{3}$$

where T is the transition time, W is the width of the target element, and D is the distance between the center of the current and the target element. In addition, a and b are empirical parameters connected with the input device. Other given data come from a pairwise matrix for the frequency of moving between elements i and j, or f_{ij}. There are three possible sources for the pairwise frequency matrix: 1) expert (designer) opinion, 2) theory, 3) data. Accordingly, this objective is calculated thus:

$$\min \sum_i \sum_j T_{ij} f_{ij} \tag{4}$$

We wish to minimize this objective. For purposes of our instance, there is assumed to be a strong relationship between elements 1 and 5. Figure 1b shows the model's result for this, in which the elements are in their minimum size ratios for purposes of decreasing the distance between them. This is in line with the goal of minimizing selection time.

3.3 Visual Clutter

"Visual clutter" refers to how confusing a display is. Intuition indicates that the more cluttered a display, the more difficult it is to place a new element there that would catch a user's attention [82]. Therefore, this value should be minimized. Good use of element size and color can be regarded as two important means of reducing it. We use a Euclidean metric (distance) for the relevant functions' calculations.

3.3.1 Size clutter. We apply the following formulation for minimizing visual clutter in terms of size:

$$\min \sum_i \sum_j \Delta E_{ij}^w + \Delta E_{ij}^l \tag{5}$$

ΔE_{ij}^w and ΔE_{ij}^l are the Euclidean width and height calculation, respectively, for elements i and j. Since this function is designed to minimize differences in the element sizes, we would expect to see similarly sized elements in the optimal layout, as indeed seen in Figure 1c (it should be noted also that element positions are not important with respect to this objective).

3.3.2 Color clutter. For matching RGB values to human perception, some modification to the color channel weights is necessary, to make it smooth [13]. This gives us

$$\begin{aligned} \hat{r}_{ij} &= \frac{C_{i,R} + C_{j,R}}{2} \\ \Delta C_{ij} &= \sqrt{(2 + \frac{\hat{r}_{ij}}{256}) * \Delta R_{ij}^2 + 4 * \Delta G_{ij}^2 + (2 + \frac{255 - \hat{r}_{ij}}{256}) * \Delta B_{ij}^2}, \end{aligned} \tag{6}$$

where $C_{i,R}$ and $C_{j,R}$ are the red-channel values for element i and element j, and their average is $\hat{r}_{ij}$. The between-element differences for the red, green, and blue channel are calculated as ΔR_{ij}, ΔG_{ij}, and ΔB_{ij}, respectively. Finally, ΔC_{ij} is obtained as the color distance between i and j. Therefore, visual clutter for colors can be obtained as

$$\min \sum_i \sum_j \Delta C_{ij}. \tag{7}$$

This function minimizes the difference for all colors. The result of applying this objective is displayed in Figure 1d; the elements have the same color in the optimal condition. When there are some image or video elements, these elements should not be included in the calculations due to their variety of colors. Note that the elements' position and size are not important with regard to the function for color-related visual clutter.

3.4 Visual Saliency

Saliency is defined as the visual conspicuity of areas in the interface [46]. Salient areas are those more likely to catch the attention of the user. Considering this measurement is useful for making the key elements more striking.

As in the case of visual clutter, we use the Euclidean metric. However, there are two further considerations for these calculations: 1) the relative visual saliency between each salient element and non-salient elements and 2) the saliency relation between all salient elements. It is clear that the value obtained should be maximized if the salient elements are to be readily distinguished. We use s for the salient elements and n for non-salient elements, and w is the weight for importance.

3.4.1 Element size for saliency. We use the following formula for connecting visual saliency with size:

$$w \sum_{s} \sum_{j \in n} \left(\Delta E^{w}_{sj} + \Delta E^{l}_{sj} \right) + (1 - w) \sum_{s} \sum_{j \in s} \left(\Delta E^{w}_{sj} + \Delta E^{l}_{sj} \right) \tag{8}$$

To draw the user's attention to the important elements, the value for this objective function should be maximized. The result for this objective is shown in Figure 1e. For our instance, we defined element 3 as a salient element, and that element indeed differs in size from all other elements in this pane. Obviously, conflicts will arise between this function and other objectives, including those related to alignment, selection time, and visual clutter.

3.4.2 Element color for saliency. In a parallel with the section above, we consider visual saliency as represented by color, both that shared by all salient elements and color differences between salient and non-salient elements. For visual saliency indicated by color, we have

$$w \sum_{s} \sum_{j \in n} \Delta C_{sj} + (1 - w) \sum_{s} \sum_{j \in s} \Delta C_{sj}. \tag{9}$$

The result for connecting colors with visual saliency is displayed in Figure 1f. A unique color is used to denote element 3 as salient, rendering this element easily detectable. Since only colors are important under this objective, the elements may vary in size within the ranges specified for them.

3.5 Color Harmony

Color harmony is defined as combining colors in a manner that is aesthetically pleasing for human visual perception: selecting colors with positions in the color space that display harmonic relationships. Though there is no definitive way to calculate this, Cohen-Or et al. [17] introduced one way of automatically identifying harmonic colors. They presented eight distinct harmonic templates, which refer to particular sectors of the hue wheel. When all colors in the set are within the wedges indicated, the combination of colors is harmonious. The colors' distance from these sectors measures how far one is from a harmonic color template, so the aim is to reduce this distance as much as possible. Distance $D(.)$ from each template T can be formulated as

$$\min D(T) = \sum_{c} DH(c, T) S(c), \tag{10}$$

where the arc-length distance from color c to a hue sector in template T is denoted as $DH(c, T)$ and the saturation value for c is $S(c)$. If all the colors fall within the wedges identified in the template, this value will be 0. The result produced for our instance, presented in Figure 1g, demonstrates harmony under a "Y type"[17] template.

3.6 Element Grouping

Visually grouping elements aids in understanding larger structures within a scene [55]. In psychology, Gestalt theory asserts that the whole of anything is more important than its parts, and Gestalt laws set forth principles of grouping accordingly [38]. According to Rock and Palmer [81], there are eight Gestalt laws, including similarity, closure, symmetry, and continuity. Here we use the law of proximity for calculating the visual perception of grouping. According to this law, elements that are clustered near each other are perceived as a unified object. Therefore, we have the following mathematical objective:

$$\min \sum_{g} \frac{C_g}{A_g} \tag{11}$$

Here, g is the group index, A_g is the summation area of all elements in group g, and C_g is the convex hull of all elements in that group. The value produced in this function for each group is greater than or equal to 1. We can subtract the total number of groups from function 11 to obtain a value of 0 for the optimal solution; however, it is a constant parameter, so we exclude it from that function. Hence, the minimum value is found when there is no gap area among the elements of each group. For our instance, we defined element 2 and element 4 to be in the same group and the other elements to be in a separate group. As Figure 1h shows, the result grouped the elements in accordance with visual perception theory. However, for example, the alignment and selection-time objectives are not fully satisfied.

4 OVERVIEW OF APPROACH

We propose grid-based operators for NSGA-III [25], which is an improved version of NSGA-II [26] that takes GAs [41] further and suits many-objective optimization problems. Deb and Jain [25] suggested changing the selection step in NSGA-II for three main reasons: to 1) address the existence of multiple optimal solutions, 2) increase the number of non-dominated solutions, and 3) tackle the difficulty of approximating the Pareto front. They proposed using reference points in NSGA-III.

In overview, NSGA-III comprises five main steps. The first three are chromosome representation, the initial population, and GA operators. The next is non-dominated sorting, based on defining several layers of approximate Pareto solutions. Finally, the replacement step is used to select the best solutions in the final layers to be the next generation. Crowding distance was introduced in NSGA-II, but this strategy does not assist with many-objective problems, since it brings more non-dominated solutions, difficulty in approximating the Pareto front, and high computational cost [5].

In the suggested strategy of using a reference-point selection mechanism for NSGA-III, the algorithm spreads several reference points uniformly over a hyper-plane, then evaluates new solutions' closeness to these reference points and selects the nearest ones. This strategy is intended to distribute the solutions in the feasible space. Figure 2 presents a flowchart of this algorithm.

Since the model presented in this paper entails some constraints, a constraint-handling strategy is necessary. Our constraint-handling is based on a method described by Jain and Deb [47] wherein the main idea is to compare penalty values between solutions: the one with the smaller penalty prevails over the other solution. In cases wherein neither brings a penalty, a non-dominate sorting step is applied. With our constraints, penalties arise from any of three violations: 1) at least part of one element is outside the canvas (violating the boundary constraint can be rectified simply by defining the ultimate possible start positions of the elements on the canvas – the relevant canvas dimension minus the minimum width and height of each element); 2) elements overlap, which the proposed grid-based operators do not permit; or 3) there is empty space between elements. The last is the only violation that the penalty approach described must handle.

As discussed in Section 1 and 2, NSGA-III has been studied successfully for a wide range of applications. We empower it with greater efficiency in two ways, one of which is related to the initial generation while the other involves modifying GA operators to satisfy the overlap constraint. These improvement strategies are explained in detail in the subsections that follow.

4.1 The Initial Generation

The first step in defining the problem in GA terms is to choose an appropriate chromosome encoding. This mapping represents the search space for the problem. We use the seven decision variables discussed above, and we consider a block of genes with seven rows and n columns, for which n is the total number of elements. The values in the first two rows, which are X and Y positions, are integers between 0 and the width/height of the canvas minus the minimum width/height of

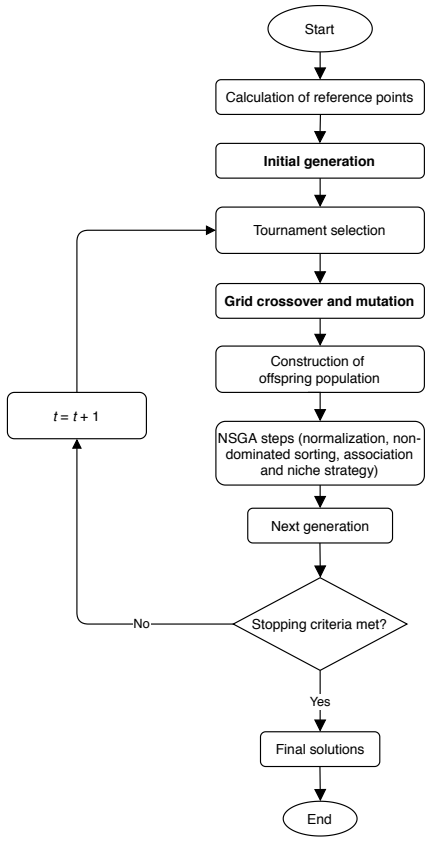


Fig. 2. NSGA-III flowchart covering the initial generation and grid-based operators.

each element, respectively. The next two rows, for element widths and heights, indicate the sizes permitted for each element. Finally, the last three rows are used for color selection in the *RGB* space, with the value in each channel being between 0 and 255, inclusive. Figure 3 offers an example of chromosome representation for five elements.

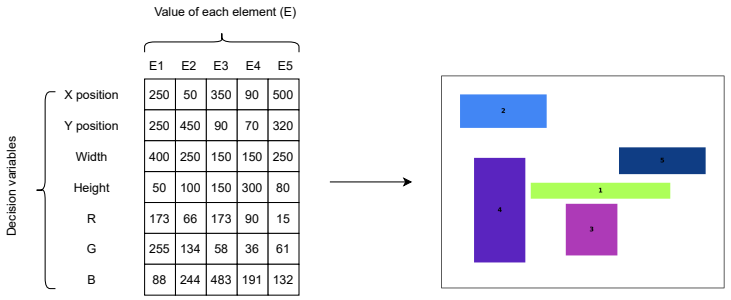


Fig. 3. A sample of chromosome representation for five elements.

After encoding of the problem, the next step is to generate several chromosomes, referred to as the initial population. This population has an impact on the performance of the algorithm [77].

One strategy for generating this population is to rely entirely on random solutions and let the algorithm find its way to better solutions. The other strategy, which we employed, is to generate various, divergent solutions associated with the problem. Several studies [32, 77, 84] have found the second strategy to exhibit better performance.

To obtain a better sense of the inefficiency of generating random initial solutions, we repeatedly generated 200 random initial solutions, performing this procedure 10 times for three distinct data instances (the values of the parameters for these instances are given in the appendix's Table 10). We calculated the percentages of solutions not violating the overlap constraint, and Table 1 presents the averages for these.

Table 1. Random initial solutions satisfying the overlap constraint

Number of elements	**Average percentage of solutions satisfying the overlap constraint**
7	12.3
10	8.6
13	2.9

The averages shown in Table 1 attest that using a random initial generation is not an efficient approach to handling the constraint of non-overlap. For example, only 2.9 percent of the random initial generated solutions are able to meet the overlap constraint. In addition, the main reason why this percentage is higher for the seven elements is only due to the canvas size. Actually, the size of the canvas is considered to be identical across all the data instances, satisfying this constraint with fewer elements is more likely. It should be noted that all solutions satisfy the boundary constraint.

Accordingly, we suggest a new strategy for generating initial solutions that satisfies the overlap constraint. The first step is to generate several random solutions, where only the elements' start position is identified. All these solutions are generated to fulfil the problem's boundary limitations by definition. Then, the following linear formulation determines the width and height of the elements:

$$\min \quad \sum_i w_i + h_i \qquad \text{and} \qquad \max \quad \sum_i w_i + h_i \tag{12}$$

$$\text{s.t.} \quad \mathbb{x}_i + \tfrac{w_i}{2} \le \mathbb{W} \quad \forall i \tag{13}$$

$$\mathbb{y}_i + \tfrac{h_i}{2} \le \mathbb{H} \quad \forall i \tag{14}$$

$$\tfrac{w_i+w_j}{2} - \left|\mathbb{x}_i - \mathbb{x}_j\right| \le \mathbb{M} l_{ij} \quad \forall i, j \tag{15}$$

$$\tfrac{h_i+h_j}{2} - \left|\mathbb{y}_i - \mathbb{y}_j\right| \le \mathbb{M} k_{ij} \quad \forall i, j \tag{16}$$

$$l_{ij} + k_{ij} \le 1 \quad \forall i, j \tag{17}$$

$$\mathbb{w}_i^l \le w_i \le \mathbb{w}_i^u \quad \forall i \tag{18}$$

$$\mathbb{h}_i^l \le h_i \le \mathbb{h}_i^u \quad \forall i \tag{19}$$

$$l_{ij}, k_{ij} \in \{0, 1\} \quad \forall i, j \tag{20}$$

Here, w_i and h_i are the width and the height decision variable, respectively, for element i. Binary variables l_{ij} and k_{ij} determine whether elements i and j overlap in their x- and y-coordinates, respectively. The parameters $\mathbb{x}_i$ and $\mathbb{y}_i$ refer to the center point of element i, obtained via random generation. The values $\mathbb{w}_i^l$ and $\mathbb{w}_i^u$ denote the minimum and maximum width for element i, and corresponding limitations apply for the height of element i: $\mathbb{h}_i^l$ and $\mathbb{h}_i^u$. Finally, $\mathbb{W}$ and $\mathbb{H}$ refer to the

width and height of the canvas, respectively, and M is a sufficiently large constant, which should be bigger than the canvas size in our case.

We solve the above model for two cases. In the first case, function 12 minimizes the elements' width and height (see line 10 of Algorithm 1), and the second case involves maximizing the sizes of the elements (see line 16 of Algorithm 1). These cases are used to generate solutions with various element sizes, and the constraints guarantee fulfilling the canvas's limitations and the condition of non-overlap (see lines 11-15 and lines 17-21 of Algorithm 1). Specifically, constraints 13 and 14 guarantee that all elements are entirely on the canvas, constraints 15–17 do not allow overlapping, and constraints 18 and 19 make sure the width and height of element i are within the set range.

None of the solutions for these two linear models violates our overlap or canvas-border constraints. We repeat all these steps, for different randomly generated starting positions, until all the solutions required for the initial population are obtained. The pseudocode for these steps is shown as Algorithm 1. Obviously, changing the gap value returns solutions with different element sizes (see line 4 of Algorithm 1).

Algorithm 1 Initial-generation steps

1: **Input:** number of initial solutions and gap value
2: N = number of elements
3: F = number of initial solutions
4: G = the gap set for the model (conditions 12–20)
5: S = list of solutions
6: $n = 0$
7: **while** $n < F$ **do**
8: $\mathbb{x}_i$ and $\mathbb{y}_i$ = randomly generate the starting position of the elements, $\forall i \in N$
9: Insert the $\mathbb{x}_i$ and $\mathbb{y}_i$ values for constraints 13–16
10: M_1 = solve the model (conditions 12–20) with minimum objective function and gap G
11: **if** M_1 feasible **then**
12: Return values w_i and h_i for all elements
13: Add w_i, h_i, $\mathbb{x}_i$, and $\mathbb{y}_i$ as a solution to S
14: $n = n + 1$
15: **end if**
16: M_2 = solve the model (conditions 12–20) with maximum objective function and gap G
17: **if** M_2 feasible **then**
18: Return values w_i and h_i for all elements
19: Add w_i, h_i, $\mathbb{x}_i$, and $\mathbb{y}_i$ as a solution to S
20: $n = n + 1$
21: **end if**
22: **end while**
23: **Output:** list of all solutions, S

4.2 The Grid-Based Operators

The two main operators in GAs are crossover and mutation. These operators are used to generate offspring. Their primary roles are to explore and exploit the search space. Therefore, genetic operations should be designed carefully to investigate the search space properly. We developed our grid-based operators accordingly to satisfy the overlap constraint. These operators, discussed in

detail in the two subsections below, can be used for any problem with overlap-related constraints, such as bin-packing [16] and UA-FLPs.

4.2.1 The grid approach to crossover operators. The primary role of crossover is to exploit the search space between two chromosomes (called parents) so as to generate a new chromosome (the offspring or child). The offspring inherits a portion of its genetic makeup from one parent and the rest from the other parent. While the original form of uniform crossover is used in our case, one should remember that applying this crossover in its original form leads to several infeasible solutions. For example, Figure 4 shows a result from its use for five elements (the corresponding data are found in Table 9). Note that Figure 4 shows the result only of considering size limitations; the element colors are chosen purely to clarify the propagation result with the crossover operator, since the color decisions do not violate any constraint and their evaluation is not necessary for this particular example.

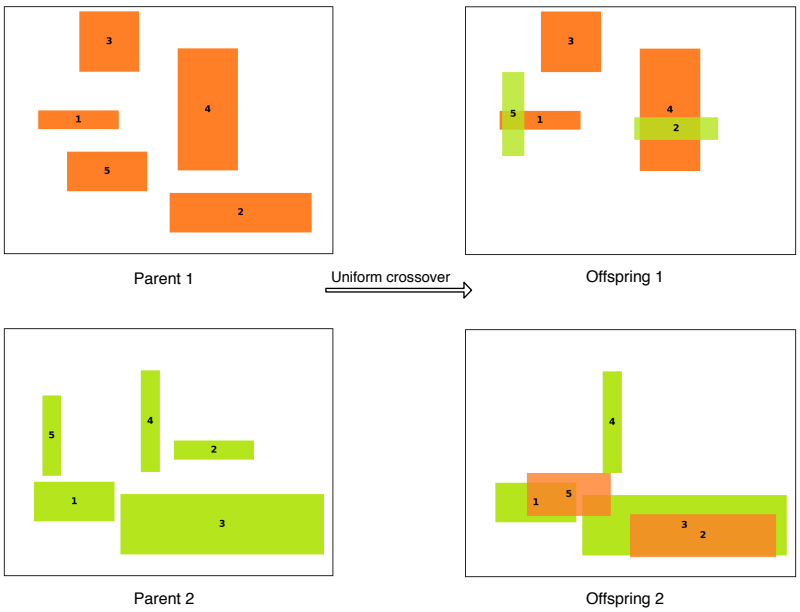


Fig. 4. An instance with the original uniform-crossover operator that violates the overlap constraint.

In Figure 4, there are two parents, each with five elements, and uniform crossover appears for element 2 and element 5. Both offspring exemplify original uniform crossover's violation of the overlap constraint: Two overlaps render offspring 1's solution infeasible, that between elements 1 and 5 and that between elements 2 and 4. Offspring 2, in turn, is infeasible because element 5 overlaps with elements 1 and 3 both and there is overlap between elements 2 and 3.

For concrete evidence, we evaluated the performance of the original uniform-crossover operator for three distinct instances. The results are summarized in Table 2. The algorithm was executed five times, with all parameters taken from Table 10.

Table 2 presents the average percentage of the solutions that satisfied the overlap constraint, for three distinct instances. As the table shows, these figures are quite small. The highest percentage, which is 11.3%, was seen after 100 generations for the seven-element instance. For all instances,

Table 2. The average percentage of overlap-constraint-satisfying solutions among the solutions generated via the original uniform-crossover operator (all solutions generated via the grid-based crossover operator satisfy this constraint)

	Average percentage of solutions satisfying the overlap constraint			
Number of elements	1st generation	10th generation	50th generation	100th generation
7	8.1	8.9	9.6	11.3
10	5.1	5.5	6.4	7.7
13	2.2	2.4	3.1	3.8

the percentage increases with the number of generations. This is because the next generations are generated through better parents, in line with the main principle of GAs. All these small percentages attest to the inefficiency of using the original uniform-crossover operator.

A better strategy might be to generate only feasible solutions or modify the infeasible ones to satisfy the constraint of no overlapping. The crossover strategy we propose follows the grid design. Figure 5 shows how this can process a solution to meet the overlap constraint. It should be noted that the crossover points in this example, used also for Figure 4, are at elements 2 and 5. As the latter figure shows, these two offspring violate the overlap constraint. However, building grid lines with the parts of the parents' genes (see "Grid-based 1" and "Grid-based 2" in Figure 5) produces some feasible empty spaces. Elements 2 and 5 can be assigned randomly to these spaces as their size limitations permit.

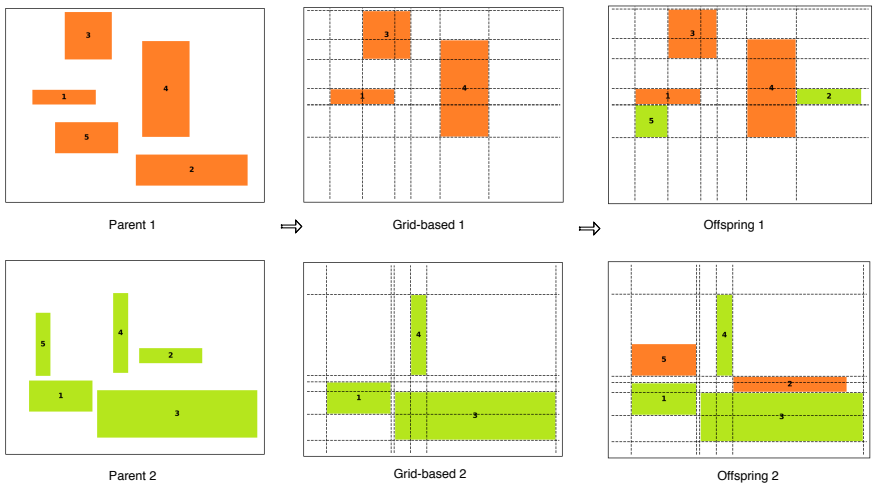


Fig. 5. Use of a grid-based crossover operator.

Algorithm 2 presents all these steps in detail. In general, after selecting two parents (shown in line 1 of Algorithm 2), a uniform-crossover operator randomly selects the crossover elements (shown in line 2 of Algorithm 2). In this stage, we begin by creating the grid lines for the remaining part of the parent's gene (shown in lines 6 and 17 of Algorithm 2). Then, if a gene of the other parent at the crossover point can lie within this set of lines, the algorithm accepts it (shown in lines

7-9 and lines 18-20 of Algorithm 2). Otherwise, it randomly moves the element to one of the empty spaces in the grid (shown in lines 10 and 21 of Algorithm 2). This process continues until all the crossover elements are placed.

Algorithm 2 Grid-based crossover steps

1: **Input:** two parents, P_1 and P_2
2: V= a uniform random binary set to determine the crossover elements
3: L_1, L_2 = an index of elements in V for which the value is 0
4: **for** element i in set V **do**
5: **if** the value of the ith element in set V is 1 **then**
6: G_1 = build grid lines with L_1
7: **if** crossover at element i makes G_1 infeasible **then**
8: $temp_1$ = a set of possible slots within the grid lines G_1 (which are fitted to the size
9: of element i from parent P_2 or within the size range for element i)
10: randomly select one of the slots from $temp_1$ and move/adjust element i for this slot
11: **end if**
12: add element i to list L_1
13: **end if**
14: **end for**
15: **for** element i in set V **do**
16: **if** the value of the ith element in set V is 1 **then**
17: G_2 = build grid lines with L_2
18: **if** crossover at element i makes G_2 infeasible **then**
19: $temp_2$ = a set of possible slots within the grid lines G_2 (which are fitted to the size
20: of element i from parent P_1 or within the size range for element i)
21: randomly select one of the slots from $temp_2$ and move/adjust element i for this slot
22: **end if**
23: add element i to list L_2
24: **end if**
25: **end for**
26: **Output:** two generated solutions

4.2.2 The grid-based operator for mutation. Mutation functions by exploring the space via a minor change in the chromosome. It randomly jumps to a different part of the search space (refer to line 2 of Algorithm 3), hence keeping the algorithm from getting "stuck" at local optima. Randomly changing the size and position of some elements generates infeasible solutions for the most part. Accordingly, we use the same method based on grid lines and change the position, the size, or both to satisfy the overlap constraint (refer to lines 6-12 of Algorithm 3). Algorithm 3 covers all of the steps proposed.

5 IMPLEMENTATION AND COMPUTATIONAL PERFORMANCE

To evaluate the efficiency of our model and algorithm, we implemented it in Python 3.6 and ran it on a computer with a 1.7 GHz CPU, a Core i5 processor, and 8 GB of RAM. In addition, we used the Gurobi™ solver to find solutions for the mathematical model (defined by conditions 12 to 20) and used the DEAP [31] Python package to code the algorithm.

The following hyperparameters affect the performance of this algorithm: 1) Population size: this parameter has a main role in the computational time because in each generation all the steps of

Algorithm 3 Grid-based mutation steps

Input: a parent, P
R = a random binary set for determining the mutation elements
L = an index of elements in R for which the value is 0
for element i in set R **do**
  if the value of the ith element in set L is 1 **then**
    G = build grid lines with L
    if mutation at element i makes G infeasible **then**
      $temp$ = a set of possible slots within the grid lines G (which are fitted to the size of element i from parent P or within the size range for element i)
      randomly select one of the slots from $temp$ and move/adjust element i for this slot
    end if
    add element i to list L
  end if
end for
Output: generated solution

loop in Figure 2 should be calculated. In most studies, this number is between 50 and 200 [26, 39], and we considered 200 for it because our problem has 8 different objectives and a higher number for the population size could improve the diversity of final Pareto solution. 2) Number of generations: it refers to the number of cycles in which the algorithm is executed to reach the state of convergence. We run the algorithm 10 times and the solutions did not improve after around 450 generations. Therefore, we selected 500 generations as a condition to terminate the algorithm.

5.1 Performance Comparison

In this section, we consider the performance of the proposed algorithm relative to generic NSGA-III. We evaluated four cases: 1) random initial solutions and general operators (RI), 2) non-overlap initial solutions and general operators (NI), 3) random initial solutions and grid-based operators (RI-G), and 4) non-overlap initial solutions and grid-based operators (NI-G). Case 4 represents our algorithm.

To test performance in all cases, we considered three data instances. The values of the parameters for these instances are given in the appendix's Table 10. Various element sizes were tested, for more thorough comparison. For all the cases, we assumed a population size of 200 and set the number of generations to 500. The algorithm was run five times. Average computation times are reported in Table 3. For all cases, those for NI were a bit higher than those for RI, and NI-G's were slightly higher than RI-G's, on account of the algorithm taking more time to generate feasible initial solutions through solving of the model proposed in Subsection 4.1. In addition, the higher computation times reported for RI-G and NI-G stem mainly from the modification steps required with the grid-based operators. Tables 5 and 6 illustrate that the greater time use is more than offset by the performance benefits with the proposed algorithm.

Using the eight color-harmony templates and finding each element's distance from the various sectors of the hue wheel causes the color-harmony calculations to take too long. This significantly increases total calculation time. The effect is reported in Table 4, which shows that the mean value for all tests decreases around 50% when this measurement is omitted – a noticeable difference. Hence, using surrogate-based methods [14] to approximating the relevant value could significantly reduce total computation time.

Table 3. Computation time, in hours, for random initial solutions with general operators (RI), non-overlap initial solutions with general operators (NI), random initial solutions with grid-based operators (RI-G), and non-overlap initial solutions with grid-based operators (NI-G)

Number of elements	Statistical values	Computational time (h)			
		RI	NI	RI-G	NI-G
7	Mean	3.2	3.4	4.1	4.2
	SD	0.221	0.142	0.137	0.245
10	Mean	5.4	5.9	7.1	7.6
	SD	0.209	0.291	0.301	0.386
13	Mean	10.3	10.8	12.2	12.9
	SD	0.512	0.451	0.815	0.791

Table 4. Computation time, in hours, for all objectives and for all objectives without color harmony

Number of elements	Statistical values	Computational time (h)	
		All objectives	Without color harmony
7	Mean	4.2	2.4
	SD	0.245	0.317
10	Mean	7.6	3.2
	SD	0.386	0.289
13	Mean	12.9	6.82
	SD	0.791	0.495

To evaluate the performance of our algorithm, we used the two most popular metrics: inverted generational distance (IGD) [15] and hypervolume (HV) [100]. Together, these can reveal accuracy, or the convergence of a set [80], and diversity, referring to the set's distribution on the Pareto front [54]. They can be calculated, respectively, through

$$\mathrm{IGD}\,(P,S) = \frac{\left(\sum_{i=1}^{|P|} d_i^q\right)^{\frac{1}{q}}}{|P|}, \tag{21}$$

where d_i^q is the minimum distance from Pareto reference i to Pareto solution q – the calculation indicates how far an approximated Pareto solution is from the reference Pareto front, so a smaller value is better for this metric – and

$$\mathrm{HV}\,(S,R) = volume\left(\bigcup_{i=1}^{|S|} v_i\right). \tag{22}$$

For the HV metric, S is an index for the Pareto approximation solutions and R is a reference point, and v_i is the volume of solution i. This metric captures the diversity and distribution of Pareto solutions. Obviously, higher values are preferable for this metric. The results under the two metrics are shown in tables 5 and 6 (for IGD and HV, respectively).

In Table 5, NI-G has the lowest mean values for all three cases, demonstrating the efficiency of grid-based operators for improving the solutions' diversity relative to the approximated Pareto solutions.

Table 5. Statistics for the IGD values for various sizes – random initial solutions with general operators (RI), non-overlap initial solutions with general operators (NI), random initial solutions with grid-based operators (RI-G), and on-overlap initial solutions with grid-based operators (NI-G)

Number of elements	Statistical values	IGD value			
		RI	NI	RI-G	NI-G
7	Mean	0.643	0.586	0.132	**0.073**
	SD	0.026	0.134	**0.018**	0.107
10	Mean	0.731	0.504	0.187	**0.121**
	SD	0.242	**0.073**	0.119	0.094
13	Mean	0.908	0.872	0.265	**0.152**
	SD	0.143	0.159	0.153	**0.074**

Moreover, that the mean values for NI are smaller than those for RI shows the improvement in solution quality that arises from generating overlap-free initial solutions. Overall, the proposed non-overlap initial method with application of grid-based operators displays better performance in comparison to random initial solutions and general operators, under the IGD metric.

The results for the HV metric are tabulated in Table 6, where NI-G has the highest mean values across all three cases. As with IGD, the mean values reported for NI are higher than those for RI, which demonstrates the effectiveness of generating non-overlap initial solutions. Since the HV metric calculates the volume for a set of solutions, we can conclude that our method performs better in that it covers more space. This points to improved diversity of solutions relative to the approximated Pareto solutions.

Table 6. Statistics for the HV values for various sizes – random initial solutions with general operators (RI), non-overlap initial solutions with general operators (NI), random initial solutions with grid-based operators (RI-G), and non-overlap initial solutions with grid-based operators (NI-G)

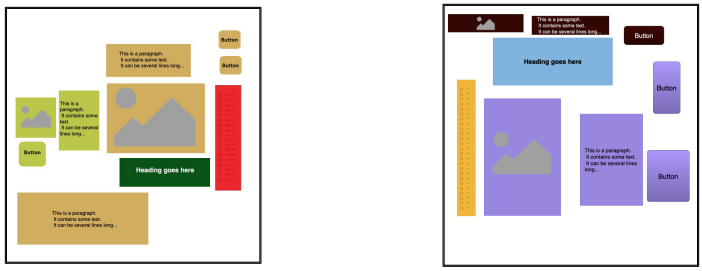
Number of elements	Statistical values	HV value			
		RI	NI	RI-G	NI-G
7	Mean	3.425	4.231	7.953	**8.497**
	SD	0.351	**0.159**	0.353	0.274
10	Mean	5.754	6.357	10.756	**11.346**
	SD	0.659	**0.344**	0.767	0.527
13	Mean	8.976	10.642	14.907	**15.549**
	SD	1.354	0.858	**0.564**	0.635

Another, more systematic way to measure the strength of the relation between two random search -based algorithms is to calculate their effect size [3, 49]. The effect sizes for 7, 10, 13 elements between NI-G and RI are 0.76, 0.61 and 0.94, respectively. According to the suggested guidelines [49, 90], the difference level between the result is 'large', which shows the efficacy of the algorithm proposed in this paper.

5.2 Example Application

To illustrate our approach concretely, we examine three cases. Two of them are for a 800×1260 desktop screen, and the other is for a mobile application with 375×667 screen size. Their data

are shown in the appendix's Table 11, 12, and 13. As already mentioned, the effect of image and video elements should not be included in the color performance metrics, except when there is a logo or a solid color for these elements. Firstly, the results for a run of generic NSGA-III for 10 elements after 500 generations are presented in Figure 6. As discussed in previous sections, applying this algorithm with general operators and a random initial generation cannot yield promising solutions. As explained in Subsection 2.3, the number of objectives led to nearly all the solutions generated being non-dominated ones after 10 generations. Hence, after execution of the algorithm, the number of solutions on the approximated Pareto front is equal to the population size. Obviously, the approximated Pareto solutions are not represented well. Figure 6 illustrates that the elements are not well-aligned and there are large empty spaces present. Hence, this technique does not seem appropriate for a final layout. However, the algorithm can find good color harmony among elements, and the elements' clustering by salience group attests to the algorithm's utility for addressing the grouping objective function.



(a) (b)

Fig. 6. A layout of solutions for NSGA-III via the original uniform operator and random initial generation for a 10-element task.

Next, we ran our proposed algorithm for 500 generations with a population size of 200. A sample of eight layouts on the approximated Pareto front is shown in Figure 7. All these layouts are well-aligned, and there are no gaps between elements. Furthermore, there are 200 solutions on the approximated Pareto front, from among which the designer can choose on the basis of his or her preferences.

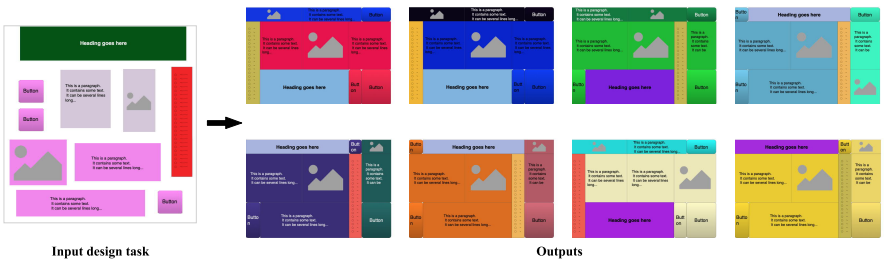


Fig. 7. A gallery of layouts representing the approximated Pareto-optimal solutions for 10 elements.

Figure 8 shows all the solutions, with their normalized objective values. Such parallel-coordinate plots are useful for summarizing all solution values in a single visualization, and a designer can use this tool to aid in identifying a preferred layout. For example, the image displays three solutions in

the approximated Pareto set in red, blue, and green, showing that these solutions do not dominate each other and that decreasing the value for one objective causes the values for the others to decrease or increase. The solution in red stands out from the blue and green ones. With such a figure, the designer can easily specify which combination of priorities is desired in the final layout.

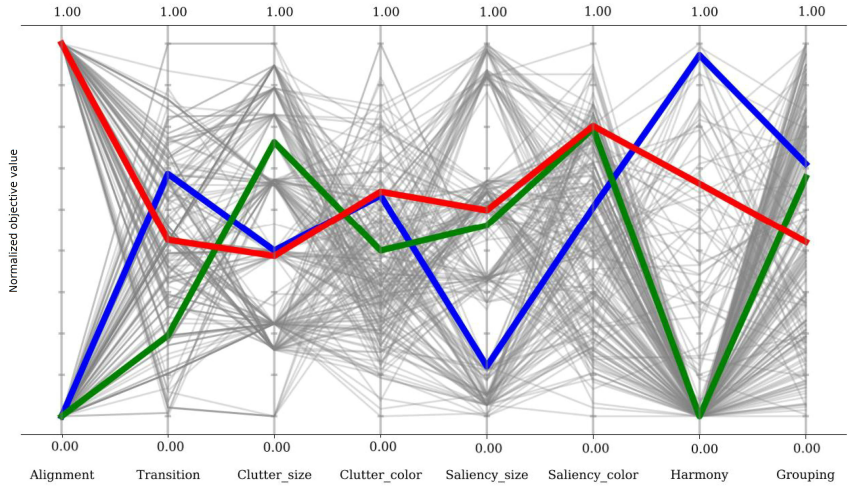


Fig. 8. A parallel-coordinate plot for all solutions on the Pareto front for 10 elements. As a sample, three randomly selected solutions are represented with green, red, and blue lines.

Figure 9 presents a sample of eight layouts for 13 elements on the final approximated Pareto front. Just as with the 10-element case, there are clear tradeoffs among objectives for the input design task. For example, some layouts use similar colors for some elements; the value for the color-clutter-related objective is lower for those layouts. Variety in shapes is evident too. The layouts in the rightmost column have six vertical and four horizontal alignment lines, while the others have five vertical and five horizontal alignment lines. Accordingly, the layouts stretch differently.

Fig. 9. A gallery of layouts for the approximated Pareto-optimal solutions for 13 elements.

The final example, for the mobile application, involves 10 elements: a header, a list, two images, two text elements, and four buttons. Eight of the final solutions are shown in Figure 10 for illustration purposes. In all shown solutions, "list element" is placed on the left or right side of the canvas. The salient element, which in our example is a "header", distinguished with a different color and size from other elements.

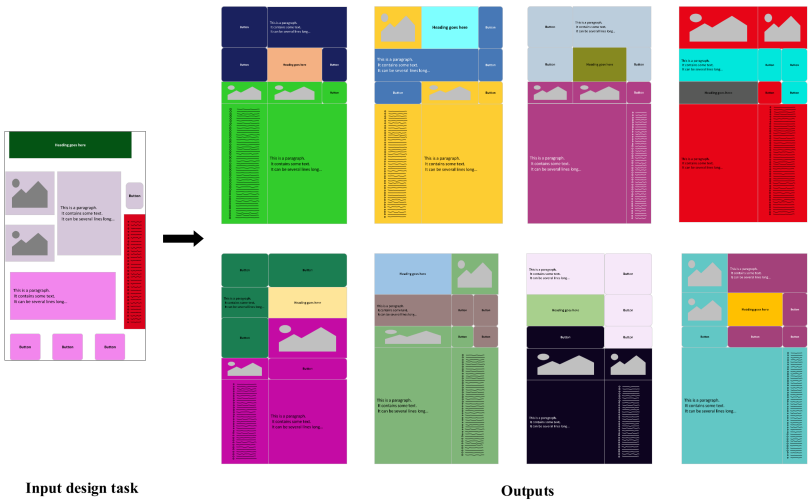


Fig. 10. A gallery of layouts for the approximated Pareto-optimal solutions for the mobile-application instance.

5.3 Perceived quality by users

We evaluated the efficiency of the generated layouts in a case study. To validate our approach, the perceived quality of layouts was assessed by users. Details and results of the study are explained as follows.

Method

Participants: We used Prolific[1] to recruit online users. This is an online research platform to find participants with different backgrounds [75]. A total of 50 participants were recruited and each participant spent around 5 minutes completing the task. According to the Prolific recommendation, each participant received £0.65 for the purpose of compensation. All participants were above 18 years old with normal or corrected-to-normal vision. Four of the participants did not complete the study, so the data of 46 were collected in the final analysis. Their age ranged from 18 to 58 years old (Mean: 29.98, SD: 11.52). 20 of the participants were female.

Procedure and design: Before the study, we executed our approach for a 15-element design task (their data are shown in the appendix's Table 14). Then, we randomly selected 5 of the layouts on the Pareto front solutions at three different NSGA-III generations (100, 300, and 500). For example, the layouts of 5 random solutions at generation 500 are shown in Figure 11. We used these steps because there is no other baseline available to consider colors or multiple objectives. During the study, all these 15 different layouts are randomly presented and the participants were asked to

[1]Prolific, https://www.prolific.co/

rate their perceived quality on a discrete scale of 0 (extremely low) to 10 (extremely high). The participants were completely free to rate according to their own judgment and preferences, and no other information was given to them about which design has a better/worse performance.

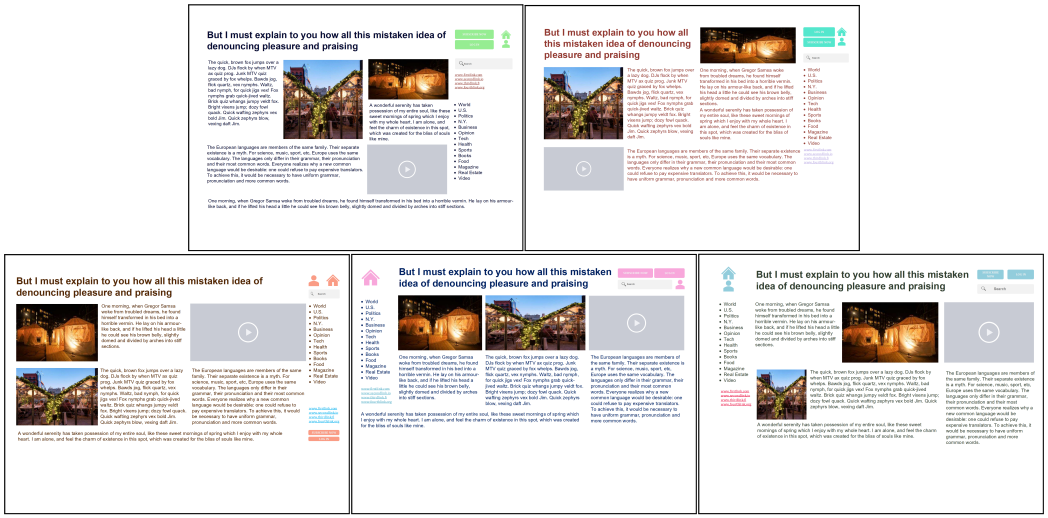


Fig. 11. A gallery of layouts for the approximated Pareto-optimal solutions for a 12-element instance.

Results: The summary of results for different generations are reported in Table 7. To statistically test the effect of generations on the perceived quality, the ANOVA test was calculated. This effect was found to be statistically significant ($p < 0.0001$). Since the ANOVA test only shows whether there is a difference between the means of groups or not, another test is required to reveal which groups are different. Therefore, post-hoc Tukey test [70] results are shown in Table 8. The difference between all generations is significant ($p < 0.001$) which means that our approach generated better layouts in its final generations.

Table 7. Mean and standard deviation from user ratings for different generations.

Generation	Quality rating	
	Mean	SD
100	1.91	1.61
300	4.58	2.01
500	6.98	1.65

6 LIMITATIONS

The control condition we used in the user study was based on the number of generations. The results indicate that by increasing the number of generations, human-perceived quality increases, which supports the choice of objectives in our system. However, this leaves open the question how good the outputs are in comparison to human-designed layouts. In the future, empirical studies should include layouts that are designed or improved by a human (e.g., [22, 67]). Also, our study

Table 8. Tukey test result for comparing different generations

Generation number		Mean diff.	P-Value	Reject
Case 1	Case2			
100	300	2.67	0.001	True
100	500	5.07	0.001	True
300	500	2.4	0.001	True

asked users to rate the general quality of a layout. In order to complement understanding of the components involved, future studies could include ratings that match the objectives [88].

Perhaps the main limitation of the current algorithm concerns the computation effort required. This limits the application of this method especially within interactive tools. However, we note that large and realistic task instances as reported here have never been solved earlier in the relevant literature.

7 CONCLUSION

The graphical-layout problem is markedly different from other layout problems, most notably the oft-studied facility layout problem. It is complicated by both the graphical elements and the objectives associated with visual appeal and attention. Hence, previous attempts of genetic algorithms in user-interface design were limited to keyboards and menus.

This paper marks the first demonstration that the full problem can be solved with genetic algorithms. Given that out-of-the-box approaches to GAs suffer from a high proportion of infeasible candidate designs, we were motivated to develop grid-based operators that make sure all candidates are well-aligned. This improves result quality significantly. Implementing the concept in NSGA-III allowed us to show that meaningful Pareto sets can be obtained in this manner.

We point to two avenues for future work. Firstly, although we have shown that high quality results can be produced by these objectives, researchers could explore new objectives and their effects on performance. For example, one might add visual balance and visual flow. Further, these could be fit to the individual style or preference of the designer. Secondly, computational time should be decreased, to permit interactive optimization with a human designer in the loop. Emerging work that combines machine learning approaches, such as deep reinforcement learning, with optimizer-generated data, are promising for this purpose, because – after training – running the optimizer can be much faster Bengio et al. [6].

ACKNOWLEDGMENTS

This work was supported by the Technology Industries of Finland (SOWP) grant number 700047 and the Academy of Finland (Human Automata) grant number 328813.

REFERENCES

[1] Murat Albayrak and Novruz Allahverdi. 2011. Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms. *Expert Systems with Applications* 38, 3 (2011), 1313–1320.

[2] Daniel Angus and Clinton Woodward. 2009. Multiple objective ant colony optimisation. *Swarm intelligence* 3, 1 (2009), 69–85.

[3] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 1–10.

[4] Gordon C Armour and Elwood S Buffa. 1963. A heuristic algorithm and simulation approach to relative location of facilities. *Management Science* 9, 2 (1963), 294–309.

[5] Slim Bechikh, Maha Elarbi, and Lamjed Ben Said. 2017. Many-objective optimization using evolutionary algorithms: a survey. In *Recent Advances in Evolutionary Multi-objective Optimization*. Springer, 105–137.

[6] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. 2020. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research* (2020).

[7] Jacques Bertin. 1983. *Semiology of graphics; diagrams networks maps*. Technical Report.

[8] Eric A Bier and Maureen C Stone. 1986. Snap-dragging. *ACM SIGGRAPH Computer Graphics* 20, 4 (1986), 233–240.

[9] Sara Bouzit, Gaëlle Calvary, Denis Chêne, and Jean Vanderdonckt. 2016. A design space for engineering graphical adaptive menus. In *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. 239–244.

[10] Ignacio Castillo, Joakim Westerlund, Stefan Emet, and Tapio Westerlund. 2005. Optimization of block layout design problems with unequal areas: A comparison of MILP and MINLP optimization methods. *Computers & Chemical Engineering* 30, 1 (2005), 54–69.

[11] Junjae Chae and Amelia C Regan. 2016. Layout design problems with heterogeneous area constraints. *Computers & Industrial Engineering* 102 (2016), 198–207.

[12] Ran Cheng, Miqing Li, Ye Tian, Xingyi Zhang, Shengxiang Yang, Yaochu Jin, and Xin Yao. 2017. A benchmark test suite for evolutionary many-objective optimization. *Complex & Intelligent Systems* 3, 1 (2017), 67–81.

[13] Asim Kumar Roy Choudhury. 2014. *Principles of colour and appearance measurement: Object appearance, colour perception and instrumental measurement*. Elsevier.

[14] Tinkle Chugh, Karthik Sindhya, Jussi Hakanen, and Kaisa Miettinen. 2019. A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms. *Soft Computing* 23, 9 (2019), 3137–3166.

[15] Carlos A Coello Coello and Nareli Cruz Cortés. 2005. Solving multiobjective optimization problems using an artificial immune system. *Genetic programming and evolvable machines* 6, 2 (2005), 163–190.

[16] Edward G Coffman, János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. 2013. Bin packing approximation algorithms: survey and classification. In *Handbook of combinatorial optimization*. Springer New York, 455–531.

[17] Daniel Cohen-Or, Olga Sorkine, Ran Gal, Tommer Leyvand, and Ying-Qing Xu. 2006. Color harmonization. In *ACM Transactions on Graphics (TOG)*, Vol. 25. ACM, 624–630.

[18] Yann Collette and Patrick Siarry. 2013. *Multiobjective optimization: principles and case studies*. Springer Science & Business Media.

[19] Constantinos K Coursaris, Sarah J Swierenga, and Ethan Watrall. 2008. An empirical investigation of color temperature and gender effects on web aesthetics. *Journal of usability studies* 3, 3 (2008), 103–117.

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

[21] Dianne Cyr, Milena Head, and Hector Larios. 2010. Colour appeal in website design within and across cultures: A multi-method evaluation. *International journal of human-computer studies* 68, 1-2 (2010), 1–21.

[22] Niraj Ramesh Dayama, Morteza Shiripour, Antti Oulasvirta, Evgeny Ivanko, and Andreas Karrenbauer. 2021. Foraging-based optimization of menu systems. *International Journal of Human-Computer Studies* 151 (2021), 102624.

[23] Niraj Ramesh Dayama, Kashyap Todi, Taru Saarelainen, and Antti Oulasvirta. 2020. GRIDS: Interactive Layout Design with Integer Programming. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[24] Kalyanmoy Deb. 2014. Multi-objective optimization. In *Search methodologies*. Springer, 403–449.

[25] Kalyanmoy Deb and Himanshu Jain. 2013. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE transactions on evolutionary computation* 18, 4 (2013), 577–601.

[26] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[27] William Drenttel and Jessica Helfand. 2006. Method and system for computer screen layout based on a recombinant geometric modular structure. US Patent 7,124,360.

[28] Maha Elarbi, Slim Bechikh, Lamjed Ben Said, and Rituparna Datta. 2017. Multi-objective optimization: classical and evolutionary approaches. In *Recent Advances in Evolutionary Multi-objective Optimization*. Springer, 1–30.

[29] Peter J Fleming, Robin C Purshouse, and Robert J Lygoe. 2005. Many-objective optimization: An engineering design perspective. In *International conference on evolutionary multi-criterion optimization*. Springer, 14–32.

[30] David B Fogel. 1997. The Advantages of Evolutionary Computation.

[31] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13 (jul 2012), 2171–2175.

[32] Garrett Foster and Scott Ferguson. 2013. Enhanced Targeted Initial Populations for Multiobjective Product Line Optimization. In *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers Digital Collection.

[33] Mathias Frisch, Sebastian Kleinau, Ricardo Langner, and Raimund Dachselt. 2011. Grids & guides: multi-touch layout and alignment tools. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1615–1618.

[34] Krzysztof Gajos and Daniel S Weld. 2004. SUPPLE: automatically generating user interfaces. In *Proceedings of the 9th international conference on Intelligent user interfaces*. 93–100.

[35] Michael R Garey, David S Johnson, and Larry Stockmeyer. 1974. Some simplified NP-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*. ACM, 47–63.

[36] Mikhail Goubko and Alexander Varnavsky. 2016. Users' preference share as a criterion for hierarchical menu optimization. In *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. 305–310.

[37] Mikhail V Goubko and Alexander I Danilenko. 2010. An automated routine for menu structure optimization. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*. 67–76.

[38] David Walter Hamlyn. 2017. *The psychology of perception: A philosophical examination of Gestalt theory and derivative theories of perception*. Routledge.

[39] Randy L Haupt and Sue Ellen Haupt. 2004. Practical genetic algorithms. (2004).

[40] Francisco Herrera, Manuel Lozano, and Ana M Sánchez. 2003. A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *International Journal of Intelligent Systems* 18, 3 (2003), 309–338.

[41] John H Holland. 1992. Genetic algorithms. *Scientific american* 267, 1 (1992), 66–73.

[42] Hasan Hosseini-Nasab, Sepideh Fereidouni, Seyyed Mohammad Taghi Fatemi Ghomi, and Mohammad Bagher Fakhrzad. 2018. Classification of facility layout problems: a review study. *The International Journal of Advanced Manufacturing Technology* 94, 1-4 (2018), 957–977.

[43] Allen Hurlburt. 1978. The grid: A modular system for the design and production of newspapers, magazines.

[44] Abid Hussain and Yousaf Shad Muhammad. 2019. Trade-off between exploration and exploitation with genetic algorithm using a novel selection operator. *Complex & Intelligent Systems* (2019), 1–14.

[45] Hisao Ishibuchi, Noritaka Tsukamoto, and Yusuke Nojima. 2008. Evolutionary many-objective optimization: A short review. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 2419–2426.

[46] Laurent Itti and Christof Koch. 2000. A saliency-based search mechanism for overt and covert shifts of visual attention. *Vision research* 40, 10-12 (2000), 1489–1506.

[47] Himanshu Jain and Kalyanmoy Deb. 2013. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: handling constraints and extending to an adaptive approach. *IEEE Transactions on Evolutionary Computation* 18, 4 (2013), 602–622.

[48] Yue Jiang, Wolfgang Stuerzlinger, Matthias Zwicker, and Christof Lutteroth. 2020. ORCSolver: An Efficient Solver for Adaptive GUI Layout with OR-Constraints. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–14.

[49] Vigdis By Kampenes, Tore Dybå, Jo E Hannay, and Dag IK Sjøberg. 2007. A systematic review of effect size in software engineering experiments. *Information and Software Technology* 49, 11-12 (2007), 1073–1086.

[50] Sumin Kang and Junjae Chae. 2017. Harmony search for the layout design of an unequal area facility. *Expert Systems with Applications* 79 (2017), 269–281.

[51] Kengo Katayama, Hisayuki Hirabayashi, and Hiroyuki Narihisa. 2003. Analysis of crossovers and selections in a coarse-grained parallel genetic algorithm. *Mathematical and Computer Modelling* 38, 11-13 (2003), 1275–1282.

[52] Mustafa Kaya. 2011. The effects of two new crossover operators on genetic algorithm performance. *Applied Soft Computing* 11, 1 (2011), 881–890.

[53] André A Keller. 2017. *Multi-objective optimization in theory and practice i: classical methods*. Bentham Science Publishers.

[54] Vineet Khare, Xin Yao, and Kalyanmoy Deb. 2003. Performance scaling of multi-objective evolutionary algorithms. In *International conference on evolutionary multi-criterion optimization*. Springer, 376–390.

[55] Kurt Koffka. 2013. *Principles of Gestalt psychology*. Routledge.

[56] Sadan Kulturel-Konak and Abdullah Konak. 2011. Unequal area flexible bay facility layout using ant colony optimisation. *International Journal of Production Research* 49, 7 (2011), 1877–1902.

[57] Eugene L Lawler. 1963. The quadratic assignment problem. *Management science* 9, 4 (1963), 586–599.

[58] Hsin-Ying Lee, Weilong Yang, Lu Jiang, Madison Le, Irfan Essa, Haifeng Gong, and Ming-Hsuan Yang. 2019. Neural Design Network: Graphic Layout Generation with Constraints. *arXiv preprint arXiv:1912.09421* (2019).

[59] Bingdong Li, Jinlong Li, Ke Tang, and Xin Yao. 2015. Many-objective evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)* 48, 1 (2015), 13.

[60] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. 2019. Layoutgan: Generating graphic layouts with wireframe discriminators. *arXiv preprint arXiv:1901.06767* (2019).

[61] Jianan Li, Jimei Yang, Jianming Zhang, Chang Liu, Christina Wang, and Tingfa Xu. 2020. Attribute-conditioned Layout GAN for Automatic Graphic Design. *IEEE Transactions on Visualization and Computer Graphics* (2020).

[62] Simon Lok, Steven Feiner, and Gary Ngai. 2004. Evaluation of visual balance for automated layout. *International Conference on Intelligent User Interfaces, Proceedings IUI*, 101–108. https://www.scopus.com/inward/record.uri?eid=2-s2.0-18744410375&partnerID=40&md5=f9690e725468c7bd2457bf6781818d00

[63] Ellen Lupton. 2014. *Thinking with type: A critical guide for designers, writers, editors, & students.* Chronicle Books.

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

[65] Aaron Marcus. 1997. Graphical user interfaces. In *Handbook of human-computer interaction*. Elsevier, 423–440.

[66] Dimitri Masson, Alexandre Demeure, and Gaelle Calvary. 2010. Magellan, an evolutionary system to foster user interface design creativity. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*. 87–92.

[67] Nolwenn Maudet, Ghita Jalal, Philip Tchernavskij, Michel Beaudouin-Lafon, and Wendy E Mackay. 2017. Beyond grids: Interactive graphical substrates to structure digital layout. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 5053–5064.

[68] Mohamed Wiem Mkaouer, Marouane Kessentini, Slim Bechikh, Kalyanmoy Deb, and Mel Ó Cinnéide. 2014. High dimensional search-based software engineering: finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. 1263–1270.

[69] N Monmarché, G Nocent, M Slimane, G Venturini, and P Santini. 1999. Imagine: a tool for generating HTML style sheets with an interactive genetic algorithm based on genes frequencies. In *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028)*, Vol. 3. IEEE, 640–645.

[70] Douglas C Montgomery. 2017. *Design and analysis of experiments*. John wiley & sons.

[71] E Osaba, R Carballedo, F Diaz, E Onieva, I De La Iglesia, and A Perallos. 2014. Crossover versus mutation: a comparative analysis of the evolutionary strategy of genetic algorithms applied to combinatorial optimization problems. *The Scientific World Journal* 2014 (2014).

[72] Antti Oulasvirta, Niraj Ramesh Dayama, Morteza Shiripour, Maximilian John, and Andreas Karrenbauer. 2020. Combinatorial Optimization of Graphical User Interface Designs. *Proc. IEEE* (2020).

[73] 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*. 2765–2774.

[74] Frederico Galaxe Paes, Artur Alves Pessoa, and Thibaut Vidal. 2017. A hybrid genetic algorithm with decomposition phases for the unequal area facility layout problem. *European Journal of Operational Research* 256, 3 (2017), 742–756.

[75] Stefan Palan and Christian Schitter. 2018. Prolific. ac—A subject pool for online experiments. *Journal of Behavioral and Experimental Finance* 17 (2018), 22–27.

[76] Juan M Palomo-Romero, Lorenzo Salas-Morera, and Laura García-Hernández. 2017. An island model genetic algorithm for unequal area facility layout problems. *Expert Systems with Applications* 68 (2017), 151–162.

[77] Silvia Poles, Yan Fu, and Enrico Rigoni. 2009. The effect of initial population sampling on the convergence of multi-objective genetic algorithms. In *Multiobjective programming and goal programming*. Springer, 123–133.

[78] Aurora Ramirez, José Raúl Romero, and Sebastian Ventura. 2019. A survey of many-objective optimisation in search-based software engineering. *Journal of Systems and Software* 149 (2019), 382–395.

[79] Margarita Reyes-Sierra, CA Coello Coello, et al. 2006. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International journal of computational intelligence research* 2, 3 (2006), 287–308.

[80] Nery Riquelme, Christian Von Lücken, and Benjamin Baran. 2015. Performance metrics in multi-objective optimization. In *2015 Latin American Computing Conference (CLEI)*. IEEE, 1–11.

[81] Irvin Rock and Stephen Palmer. 1990. The legacy of Gestalt psychology. *Scientific American* 263, 6 (1990), 84–91.

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

[83] Jerrold M Seehof, Wayne O Evans, James W Friederichs, and James J Quigley. 1966. Automated facilities layout programs. In *Proceedings of the 1966 21st national conference*. ACM, 191–199.

[84] Ingrida Steponavičė, Mojdeh Shirazi-Manesh, Rob J Hyndman, Kate Smith-Miles, and Laura Villanova. 2016. On sampling methods for costly multi-objective black-box optimization. In *Advances in Stochastic and Deterministic Global Optimization*. Springer, 273–296.

[85] KS Swarup and S Yamashiro. 2002. Unit commitment solution methodology using genetic algorithm. *IEEE Transactions on power systems* 17, 1 (2002), 87–91.

[86] Amanda Swearngin, Chenglong Wang, Alannah Oleson, James Fogarty, and Amy J Ko. 2020. Scout: Rapid Exploration of Interface Layout Alternatives through High-Level Design Constraints. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[87] Madjid Tavana, Zhaojun Li, Mohammadsadegh Mobin, Mohammad Komaki, and Ehsan Teymourian. 2016. Multi-objective control chart design optimization using NSGA-III and MOPSO enhanced with DEA and TOPSIS. *Expert Systems with Applications* 50 (2016), 17–39.
[88] Kashyap Todi, Daryl Weir, and Antti Oulasvirta. 2016. Sketchplore: Sketch and explore with a layout optimiser. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*. ACM, 543–555.
[89] Andrew C Trapp and Renata A Konrad. 2015. Finding diverse optima and near-optima to binary integer programs. *IIE Transactions* 47, 11 (2015), 1300–1312.
[90] Tammi Vacha-Haase and Bruce Thompson. 2004. How to estimate and interpret various effect sizes. *Journal of counseling psychology* 51, 4 (2004), 473.
[91] Patricia Valdez and Albert Mehrabian. 1994. Effects of color on emotions. *Journal of experimental psychology: General* 123, 4 (1994), 394.
[92] Pengfei Xu, Hongbo Fu, Takeo Igarashi, and Chiew-Lan Tai. 2014. Global beautification of layouts with interactive ambiguity resolution. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, 243–252.
[93] Takuto Yanagida, Hidetoshi Nonaka, and Masahito Kurihara. 2009. Personalizing graphical user interfaces on flexible widget layout. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*. 255–264.
[94] Xin-She Yang. 2014. *Nature-inspired optimization algorithms*. Elsevier.
[95] Xiaohui Yuan, Hao Tian, Yanbin Yuan, Yuehua Huang, and Rana M Ikram. 2015. An extended NSGA-III for solution multi-objective hydro-thermal-wind scheduling considering wind power cost. *Energy Conversion and Management* 96 (2015), 568–578.
[96] Shumin Zhai, Michael Hunter, and Barton A Smith. 2002. Performance optimization of virtual keyboards. *Human–Computer Interaction* 17, 2-3 (2002), 229–269.
[97] Xinru Zheng, Xiaotian Qiao, Ying Cao, and Rynson WH Lau. 2019. Content-aware generative modeling of graphic design layouts. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–15.
[98] Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Qingfu Zhang. 2011. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation* 1, 1 (2011), 32–49.
[99] Yingying Zhu, Junwei Liang, Jianyong Chen, and Zhong Ming. 2017. An improved NSGA-III algorithm for feature selection used in intrusion detection. *Knowledge-Based Systems* 116 (2017), 74–85.
[100] Eckart Zitzler and Lothar Thiele. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE transactions on Evolutionary Computation* 3, 4 (1999), 257–271.

APPENDIX

Table 9. Instance data for five elements

Element number	Min. width	Max. width	Min. height	Max. height
1	200	500	50	350
2	200	350	50	400
3	100	500	100	200
4	50	400	100	300
5	50	250	50	600

This test has 10 elements, the data for which are shown in Table 11. The elements differ in their size limitations and belong to different groups. Elements 1 and 6 are the most important elements, and we consider them the salient ones. The pairwise transition matrix is generated randomly.

This test has 13 elements, the data for which are listed in Table 12. There are two groups, and elements 2 and 8 are the salient elements. As in the previous test, the pairwise transition matrix is generated randomly.

Table 10. Values of the parameters

Parameter	**Value**
Canvas size	600×800
Min. width	$\sim$ Uniform[50, 150]
Max. width	(Min. width) $*$ 1.2
Min. height	$\sim$ Uniform[100, 250]
Max. height	(Min. height) $*$ 1.2
Transition matrix	$\sim$ Uniform[0, 10]
Number of groups	A random integer between 2 and 4

Table 11. Data for 10 elements

Row	Element	Min. width	Max. width	Min. height	Max. height	Group number	Salient
1	Heading	500	700	50	200	1	*
2	Image	200	350	200	350	1	
3	Paragraph	200	450	200	350	1	
4	Button	50	200	50	200	1	
5	Button	50	200	50	200	1	
6	Paragraph	400	600	300	550	1	*
7	List	50	200	400	650	1	
8	Image	200	350	100	300	2	
9	Paragraph	200	350	100	300	2	
10	Button	50	200	50	200	2	

Table 12. Data for 13 elements

Row	Element	Min. width	Max. width	Min. height	Max. height	Group number	Salient
1	Heading	100	300	50	150	1	
2	Image	50	250	100	250	1	*
3	Button	50	200	100	200	1	
4	Button	50	150	150	300	1	
5	Button	100	300	50	200	1	
6	Paragraph	100	300	100	350	1	
7	List	50	300	150	400	2	
8	Image	100	300	100	350	2	*
9	Paragraph	50	250	250	450	2	
10	Image	50	500	100	600	2	
11	List	50	250	250	450	2	
12	Paragraph	50	500	100	600	2	
13	Paragraph	50	500	100	600	2	

Table 13. Data for the mobile-application instance

Row	Element	Min. width	Max. width	Min. height	Max. height	Group number	Salient
1	Heading	150	250	50	150	1	*
2	Button	50	250	50	150	1	
3	List	50	150	200	500	1	
4	Paragraph	200	350	250	400	1	
5	Image	100	250	50	150	1	
6	Image	100	250	50	150	1	
7	Button	50	150	50	150	2	
8	Button	50	150	50	150	2	
9	Button	50	150	50	150	2	
10	Paragraph	200	375	100	250	2	

Table 14. Data for the user study

Row	Element	Min. width	Max. width	Min. height	Max. height	Group number	Salient
1	Heading	200	900	100	350	1	
2	Image	200	500	150	400	1	
3	Paragraph	200	500	150	400	1	
4	Image	200	500	150	400	1	
5	Paragraph	200	500	150	400	1	
6	Video	200	600	150	500	1	
7	Paragraph	200	500	150	400	1	
8	Hyperlink	50	300	50	250	1	*
9	Paragraph	300	1200	50	250	1	
10	Search box	50	300	50	150	1	
11	List	50	250	100	350	1	
12	Button	50	200	50	150	2	*
13	Button	50	200	50	150	2	*
14	Button	50	150	50	150	2	*
15	Button	50	150	50	150	2	*

Received October 2020; revised March 2021; accepted April 2021