MenuOptimizer: Interactive Optimization of Menu Systems

Gilles Bailly[1] **Antti Oulasvirta**[1] **Timo Kötzing**[2] **Sabrina Hoppe**[1]

[1]Max Planck Institute for Informatics and Saarland University [2]University of Jena

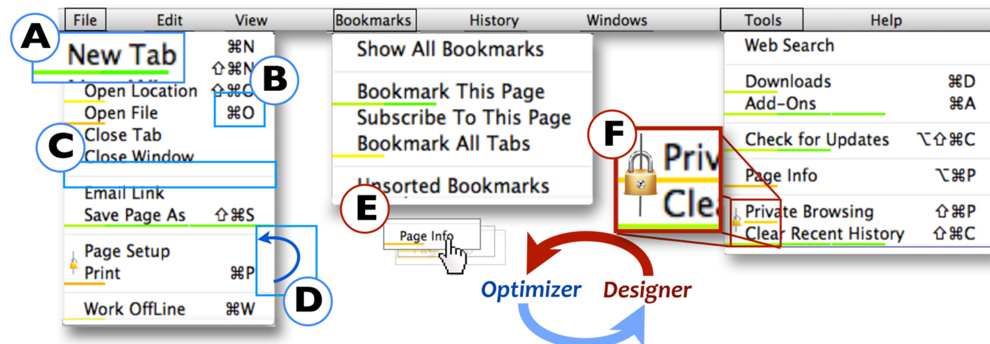

Figure 1: MenuOptimizer assists in the design of menus: While the designer edits the menu (action in red), a model-based optimizer updates itself to provide feedback and suggestions (in blue): A) Item feedback indicates the frequency (line width) and user performance over time (color gradient). B–C) Hotkeys and separators are automatically assigned. D) Item placements to improve user performance are suggested. E) Designers can normally edit items (move, delete, etc.) and also F) lock items to constrain them together to accelerate optimization.

ABSTRACT

Menu systems are challenging to design because design spaces are immense, and several human factors affect user behavior. This paper contributes to the design of menus with the goal of interactively assisting designers with an optimizer in the loop. To reach this goal, 1) we extend a predictive model of user performance to account for expectations as to item groupings; 2) we adapt an ant colony optimizer that has been proven efficient for this class of problems; and 3) we present MenuOptimizer, a set of interactions integrated into a real interface design tool (QtDesigner). MenuOptimizer supports designers' abilities to cope with uncertainty and recognize good solutions. It allows designers to delegate combinatorial problems to the optimizer, which should solve them quickly enough without disrupting the design process. We show evidence that satisfactory menu designs can be produced for complex problems in minutes.

Author Keywords

Menus; Predictive models; Interactive optimization.

ACM Classification Keywords

H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION

Menu systems, consisting of menus, hotkeys or toolbars, are widespread interfaces for selecting commands. Interface design strongly affects their usability. However, despite apparent simplicity, designing usable menu systems is challenging because the number of alternative designs grows superexponentially as a function of the number of commands. For instance, a linear menu with n items can, in theory, be organized in $n!$ ways. However, professional applications comprise hundreds of items organized in hierarchical menus. A menu hierarchy can be organized in about $(2n)!$ ways. For 50 items, the size of the search space is a whopping $100! \approx 10^{158}$. Design heuristics, such as placing frequently used items at the top [5], may be effective for small n but fail with larger n or if additional human factors such as semantic relationships among items are considered. Although experts can quickly generate a handful of solutions to hard design problems [6], they cannot examine all promising solutions. Novices, known to search the space depth-first [3], are likely to get stuck in a local search space.

Combinatorial optimization methods (e.g., [29]) have been successfully used to generate user interfaces such as virtual keyboards [8,21,24,34]. These methods explore a large number of designs in order to find ones that minimize or maximize a pre-specified objective function. Computation time is on a scale of hours, days, or weeks. While empirical evidence confirms improvements in usability in other contexts (e.g., [24,34]), there is reason to suggest that they may be impractical for the design of menu systems. First, designers cannot be expected to wait days or weeks for a solution. Moreover, designers may not be able to define the optimization problem completely in advance. Interaction design, in general, is rather more an iterative process of redefinition and refinement. Finally, predictive models for menu systems performance are only just emerging [5,20,25], are limited to linear menus, and have yet to cover all important human factors that affect design choices.

This paper contributes the *design of a user interface and an optimization method for interactively assisting in menu design with an optimizer in the loop*. MenuOptimizer illustrates our approach by integrating an optimizer into QtDesigner, a widely used design tool. The goal is to improve the quality of process and outcomes in the design of menus. The key challenges are, on the one hand, how to support designers' abilities to cope with uncertainty, define constraints, and recognize good solutions and, on the other hand, to allow designers to delegate combinatorial subproblems to an optimizer that should solve them rapidly enough not to disrupt the process of design.

Figure 1 reviews the concept of designer–optimizer interactions in MenuOptimizer. At present, it supports the design of static hierarchical menus. Two design scenarios are supported: 1) In the *updating* of existing menus, user behavior is partially known from logs or interviews. 2) *Sketching* novel menus involves assumptions about usage and objective, which can change during the design phase. Both scenarios are supported with the same functionality: Designers can edit menus by adding, renaming, or moving commands in it, and every edit updates the optimizer to work with the present menu design. The optimizer provides suggestions at item level (optimal placement for an item), menu level (improvements made by moving items), and the menu system level (menu templates). Designers can specify and change their assumptions about use and users by editing distribution graphs directly. To direct the optimizer, designers can also change the objective function by setting the relative weight of user performance, consistency (expectations as to item locations), and edit distance to the present design. With multiple optimization objectives, the optimizer's suggestions are plotted in real time on a Pareto front plot. Designers can also lock items, which restricts the optimizer's search space.

To realize the interactions in MenuOptimizer, we extended a state-of-the-art predictive model of user performance (*Search–Decision–Pointing; SDP* [5]) with a model of expected item groupings; the system proposes logical groups and hierarchies that are statistically consistent with other applications. Furthermore, we show that the menu design problem can be formulated as a known optimization problem. We then describe an ant colony optimizer, a method proven to be efficient for this class of problem.

After presenting MenuOptimizer, we show evidence from several evaluations. We consider two realistic design cases. The first is the optimization of Mozilla Firefox from logs collected on regular users; the second involves the image editing software Seashore. We report on the short- and long-term performance of the optimizer, showing that it can rapidly produce satisfactory suggestions for menus. Moreover, data from a preliminary study suggests that interactions with MenuOptimizer can help novice designers to design menus with less editing effort. Finally, we detail further directions for interactive optimization of menu systems.

MOTIVATION AND FORMATIVE STUDY

Designing user interfaces is, in general, a complex, expensive, and time-consuming process [6,26,28,32]. Some reasons cited in existing literature are related to the issue of combinatorial explosion as described above: 1) Designers cannot explore the space of possible interface designs manually [14,26]. 2) Designers underestimate the diversity of the user population [15]. 3) They have difficulties in demonstrating the advantages of a design to others [26]. 4) Finally, designers find it difficult to analyze a design's efficiency [32]. For instance, in the evaluation of a design, there is a tendency to favor aesthetics over efficiency [32].

Because we were unable to find literature on such problems specific to *menu systems*, we conducted a pre-study. Six experienced designers were interviewed. They all work in the industry as designers or usability professionals. They stated that they design between zero and three *complex* (~1,000-item) menu systems per year in the context of software or Web applications and 1–5 intermediately complex menu systems (~100 items) a year. Their reported range of command sets is 50 to 1,000 commands. They stated their main design goals as being "understandability" and "allowing users to find what they want." However, item selection time for experienced users (users who already know item locations) appears not to be among their main concerns because they need to satisfy the immediate expectations of their clients.

The interviewees confirmed that designing large menu systems is "very difficult" and involves "trial and error in the beginning." Several design challenges were mentioned, among them "labeling top-level menus," "coherence between commands and their labels," and "choosing the best placements for items." They described their design process as "iterative" and involving feedback from final users. They also confessed that they do not always seek the best solutions. For instance, "it happens that we put commands in two places because they are not in a perfect location in either. However, we know that this also introduces confusion." To solve the problems mentioned above, the designers considered important "the logic of the software," "experience," and "feedback from customers." None of the designers mentioned collecting logs for individual users, but they do use analytical tools (e.g., Google Analytics) to learn about frequency distributions for commands. We learned that updating the hierarchy of commands (links on Web sites) in line with visitors' expectations was a frequent task.

Finally, when asked what kind of support they would desire, the designers stated tools that "provide alternative versions of the current menu system" so that they "can compare designs." They also wanted to "check whether all commands are in the menu system."

While this pre-study interviews included only professionals, we believe that designing small-to-intermediate menus must involve a large number of applications and designers.

RELATED WORK

MenuOptimizer contributes to the areas of automatically generated interfaces [11,17,28], mixed-initiative design [16] and, in particular, to optimizer-assisted design. We concentrate on optimizer-assisted design here, identifying three sub-areas: 1) designs generated by one-shot (offline) optimizers, 2) design guided by the output of an offline optimizer, and 3) design optimization with a human in the loop.

1. *Offline (non-interactive) optimization* has been used in HCI in the context of widget layouts [31], text entry [21,24,34], menus [14,23], accessibility [1213], and dialog layout [9]. Regarding menus, Goubko et al. [14] proposed a framework for optimizing menus on cellular phones. However, they impose a constraint on the menu structure, use a contrived predictive model [20], and describe neither the optimization method nor evaluate its outcomes. Matsui et al. [23] proposed an optimizer for hierarchical menus on mobile devices that is based on genetic algorithms and simulated annealing. As the objective function, they used the SDP model [5] and introduced two costs: that of favoring proximity between semantically related items and a granularity cost to favor balanced hierarchies. Our optimization method also builds on the SDP model, but we introduce several necessary improvements, including a model of consistency that can suggest item groupings. Importantly, we reject the constraint of balanced hierarchies, making the task harder but more realistic.

2. *Optimization results can be seeds for the design process.* AIDE [32] initially computes a layout that the designer can then edit. At all times, the designer can compare the present layout with the precomputed optimum generated with the branch and bound method. In the design of Template Gallery (see Figure 2B), we follow the idea of comparing the present menu with the best known. MenuOptimizer also continuously provides suggestions during editing.

3. *To involve the designer in the optimization process* [3,27,30]: Quiroz et al. proposed an interactive genetic algorithm for generating widget layouts when no predictive models are available [27]. In each iteration, the designer chooses the best three designs out of ten to guide the optimizer. While not addressing UI design, Scott et al. [30] proposed a technique in which the user can guide an optimizer by constraining its search space during optimization.

To our knowledge, *interactive* optimization of menus has not previously been studied. MenuOptimizer builds on ideas from the three areas mentioned above. 1) The optimizer can be used in offline mode; after defining items and user behavior, the optimizer searches for the best solutions. 2) MenuOptimizer also provides a visualization of suggestions that the designer can choose to follow, and guidance is offered via comparison of the current menu with the best known. 3) MenuOptimizer does not assume perfect knowledge in the predictive model. Several interactions allow the designer to guide and focus the optimizer.

WALKTHROUGH

We introduce MenuOptimizer in a design scenario. The interface elements in parentheses refer to Figures 1–5. The scenario is divided into two stages: global and local editing.

1. Global editing: *Kim is designing a menu system for image editing software with a set of 50 commands. Kim first states her command set by typing commands in an empty menu. Then, she explicates her assumptions about users and their frequencies of command selections* (User Profile Panel). *She uses a slider to indicate that she prefers to optimize for consistency instead of performance* (Objective Panel). *Now the optimization problem has been set, the optimizer starts to propose menu designs, which are visible* (Template Gallery). *Kim examines the gallery as the suggestions appear and selects one with four submenus and large logical groups. Two menus have titles that are given automatically.*

2. Local editing: *Kim is not happy with the* Crop item *being far away from* Select*, so she moves the two together by drag-and-drop. The optimizer shows that this placement decreases a user's average selection performance by 1.5%. Nevertheless, Kim moves the items because she wants to keep them together. Then, Kim follows the optimizer's suggestion to move* Select *to the* Edit *menu* (Suggestions)*. Kim then marks* Undo*,* Redo*, and a few other items with* Locks *because she wants to keep these items together. From now on, the optimizer only suggests designs maintaining this group. Finally, Kim finalizes the design by asking* MenuOptimizer *to assign hotkeys. As the menu is created in* QtDesigner*, it can be easily instantiated in the application.*

This scenario should take about 15 minutes, with the time for the global and local phases split roughly half and half. To realize this scenario, we need 1) a predictive model of menu performance, 2) an optimization method, and 3) interactions between the designer and the optimizer. We now present these three components.

ADAPTING A PREDICTIVE MODEL FOR MENUS

Model-based optimization critically relies on a valid and comprehensive *predictive model*. Several models of menu performance have been proposed [1,2,5,20,25]. We implement and extend the most recent model: the SDP model [5]. It covers four human factors: target acquisition, visual search, decision-making, and learning (see below).

To assist the designer better, two adaptations have been made to the existing model. First, we extend the model to user performance with hierarchical menus attached to the menu bar. Second, the grouping of commands into submenus or logical groups (marked by separator lines) is central to menus. Such decisions should be informed by a notion of "semantics", indicating which commands belong together. Our informal tests suggested that user *expectations* of item groups from previous applications are central. Both designers and users expect certain commands to be together (e.g., Undo and Redo). Having these adaptations in the optimizer frees the designer from the work of grouping items.

SDP: A Predictive Model for Linear Menus

SDP (Search–Decision–Pointing) is a three-component linear regression model that predicts the selection time of an item in a menu [5]: **Search**, the time to localize an item, increases linearly with the number of items in the menu. **Decision** is the time to decide from among items given the "entropy" determined by the frequencies of previous selections and given by the Hick-Hyman law. **Pointing** is based on Fitts' law and predicts that items closer to the top are faster to select. Search and decision are modulated by the number of repetitions of an item. With practice, performance shifts from being dominated by search (linear) to decision (logarithmic). Our model parameters are the same as [5], but the designer can change them to test hypotheses.

Adaptation 1: Menus Attached to the Menu Bar

To deal with a menu system, a model must account for two aspects: 1) steering from one submenu to another and 2) menu bars. For the former, we use a *steering cost,* proposed in [1], that penalizes designs with deep submenu structures. For the latter, we assume that at the beginning of a selection, the cursor is on the left side of the screen and in the middle and that items have a width four times their height.

Adaptation 2: Consistency Score for Grouping Items

Here we propose a model that allows the optimizer to produce logical groups and hierarchies that are statistically consistent with other applications. To allow this, we built a database of item co-occurrences. The database currently has 3,290 commands and 111,859 command pairs collected by a menu-logger tool [22] from 68 applications in Mac OS X. Using this database, we compute a score for the expected relative position of command pairs in a selected set of prior applications. Each command pair i, j has an expectation score $E(i, j)$ between 0 and 1, representing the two commands' tendency to be close in prior menus. A higher score is given to pairs that tend to be in the same submenu and the highest to pairs that are always in the same logical group (e.g., Undo and Redo). The expectation score $E(i, j)$ for a given menu is computed as follows:

$$E(i,j) = \frac{0.33}{1+D_{ij}}(Hierarchy(i,j) + Menu(i,j) + Group(i,j)) \quad (1)$$

where $Hierarchy(i,j)$, $Menu(i,j)$, and $Group(i,j)$ each returns 1 if i and j are in the same hierarchy (have a common ancestor), menu, or logical group, respectively; and 0 otherwise. D_{ij} is the shortest path in the hierarchy to reach j from i. Therefore, for command pairs that tend to be in a child–parent relationship in the menu tree, the score is between 0 and 1/3. The score is above 1/3 if they are in the same submenu and greater than 2/3 if they are in the same group. For commands that are *not* in the database, we use a simple lexical similarity score: the number of shared words.

$$E_{lexical}(i,j) = \frac{commonWords(i,j)}{\max(length(i), length(j))} \quad (2)$$

For instance, E(Show All Bookmarks, Hide Bookmarks) is 1/3. For item pairs not sharing words, we assign a score of 0.0. To calculate the average expectation score $E_{average}$, we average the expectation scores over selected applications.

Evaluating the consistency of a given menu: Now, the computed $E_{average}(i, j)$ score can be used for determining a score for the consistency of a given menu system C_{menu} over pairwise comparisons with its expectation score E_{menu}:

$$C_{menu} = \frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n} | E_{menu}(i,j) - E_{average}(i,j) |}{n(n-1)/2}, \quad (3)$$

where n is the number of items in menu system M.

This score has several desirable effects on the optimizer (one result is shown later in Figure 4):

1. Item pairs that tend to be in the same logical groups are "pulled" together.
2. Item pairs that cannot be positioned to match $E_{average}$ are penalized, with the penalty depending on how far from the expected position they end up.
3. Unrelated items are bumped from submenus whose other items are cohesive.

In summary, we augment the SDP model with a consistency model to allow the optimizer to suggest item groups and form hierarchies. Optimizing consistency was our primary goal according to the results of the formative study. However, speed is also important for many users because even experienced ones do not always switch to hotkeys [19]. Moreover, improving speed does not always take place at the expense of groupings. For instance, an entire group can be moved. The optimizer must propose designs according to this dual objective.

OPTIMIZER DESIGN

In this section, we show that the problem of designing menu systems can be formulated as the *Quadratic Assignment Problem* (QAP). Developed in operations research, the QAP is the task of assigning n facilities to n locations on a factory floor. The QAP is NP-hard, and optima for problems with $n > 20$ cannot be expected to be found [33]. Analogously, menu design is about assigning items to slots in submenus in a menu tree. This observation is important because it allows us to adapt an existing optimization method known to work well for this problem.

Feedback latency (s)	Search space	Response	Algorithm
10^{-1}	n	Predicted command and menu performance	Exhaustive search
10^{0}	n^2	Optimal command placement	Exhaustive search
10^{1}	n^3	Menu swap/move suggestions	Exhaustive search
10^{0}	n^3	Improvement in menu designs	Heuristic generation
10^{1}	$(2n)!$	↓	Hill-climbing
10^{2}	$(2n)!$	↓	Ant colony optimizer
10^{3}	$(2n)!$	↓	

Table 1: Optimizer responses on different time scales. n = number of items in the menu system.

Problem Formulation: Menu System Design as the QAP

The *letter assignment problem* (i.e., virtual keyboard design) has been shown to be the QAP [4]. There, n letters must be positioned in m locations (keyslots) so as to minimize typing times for a language. The goal is to minimize the cost c_{ij} of typing letter j when starting from letter i, typically given by Fitts' law, and weighted by the probability p_{ij} in the digraph distribution. More formally, the goal is

$$\min_{\psi \in P(n,m)} \sum_{i=1}^{n} \sum_{j=1}^{n} p_{ij} c_{\psi(i)\psi(j)}, \tag{4}$$

where $C = [c_{sr}]$ is an m x m matrix and $P = [p_{ij}]$ is an n x n matrix; $P(n,m)$ is the set of all 1–1 (injective) mappings from {1, …, n} to {1, …, m}, corresponding to the placement of the letters in these locations; and, $\psi(i)$ is the location of letter i in the current solution $\psi \in P(n,m)$. Now, $p_{ij} c_{\psi(i)\psi(j)}$ is the *cost contribution* of assigning letter i to location $\psi(i)$ and letter j to $\psi(j)$. This problem can be formulated with a quadratic objective function [33]: Let x_{ij} be a binary variable that is 1 if letter i is assigned to location j, and 0 otherwise. Now, Eq. 4 is equivalent to

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} p_{ij} a_{kl} x_{ik} x_{jl}, \tag{5}$$

where a_{kl} is the cost of using keyslot l after keyslot k.

The design of menu systems is a QAP, too: n commands are assigned to slots in k submenus that form a tree. Three differences must be addressed for treating menu system design as the QAP. First, instead of transitioning from a previous command j in Eq. 4, SDP assumes each selection to start from the *same* position; no changes are needed to Eq. 4. Second, every additional factor in the cost function adds a sum term to Eq. 4. However, the formula collapses back to the quadratic form when one allows a larger cost matrix C. In the case of SDP, we have four coefficients: 1) the number of items in the menu, 2) the position of the target item, 3) the number of trials for that item, and 4) decision entropy in that menu. Hence, a four-dimensional cost matrix is necessary. We also need to consider that menus form *hierarchies* (trees). If there are k submenus and n items, they can be organized in $(n + k)^{n+k-2}$ ways into a tree. It can be shown that, although this vastly expands $P(n, m)$ (the set of mappings of items to locations in the tree), Eq. 4 still applies.

Multi-scale Interactive Optimization

We follow the *multi-scale approach* presented in Table 1: The goal is to generate immediate responses to the designer's edits but improve them when more time is given:

- **Item-level responses in 10^{-1}–10^{1} seconds**: The effects of edits on user performance are computed by an exhaustive search in $O(n^2)$. User Performance Graph and item-level feedback for a case of n = 25 takes about 600 ms on a normal laptop. Move/Swap Suggestions are computed in $O(n^3)$. To provide first suggestions rapidly, we start computation with the most frequent items. For a description of these UI features, please see the next section.
- **Rapid menu-system-level responses in 10^{0}–10^{1} s**: First, a *heuristic menu generator* provides the initial suggestions by arranging commands in a frequency-based order and creating menu hierarchies of depth 3 with a minimum of three commands per submenu. With optimization for consistency, related commands are also moved to be together. A *greedy hill-climbing algorithm* is initialized by the best designs found. It improves them by generating h x 10 designs by random local edits and taking the h best solutions for the next iteration (we use h=3). It runs for about 30s before handing over to the ant colony.

The Ant Colony Optimizer: 10^2–10^3 seconds

Although the greedy search produces reasonable solutions rapidly (see below), they are limited to a local optimum. To search globally, we apply a method that is known to work particularly well for the QAP. *Ant colony optimization* (ACO) [7] is based on the biological metaphor of an ant colony foraging for food. Multiple searchers cooperate to produce solutions according to a memory of past solutions.

Implementation for Menu Systems

To our knowledge, ACO has not yet been applied to menus. We use the *Max-Min Ant System* (MMAS) variant [33]. It ensures the exploration of the search space by imposing upper and lower bounds τ_{max} and τ_{min} (respectively) to the pheromone level. To increase efficiency, only the best ants add pheromone. The pseudo-code for MMAS is:

```
1.  Initialize pheromone trails to τmax
2.  WHILE (not converged) DO
3.      FOR each ant
4.          FOR each element i
5.              PlaceElement i
6.      UpdateTrails
```

In initialization, pheromone trails are set to τ_{max}. In each iteration, the odds of placing command i in slot j are

$$p_{ij} = \frac{\tau_{ij}(t)}{\sum_{(i,j)\in N(s^p)} \tau_{il}(t)}, \tag{6}$$

where $N(s^p)$ is the set of feasible placements and $\tau_{ij}(t)$ is the pheromone value for location j. In the subsequent iteration, pheromone is updated for the best ants according to

$$\tau_{ij}(t+1) \leftarrow \rho\, \tau_{ij}(t) + \Delta\tau_{ij}^{best}, \tag{7}$$

where $\rho < 1$ is the *evaporation factor*, $\Delta\tau_{ij}^{best} = 1/f(S^{best})$, and S^{best} is the best global solution.

Hierarchies complicate optimization. Previous work has made simplifications to make the problem more tractable (see [14,23]) but did not explore unconstrained hierarchies or use a realistic cost function. To represent hierarchies, we use *pointers* to mark the locations of submenus in the parent menu. We also found the following techniques useful:

1. **Compression**: Because the ants randomly place commands, sparse placements arise. We "compress" menus by removing empty slots before pheromone deposit.
2. **Penalties for illegal structures**: Ants generating menus wherein some commands are not accessible are penalized by a decrease in pheromone values for their solutions.

3. **Two-phased search**: We first develop pheromone structures for valid hierarchies (all items included) and only then move on to focus on command assignment.
4. **Pheromone diffusion**: Pheromone deposition is *diffused* 1) for items close in consistency score and 2) to positions in neighboring submenus.

Multi-Objective Optimization

We implemented three scalarizations that collapse the normalized objectives into a single variable: weighted sum, weighted Tchebycheff scalarization, and weighted square root sum. In our experience the weighted square root sum offers the most stable performance in the Pareto frontier.

Parameter Setting

Parameters were set by testing 1,152 combinations for the number of ants, ρ, the number of winners, and diffusion heuristics. Each was tested for 5,000 iterations for the Firefox problem (see below). The winning configuration was further fine-tuned manually for a larger set of problems.

USER INTERFACE DESIGN

MenuOptimizer augments QtDesigner, a C++/Qt graphical user interface builder used by novice and expert software designers alike. MenuOptimizer can be used exactly like the regular QtDesigner. A designer can manipulate a menu system by adding, removing, and moving items, groups, and submenus by direct manipulation. The optimizer-produced features are reviewed here in three main categories. An overview of the UI is provided in Figure 2.

Model-based Feedback for Design Choices

Metrics and visualizations that summarize the current status of an interface arguably help designers to take into account elements that can be difficult to perceive; they inform and justify design choices; and they can be referred to when one is communicating a design to others [32]. MenuOptimizer provides SDP-based feedback at two levels:

Items: The user performance and selection frequency for each item are visually represented in a place underneath it (see Figure 1A). The width of the line indicates frequency: more frequent commands have longer lines. The color gradient indicates user performance over time for the item.

Menu system: User Performance Graph (see Figure 2E) shows the average selection times for each user over time. The Template Gallery plot in Figure 2C shows the currently chosen menu against suggestions by the optimizer.

Suggestions for Improvement

A major feature of MenuOptimizer is its improvement suggestions. A key difference from previous systems [32] is that *several* suggestions are offered both locally and globally (a multi-scale approach). Menu-level suggestions are updated whenever the optimizer finds a better solution.

Menu system: Template Gallery (Figure 2C) is a key feature of MenuOptimizer: It updates the optimizer's suggestions (templates) for menu systems that have a higher objective score than the present design does. Three features assist the designer in decision-making and exploration:

1. *Pareto front plot*: Templates are organized in a scatter plot to emphasize the tradeoff between *selection time* (y-axis, predicted by SDP) and *consistency* (x-axis, from Eq. 2). The plot shows the evolution of the *Pareto optimality frontier*: the frontier where it is not possible to improve for one objective without being worse for the other.
2. *Distance* of a template to the menu system currently edit-

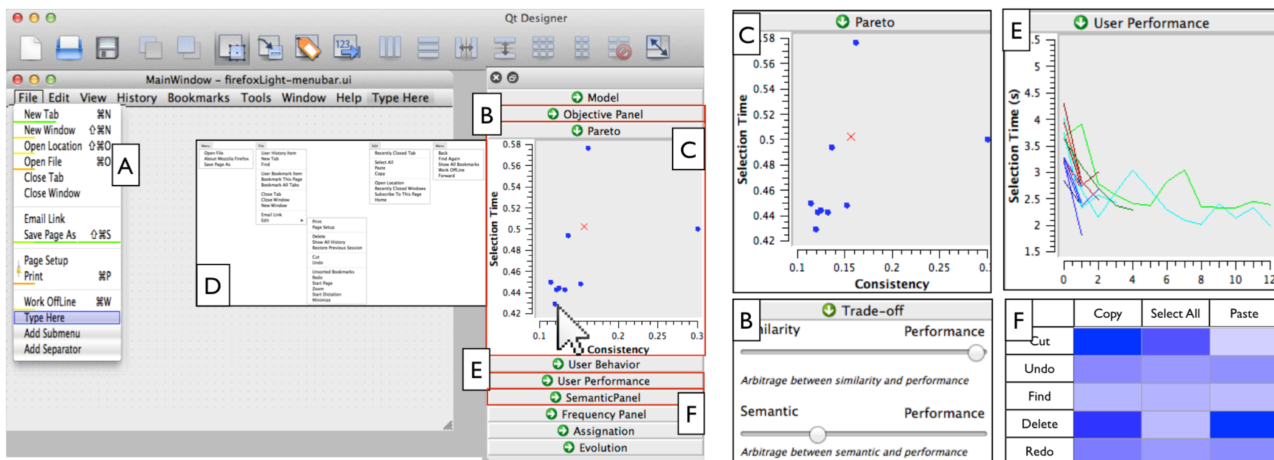

Figure 2: MenuOptimizer integrates optimizer-based assistance into the QtDesigner interface: A) The menu is edited. B) Objective Panel allows the designer to set the weights for three optimization objectives: performance, consistency, and similarity to the present menu. C) Template Gallery shows new menu system suggestions (•) on a Pareto front plot with user performance (y-axis) and consistency (y-axis) and (D) a preview of the menu. The current menu system (x) is shown to aid in comparison. E) User Profile Graph shows predicted performance for each user. F) Association Panel allows editing of assumptions about how closely related command pairs are.

ed is indicated by the size of the marker. A small circle indicates that only a few changes (moves of items) distinguish the two designs. This dimension is useful for designers who may at a later stage of design, want to constrain improvement suggestions to local changes.

3. *Preview*: The template content is previewed when the mouse is hovered over the marker. By clicking, the designer replaces the current menu with the proposed template. Figure 2D shows the preview.

Submenus: MenuOptimizer provides Move/Swap Suggestions for a submenu (Figure 1D) and between submenus. The arrows indicate where to move an item to improve the user's average selection time. Swap suggestions are marked with a double-headed arrow. A box adjacent to the arrow indicates the percentage gain. So as not to overload the display, we display only the three best suggestions at a time.

Items: Designers can also select a command in the menu (click with timeout), and the system renders a colored line next to each item to indicate whether that position would improve or worsen the current menu design.

Separator assignment: On the basis of a predefined threshold for consistency score $C(i, j)$, MenuOptimizer automatically assigns *separators* that mark groups of adjacent items. Computation is done in linear time by comparing consistency scores for adjacent items. Examples of outcomes are shown later in the paper, in Figure 4.

Hotkey Assignment: Upon request, MenuOptimizer assigns hotkeys. Because this important feature is not taken into account by the predictive model, we use a rule-based approach: First, we assign the most common command–hotkey associations (e.g., Ctrl+S for Save), to maintain consistency with expectations. We then assign hotkeys in frequency-based order, using the first letter of the command. Modifiers (e.g., Shift) are added if collisions occur.

Title suggestions: We use the command database to derive candidate *titles* for submenus. For each submenu, we choose the label that is shared by the most commands (voting) as a title. The strategy is conservative: In the case of multiple candidates, no title is suggested to avoid forcing the designer to make corrections. An example is shown in Figure 4.

Editable Objectives and Assumptions

Key aspects of the objective function are directly editable. This forces designers to explicate their assumptions and let them see the implications of these assumptions in real time as the optimizer updates its search problem and visualizations accordingly. It allows them to test different design ideas and explore the robustness of their design with different assumptions about users.

User profiles: The optimality of a menu depends on the target users and their assumed selection behavior. MenuOptimizer allows an arbitrary number of users, each with unique distribution of selections. Internally, a user is represented as a sequence of item selections. User Profile Panel lets designers add users and choose different frequency distributions for commands by clicking and dragging their distribution graphs (Figure 3). An additional feature is the loading of distributions from log files: CSV-formatted log files rendered as distributions that are also directly editable.

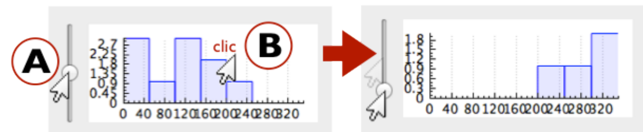

Figure 3. User Profile Panel shows directly editable distributions of item selections by users. Designers can change the frequency (A) and the distribution (B).

Locking of item groups: At some point, the designer may want to keep certain items together. Locks (Figure 1F) can be used to constrain items to groups [30] such that subsequent suggestions by the optimizer keep them intact. This mechanism also accelerates optimization because the locked groups reduce the size of the search space.

Submenu optimization: The user can ask the optimizer to optimize only a submenu. The submenu is made inactive for about 10 seconds, during which the optimizer searches for local improvements to this submenu. The suggested improvement can be previewed in place before acceptance.

Association Panel: Designers can edit the automatically suggested co-occurrence scores for command pairs to affect the groupings of items (Figure 2F). They can mark two commands as "Strongly related," "Related," "Slightly related," or "Unknown" simply by clicking the cells.

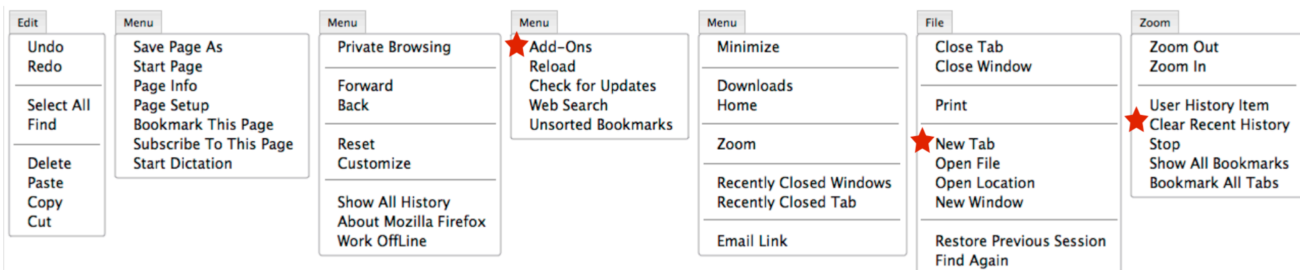

Figure 4: A menu system for the Firefox case proposed after 20,000 iterations (~15 minutes). The objective function had a weight of 75% for consistency and 25% for user performance (asterisks indicate the 3 most frequent items from 50 users' logs).

IMPLEMENTATION

To maintain a high level of interactivity in the editing interface, we use a client–server architecture wherein the optimizer runs on a second computer and communicates its suggestions over XMPP. The optimizer is implemented in Python. Ants are parallelized to accelerate the search. The optimizer is fast enough to run on a regular laptop. We use a MacBook Pro 2.8 Ghz with 8 GB of RAM for all examples reported in this paper.

TECHNICAL ASSESSMENT

This section reports on qualitative and quantitative evaluations of the optimizer system. Because the desirable tradeoff between quality and speed is designer- and application-dependent, we report tests for short-to-mid-term performance and also a benchmark for long-term performance.

Case Studies: SeaShore and Firefox

To assess the quality of the suggestions of the optimizer, we show outputs for two cases: a browser (Firefox) and an image editor (Seashore). Both have ~50 items. The goal for Firefox was to improve its menu design, given logs of real users [10], from which we chose a random sample of 50. Seashore emulates the case wherein a designer is creating a menu system from scratch. The input is a list of commands and our estimates of user behavior. To avoid bias, we use a database for consistency scores (Eq. 2) that does not include the two applications. Output for Firefox after 20,000 iterations (or 15 minutes on a laptop), is presented in Figure 4 (not handpicked). We can observe the following:

- A system with seven submenus was created with no sub-submenus.
- There are many logical groups that are consistent with other applications (e.g., Undo, Redo). The group Delete-Paste-Copy-Cut is interesting. The four are grouped together (consistency) so that users can find them. Moreover, Paste has been moved up because it is the most frequently of the four.
- Because 15 minutes is not sufficient to explore the entire search space, some groups are not optimal (e.g bookmarking commands)
- Three menus were given a title by the optimizer. The others would require edits by the designer.
- Because consistency was preferred (75%) for this case, some frequently used items are not prominently placed.

We give examples for Seashore in *Appendix*. The observations are similar and show that the optimizer is not overcalibrated to the Firefox problem. Runtimes for Seashore are about one third due to fewer users.

Temporal Performance

Parameters of the optimizer system were originally calibrated to a scenario allowing a few minutes. To evaluate its ability to improve designs over time, we performed 25 runs of 10,000 iterations (~10 min) for the two optimizers in the system: the greedy searcher and ACO. Seashore and Firefox were used as cases. Figure 5 shows the average temporal performance of the ant colony optimizer against greedy search. A crossover is seen at around 300 iterations (~20 s). After that, ACO is preferable.

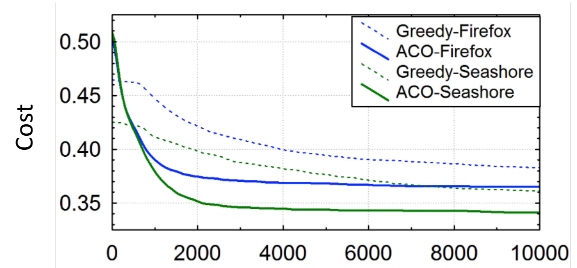

Figure 5: Temporal performance of the optimizer system for the Firefox and Seashore case. In our test setup, 10,000 iterations takes about 10 minutes.

However, the optimizer should, in theory, be useful for different classes of interface problems as long as they can be subsumed to the QAP. We therefore tested it in the context of virtual keyboards. This test is indicative of the performance of ACO when calibrated for a long-term task. Our benchmark is the *Metropolis keyboard* optimized in [34]. We created a version of ACO that uses exactly the same objective function: Fitts' law parameters for stylus movement and digraph frequencies. Parameters were set in a separate trial; we then conducted runs of 50,000 iterations.

Figure 6 shows the distribution of best-found solutions, in comparison to the Metropolis keyboard. The best-found keyboards from ACO were systematically better than Metropolis. The best, *JUSTHCI*, named for the sequence of the first letters on the second row, improves on Metropolis's words per minute (WPM) by 1.8 percent.

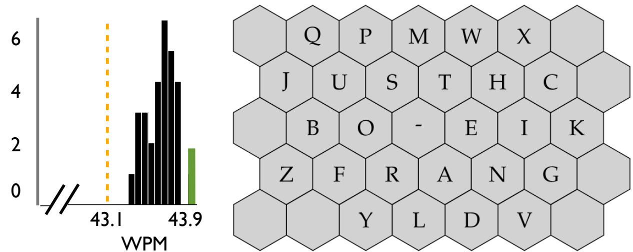

Figure 6: Results of long-term optimization benchmarked against the Metropolis keyboard [34]. The JUSTHCI keyboard (43.9 WPM) outperforms Metropolis (43.1 wpm).

PRELIMINARY USER STUDY

The focus of our user study was on the *joint performance* of the designer and MenuOptimizer. We compared novice designer performances with and without MenuOptimizer in the task of designing medium-complexity (37 and 52 items for Arduino and Seashore, respectively) menus, with a focus on the effect of the functionality that we hypothesized would help the most: using the menu design suggestions appearing in Template Gallery as a basis for editing a menu. We were also keen to understand *how* users use Template Gallery and learn about their rationale for edits.

Participants. Twelve participants without background in menu design (5 females) aged 22 to 44 were recruited from the local university. They received a compensation of 15€.

Experimental design: The study follows a 2 x 2 within-subjects design with *Condition* (MenuOptimizer vs. QtDesigner) and *Task* as the factors (see below). The order of the two factors was counterbalanced.

Tasks. The stated task was to organize commands *in a usable menu system such that its users will easily "understand it" and be able to "quickly select commands."* To avoid order effects, two versions were created: Seashore and Arduino. The first had 52 commands related to image manipulation, the second 37 concerning electronics programming. Tasks were presented on a sheet of paper showing a list of commands with definitions and frequencies.

Procedure. Participants were first instructed in the task and training was carried out with a predefined menu (different from those used during the experiment) for 15 minutes. In both conditions, a single menu was initially filled in with all the commands to let participants focus on organizing commands rather than on entering their names in the interface. Participants could freely move, insert, and remove items and submenus. With MenuOptimizer, they could also use in-place item-level feedback, User Performance Graph, Evolution Graph, and Template Gallery. Participants had 25+15 minutes to perform the task with each system. A semi-structured interview was performed after the study.

Results

For statistical analysis, we used 2 (Condition) x 2 (Task) ANOVA. The effect of *Condition* on time on task was not significant (M=23 mins). Users in both conditions spent about the same amount of time editing their menus. Users were also equally satisfied with their designs in both conditions: the difference in quality ratings between QtDesigner menus (mean Likert rating: 4.4, σ=1.4) and MenuOptimizer menus (M=4.1, σ=1.5) was not statistically significant. However, users made significantly fewer *edits* (37.5%) when using MenuOptimizer than with QtDesigner ($F_{1,20}$=8.15, p =.01), as shown in Figure 7.

User-produced menus were evaluated against model-based predictions of 1) selection time and 2) consistency score (see Figure 8). The same statistical test was used as above, but one menu design was excluded due to a technical problem. First, designs produced with the help of MenuOptimizer (M=2.36s, SE=0.018) are faster than QtDesigner (M=2.41s, SE=0.017) but the statistical test was borderline-significant ($F_{1,19}$ =4.30, p=0.052). Second, the consistency score of MenuOptimizer designs is the same than the one of QtDesigner (both M=0.088, SE=0.005). However, an interaction effect was observed, $F_{1,19}$=5.37, p=0.032. Menus designed with MenuOptimizer had slightly better consistency scores for the more complex task (Seashore; M=0.078 versus 0.094, respectively), whereas scores were worse for the simpler task (Arduino) (M=0.099 versus 0.083). (lower score is better.)

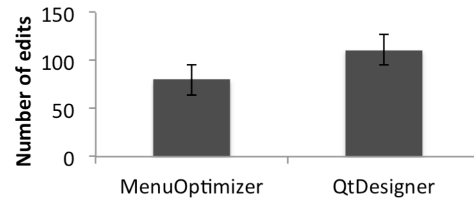

Figure 7: Number of edits required for completing the task.

Observations

Workflow. All users adopted a top-down cascade strategy. They started by using the template gallery. Then, they performed sub-menu level optimization and edits.

Template Gallery. Template Gallery was used differently by different users. Some did not have the patience to wait for good designs and instead selected the first available template and started editing it. Some selected several templates before starting to edit, sometimes only guided by scores in the Pareto plot. Finally, some users used the templates only for inspiration. For instance, one user intensively compared the current menu with suggested templates.

Item Feedback. Users reported that the item-level performance feedback (Figure 1A) is useful because it reminded them to focus also on the performance objective, and it motivated them to move frequent items to the top of groups or menus. However, most users did not understand the color coding. According to the SDP model, the selection time of an item decreases as a function of repetitions rather than as a function of position in the menu. It is, therefore, possible to have a green line for frequent items even if they are deep in the hierarchy. This was found to be confusing.

Rationale for edits. Most users mentioned focusing on groupings rather than selection time because this is the most important objective. One participant mentioned focusing on groupings here because the hierarchy was "quite small" but would focus more on selection time for large ones. All users except one started by grouping items and only afterwards, if at all, focused on selection time.

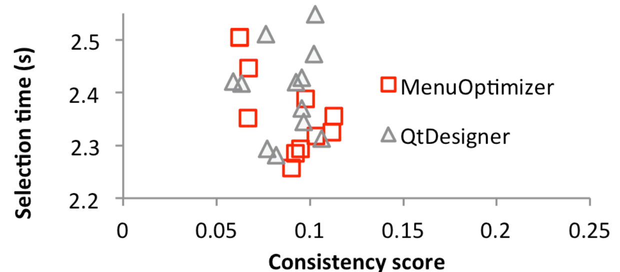

Figure 8: Selection time and consistency score for user-produced designs. Bottom left designs are better.

Discussion

This study is a first step to understand interactive optimization. Our experimental design had to adhere to the constraint of lab studies, and only 15 minutes of practice was given to explain MenuOptimizer. However, novice designers already produced designs they were equally happy with significantly less effort.

The study also provides useful recommendations for the design of MenuOptimizer and more generally interactive optimization systems. First, item feedback should stay simple enough that it does not introduce confusion. For instance, encoding only frequency would probably have been sufficient. Second, the optimizer should even quicker provide a first acceptable design to start with because many users do not have the patience to wait for better designs. Third, many designers seemed to favor the quality of groupings over performance in the beginning stages of design. Our heuristic first-response optimizer worked better for the performance objective but there is room to improve on the grouping aspect to make the suggestions more readily usable. Finally, while MenuOptimizer lets designers compare menu designs with produced templates, it seems important to provide more comparative tools. For instance, by comparing existing designs with previous ones at different levels (menu systems, submenus).

This study also has several limitations requiring additional evaluations and, in particular, longitudinal studies. The time for designing a menu was limited to 30 minutes. With more time, designers can probably better handle functionalities and refine their strategies. Moreover, the evaluation was performed by novice users with medium size menus. More work is needed to understand the needs and behaviors of professional designers. Finally, the 24 produced menus should be experimentally evaluated by final users to evaluate the real impact of MenuOptimizer on user performance.

CONCLUSION AND FUTURE WORK

This paper has investigated interface and optimizer design for the *interactive optimization of menus*. We extended the predictive model of menu performance by introducing a metric for consistency that allows the generation of menu hierarchies and item groups. We proposed MenuOptimizer, an interface providing feedback methods and proactive suggestions to operate menus at different levels of granularity: items, menus, and menu systems. This changes the design process. Instead of organizing items one by one, the designer specifies the problem and starts the design process on the basis of a template, and the system assists the designer in further edits. We have shown that the menu design problem can be formulated in terms of a QAP problem and we proposed an adaptation of the ant colony meta-heuristic that better matches the requirements of interactive and iterative design.

There are two kinds of evidence for the usefulness of this approach: 1) A user study shows that novice designers generate designs that equally satisfy them but with 38% less editing. 2) The temporal response curve of the optimizer system better suits interactive editing tasks than the prevailing "fire or forget" approach.

Finally, we see several opportunities to improve our approach.

Menu systems. The present work is limited to linear menus and should be extended to cover hotkeys, ribbons and palettes. We are currently working on a predictive model of learning and adoption of hotkeys.

Semantics. "Semantics" is a strong factor in menu use. Our approach was to use the consistency of item collocation as a secondary optimization objective. Although it allows grouping items and setting a hierarchy, it does not take into account the fact that semantics is actually a cognitive factor (or many). Ideally, semantics would be part of the predictive model. Improvements are possible via integration of knowledge of relationships like synonyms, antonyms, or tools such as Wordnet or LSA. Previous work in modeling users' navigation in hierarchies could also provide a useful starting point for this work [18].

Predictive model. More generally, developing predictive models is the key to understanding user behavior and developing optimization methods. Existing models should be empirically validated in the field: The question is still open whether behavior predicted by such models actually hold for end-users in longer-term use.

Scalability. Our optimizer presently deals well with medium size menu systems (~50 items). We are yet to see how it scales up as the number increases to hundreds. We believe that for such cases, both the optimizer and the interface will need to better guide the designer to partition a problem. For instance, the Layer menu in PhotoShop has 30 items, and is a design problem of its own.

ACKNOWLEDGMENTS

We thank R. Kirsten, H. Du, A. Roudaut, C. Coutrix, P-O. Kristensson, A. Jameson, M. Löchtefeld, K. Hornbaek, A. Krueger, H-J. Mueller for help and comments. This work has partially been funded by the Cluster of Excellence Multimodal Computing and Interaction within the German Federal Excellence Initiative.

REFERENCES

1. Ahlstrom, D. Modelling and improving selection in cascading pull-down menus using Fitt's law, the steering law and force fields. ACM CHI'05. (2005), 61-70.
2. Ahlstrom, D., Cockburn, A., Gutwin, C., Irani, P. Why it's quick to be square: modeling new and existing hierarchical menu designs. ACM CHI'10. (2010), 1371-1380.
3. Anderson, D., Anderson, E., Lesh, N., Marks, J., Mirtich, B., Ratajczak, D. and Ryall, K. 2000. Human-guided simple search. AAAI'00, 209-216.
4. Burkard,R.E.,and Offermann,D.M.J .Entwurf von schreibmaschinentastaturen mittels quadratischer zuordnungsprobleme. Zeitschrift für Operations Research 21, 4 (1977), B121–B132.
5. Cockburn, A., Gutwin, C. and Greenberg, S. A predictive model of menu performance. ACM CHI '07, (2007), 627-636.

6. Cross, N. Expertise in design: an overview. Design studies 25, 5 (2004), 427–441.
7. Dorigo,M.,Birattari,M.,and Stutzle,T.Ant colony optimization. Computational Intelligence Magazine 1, 4 (2006), 28–39.
8. Eggers,J.,Feillet,D.,Kehl,S.,Wagner,M.,and Yannou,B. Optimization of the keyboard arrangement problem using an ant colony algorithm. European Journal of Operational Research 148, 3 (2003), 672–686.
9. Fogarty, J., Hudson, S. E. GADGET: a toolkit for optimization-based approaches to interface and display generation. ACM UIST'03. (2003), 125-134.
10. https://testpilot.mozzilalabs.com/testcases/menu-item-usage/
11. Gajos, K. and Weld, D. S. SUPPLE: automatically generating user interfaces. ACM IUI'04, (2004), 93-100.
12. Gajos K., Wobbrock, J. O. and Weld, D. S. Automatically generating user interfaces adapted to users' motor and vision capabilities. ACM UIST'07, (2007), 231-240.
13. Gajos, K, Wobbrock, J. O and Weld, D. S. Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces. ACM CHI '08, (2008), 1257-1266.
14. Goubko M. V. and Danilenko A. I. An automated routine for menu structure optimization. ACM EICS'10, (2010), 67-76.
15. Gould J. D. and Lewis, C. Designing for usability: key principles and what designers think. Commun. ACM 28, 3, (1985), 300-311.
16. Horvitz, E.. Principles of mixed-initiative user interfaces. ACM CHI '99, (1999), 159-166.
17. Kim, W. C. and Foley, J. D. Don: user interface presentation design assistant. ACM UIST '90, (1990), 10–20.
18. Kitajima, M., Blackmon, M., Polson, P. A comprehension-based model of web navigation and its application to web usability analysis. *People and Comp. XIV*, Springer (2000), pp. 357-373.
19. Lane, D., Napier, A., Peres, C., and Sandor, A. The Hidden Costs of Graphical User Interfaces: The Failure to Make the Transition from Menus and Icon Tool Bars to Keyboard Shortcuts. IJHCS. 18 (2005), 133--144.
20. Lee, E. MacGregor, J. Minimizing user search time in menu retrieval systems. Hum. Fact. 27, 2, (1985), 157-162
21. Light,L.W.,and Anderson,P.G. Typewriter keyboards via simulated annealing. AI Expert (1993).
22. Malacria, S., Bailly, G, Harrison, J., Cockburn, A. and Gutwin, C. 2013. Promoting Hotkey use through rehearsal with ExposeHK. ACM CHI '13, (2013),, 573-582.
23. Matsui S. and Yamada, S. Genetic algorithm can optimize hierarchical menus. ACM CHI'08, (2008) 1385-1388.
24. Oulasvirta,A. et al. Improving two-thumb text entry on touchscreen devices. ACM CHI'13, (2013).
25. Paap,K.,and Cooke,N.Designing menus,(1997).
26. Poltrock, S. E. and Grudin, J. Organizational obstacles to interface design and development: two participant-observer studies. ACM TOCHI'94. 1, 1, (1994) 52-80.
27. Quiroz, J. C., Louis, S. J. and Dascalu S. M. Interactive evolution of XUL user interfaces. ACM GECCO '07, (2007), 2151-2158.
28. Raneburger, D., Popp, R., and Vanderdonckt, J. 2012. An automated layout approach for model-driven WIMP-UI generation. ACM EICS'12, 91-100.
29. Rao, S. S. Engineering Optimization: Theory and Practice, (2009).
30. Scott, S. D., Lesh, N., Klau, G. W. Investigating Human-Computer Optimization. ACM CHI'02. (2002), 155-162.
31. Sears. A. 1993. Layout Appropriateness: A Metric for Evaluating User Interface Widget Layout. IEEE Trans. Softw. Eng. 19, 7 (July 1993), 707-719.
32. Sears, A. AIDE: a step toward metric-based interface development tools. ACM UIST '95, (1995), 101-110.
33. Stutzle,T.,and Dorigo,M. ACO algorithms for the quadratic assignment problem. New Ideas in Optimization (1999), 33–50.
34. Zhai, S., Hunter, M., and Smith, B. A. The metropolis keyboard - an exploration of quantitative techniques for virtual keyboard design. ACM UIST '00, (2000), 119-128.
35. Zhai,S., Sue,A.,and Accot,J. Movement model, hits distribution [illegible] ACM CHI'02, (2002), 17–24.

Menu: Hide Status Bar, Hide Layers, Hide Options Bar, Rotate, Hide Point Information

File: Import | Page Setup, Print | Export, ★Save, Duplicate | Open, New | Close, Select All, Resolution, Delete

Edit: Save As, ★Paste | Redo, Undo | Cut, Copy | Zoom Out, Align Vertical Centres

Menu: Scale Layer, Flatten Image | Flip Image Vertically, Flip Image Horizontally | ★New Layer, Revert

Align or Distribute: Align Left, Duplicate Layer | Align Bottom, Align Right, Align Top | Rotate 90 Right, Align Horizontal Centres

Arrange: Bring Forward, Copy Image, Rotate 90 Left | Bring to Front, Send to Back | Scale Image, Send Backward

Menu: Zoom In, Delete Layer, Select Inverse | Edit All Channels, Edit Primary Channels | Select Alpha, Edit Alpha Channel

Consistency: 75% - Performance: 25%

Menu: ★Save, New | Export, Import | Duplicate, Zoom In

Edit: Cut, ★Paste | Undo, Duplicate Layer, Copy | Align Right, Align Bottom

Menu: Zoom Out, Align Left, ★New Layer, Rotate 90 Left

Arrange: Align Vertical Centres, Align Horizontal Centres | Bring Forward, Send Backward, Send to Back, Bring to Front | Scale Image, Save As

Edit: Redo, Flip Image Horizontally, Resolution, Align Top, Select All, Select Inverse, Hide Layers, Scale Layer

Menu: Select Alpha, Edit Primary Channels, Rotate 90 Right, Rotate, Flatten Image, Edit Alpha Channel, Flip Image Vertically, Copy Image, Edit All Channels, Delete Layer

Menu: Close, Open, Print, Delete, Revert | Hide Options Bar, Hide Status Bar | Hide Point Information, Page Setup

Consistency: 25% - Performance: 75%