Int. J. Human–Computer Studies 151 (2021) 102624

Contents lists available at ScienceDirect

International Journal of Human - Computer Studies

journal homepage: www.elsevier.com/locate/ijhcs

# Foraging-based optimization of menu systems

Niraj Ramesh Dayama [*,a], Morteza Shiripour [a], Antti Oulasvirta [a], Evgeny Ivanko [b], Andreas Karrenbauer [c]

[a] Aalto University, Finland
[b] Institute of Mathematics and Mechanics, Ural Branch, Russian Academy of Sciences, Ural Federal University, Yekaterinburg, Russia
[c] Max Planck Institute for Informatics, Germany

ARTICLE INFO

*Keywords:*
Combinatorial optimization
Integer programming
Computational design
User interfaces
Menu systems
Information foraging
Human-computer interaction

ABSTRACT

The problem of computational design for menu systems has been addressed in some specific cases such as the linear menu (list). The classical approach has been to model this problem as an assignment task, where commands are assigned to menu positions while optimizing for users' selection performance and grouping of associated items. However, we show that this approach fails with larger, hierarchically organized menus because it does not take into account the ways in which users navigate hierarchical structures. This paper addresses the computational menu design problem by presenting a novel integer programming formulation that yields usable, well-ordered command hierarchies from a single model. First, it introduces a novel objective function based on information foraging theory, which minimizes navigation time in a hierarchical structure. Second, it models the hierarchical menu design problem as a combination of the exact set covering problem and the assignment problem, organizing commands into ordered groups of ordered groups. The approach is efficient for large, representative instances of the problem. In a controlled usability evaluation, the performance of computationally designed menus was ∼ 25% faster to use than existing commercial designs. We discuss applications of this approach for personalization and adaptation.

## 1. Introduction

*Menu systems* are among the most prevalent user interfaces, offering a compact, extensible, and familiar means to access functionality. Some popular menu techniques are known as linear, tabbed, hierarchical, cascading, context, drop-down, ribbon, and toolbar menus. Designers frequently design menus, but their design remain challenging (Bailly et al., 2017). Multiple objectives must be addressed, including speed and accuracy of selection, learnability, satisfaction, efficacy, suitability for different devices, and accessibility (Bailly et al., 2017). Also the involved design spaces can be large. Professional software, such as for photo-editing or 3D modeling, involve menus comprising of in excess of fifty commands. It is not surprising that professional designers report menu design being "very difficult" and having to resort to trial and error (Bailly et al., 2013).

This paper contributes to algorithmic methods for generating and refining menu systems. Our goals are (1) to improve the quality of generated menus and (2) support a larger number of commands (over 20 and up to 100) than previous research. Generally, larger menu systems need to utilize some type of *hierarchical organization*, achieved by techniques such as tabbing, groups, folding, cascades, or sub-menus. Some promising advances notwithstanding, computational design of such hierarchical menu systems is still an unsolved problem. While there has been sustained research interest since the 1980s (Ahlström, 2005; Bailly et al., 2014; Brumby and Howes, 2004; Byrne et al., 1999; McDonald et al., 1983; Mehlenbacher et al., 1989; Norman, 1991; Paap and Cooke, 1997; Sears and Shneiderman, 1994), no method has been offered that can automatically generate demonstrably usable and well-structured menus with a larger number of commands. Professional designers would appreciate computational support on the matter (Bailly et al., 2013).

Any algorithm for menu design will need to represent essential aspects of human behavior to be successful. Two challenges stand out: (1) the size of the search space and (2) lack of valid but computationally efficient evaluative (objective) functions. Firstly, the search spaces involved in menu design are exceedingly large: $n$ commands can be organized into a linear menu in $n!$ unique ways and into a hierarchical menu in an exponentially larger number of unique ways. If we consider

* Corresponding author.
*E-mail address:* niraj.dayama@aalto.fi (N.R. Dayama).

https://doi.org/10.1016/j.ijhcs.2021.102624
Received 13 July 2020; Received in revised form 21 January 2021; Accepted 21 February 2021
Available online 2 March 2021

multiple different tabs and also potential sub-groups within tabs, the space explodes further. Standard software applications commonly comprise dozens of commands; professional applications may extend to hundreds of commands. The second issue, evaluative functions (objective functions), is even more challenging: the relevant literature has not yet identified any effective evaluative function that captures essential human factors mathematically. A well-known design objective – given by Fitts' law – characterizes the efficiency of selecting a command with a pointing device. Using Fitts' law leads to placing frequently accessed commands closer to the top of the menu (Bailly et al., 2013). Another objective investigated is related to grouping of items: placing associated commands near each other can make it easier to find them (Bailly et al., 2013). The association here can be based on distributional semantics (e.g., pairwise word associations) or on statistical co-occurrence in other menu designs. Another factor in good menu design is the perceived balance between depth (tree) and breadth (Norman, 1991). User expectations are a fourth consideration: Users may have preconceived notions, formed through exposure to prior designs, of where in the menu certain commands should be found. For instance, *About* and *Help* may be expected in the last tab. To effectively apply computational methods for hierarchical menu design, a robust mathematical model and problem definition are needed that encompasses such considerations and yet allows efficient algorithmic solutions.

This paper presents a novel combinatorial optimization approach to the design of menu systems. It describes a mixed-integer programming (MIP) formulation to handle realistic-sized task instances. It contributes a mathematical formulation of the menu design problem that (1) captures essential human aspects of menu navigation and (2) the decision problem in an efficient manner. It produces well-structured and usable menu designs when input data is provided for: (a) frequency of usage of individual commands and (b) mutual (semantic) association metrics for any pair of commands. While previous research has mostly resorted to meta-heuristic techniques – which are often based on randomization – our MIP approach guarantees optimality and provides mathematical estimates for bounds indicating the quality of a solution. For any candidate solution, it is possible to compute bounds that indicate how far the current design is from the global optimum.

Two technical contributions are made. The first lies in a new representative model of hierarchical menus. Previous approaches used an assignment-based formulation (Bailly et al., 2013). Over several studies of objective functions, we discovered that assignment alone does not sufficiently represent the organization of individual elements in larger entities such as groups or tabs. In particular, it leads to frequency-ordered groupings, at times ignoring how well the command placed at the top represents the rest of the menu. Hence, the topmost items are not necessarily semantically indicative of what the menu contains. In contrast, our formulation assigns each command to a group and then to a tab while also organizing (ordering) these for faster access. In other words, both position (assignment) and grouping are naturally addressed in this new formulation, unlike in previous work that only considered assignment. Moreover, both assignment and grouping can be handled with a single objective, here based on the foraging theory, which eliminates the need to set calibration weights. To capture this critical aspect of how users navigate menus, we posit the design problem as a variant of *the exact set covering problem*. Formally, the set covering problem is defined as follows: Given a finite set $\mathbb{S}$ and a list of some (not necessarily all) subsets of $\mathbb{S}$, the intent is to find the minimal sub-collection of disjoint sets such that all elements of $\mathbb{S}$ are covered exactly once. This covering problem precisely captures our intention of organizing the given menu commands into disjoint groups. The new objective yields organized *groups of groups* with clear inter-group boundaries. This avoids the need to compute group boundaries *post hoc* heuristically as in previous work using the assignment-based approach (Ahlström, 2005; Bailly et al., 2013).

The second contribution is a new evaluative function based on *information foraging theory* (IFT) (Fu and Pirolli, 2007; Pirolli, 2007; Pirolli and Card, 1999). Previous literature focused on minimization of selection time (Ahlström, 2005; Ahlström, 2005; Bailly et al., 2017; Card et al., 1980) and maximization of associativity among commands (Bailly et al., 2013). For example, *MenuOptimizer* used Fitts' law and a statistical consistency metric measuring structural similarity of assignments to other menus (Bailly et al., 2013). Neither component specifies how the grouping of elements affects user behavior. Our contribution is a mathematically efficient formulation of IFT, which is made feasible for existing mixed integer programming solvers. The new IFT-based objective enables assessing *search performance* in the case of groups of groups, which in our case are command groups (with separators between them) organized into tabs. Earlier work with IFT used it for modeling how users choose link panels (Pirolli and Fu, 2003).

In the case of hierarchically organized menus, it offers a quantitative model of a rational but time-limited agent navigating a hierarchy composed of patches. The agent decides whether to continue exploring the current set of commands (patch) or instead abandon/skip this set in favor of the next. Intuitively, when used in an optimizer, it evaluates and minimizes also the time *wasted* by the user in the irrelevant parts of the menu. This results in positioning of semantically indicative items toward the top of the menu. In practice, this is achieved by three means: (1) The optimizer forms groups that enable users to quickly guess whether the intended command is present or not. (2) Secondly, it inherently avoids too high a number of groups/tabs. (3) Finally, it avoids placing unrelated commands (*loners*) in groups with poor association.

The convergence of these two ideas – the exact set covering formulation with the IFT-based evaluative function – yields balanced and well-structured menus. Since the decision to include a command in a group and tab is modeled explicitly in the problem, no *post hoc* steps are needed to segment the outputs. The menus thus produced consist of a few tabs that, in turn, are made up of relatively large and well-organized groups. They also appear better for comprehension in terms of their structure than were results of previous work, also because the lead elements are semantically more indicative. It is easier to recognize the idea of a tab or group and act accordingly – that is, dismiss it or, if it is relevant for the goal at hand, zoom in. Also, the MIP implementation does not require extensive computational effort. The resulting formulation can deal with problem instances of 50 commands within a few hours of computational effort. Presently, however, it does not produce labels for the higher-level groups it has created, such as for tabs in a tabbed menu. At the end of the paper, we discuss strategies for producing labels for higher-level groups.

A designer can use the techniques (discussed in this paper) to explore the design space or to fine-tune an existing design (Oulasvirta et al., 2020). To use the optimizer, a design task (instance) must be defined. The inputs are (1) a list of command frequencies and (2) a matrix of pair-wise association scores (0–100). These can be given by the designer or obtained in a data-driven fashion. Access frequencies can be learned for example from click data, or estimated using click models. Word embeddings can be used to estimate pairwise association scores. Alternatively, when available, word association norms (e.g., based on WordNet) could be used. Also, co-occurrence of commands can be learned from existing menu designs as done in previous work (Bailly et al., 2013). Association scores are relatively easy to provide manually too. The association matrix is typically quite sparse because only a few cliques of commands have meaningful relationships and the rest can be skipped. So, filling-in the matrix does not require too much effort even for larger problem sizes. In our evaluation cases, a student could define an association matrix for about 50 commands within an hour.

To critically assess the approach, we report on a controlled comparison between optimized and commercial designs (Adobe Reader, Microsoft Notepad, and Mozilla Firefox). The results of our new approach demonstrates that users could work 25% faster with our optimized menus compared to the existing designs. The new approach is able to produce high-quality designs. We selected Adobe Reader, Microsoft Notepad, and Mozilla Firefox because these three applications represent a cross section of well-established commercial software

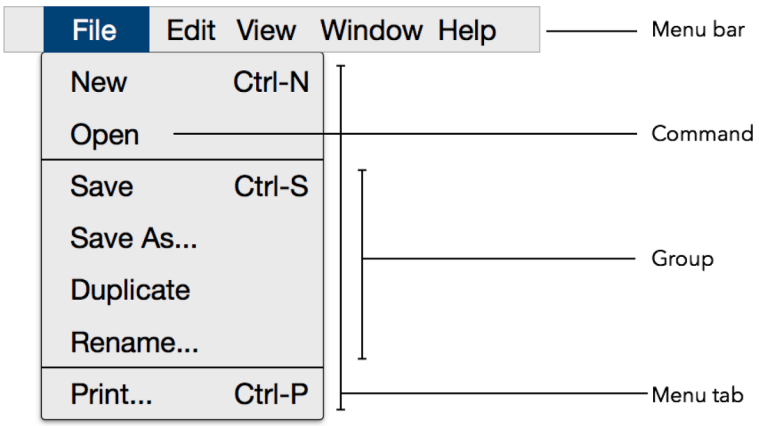



**Fig. 1.** Illustration of key terms utilized in this paper. The design task we examine is how to assign commands into groups and tabs accessible from a menu bar.

systems; the commands and context of these systems should be comprehensible to an average user without separate training or explanations. We point out the fact that this comparison baseline does not inherently include multi-level sub-menus; this matches our approach because we also do not address multi-level menus in the current paper. Further, the remainder of this paper intentionally does not discuss the naming of the tabs. Assignment of names involves natural language processing problems, which we do not address in the current paper.

The rest the paper is organized as follows: We first present a succinct review of the literature related to algorithmic menu design and related areas in operations research. After this, we define the design problem rigorously and use the definition to inform a "classical" MIP formulation that replicates the objective function used in previous work using Fitts' law and associativity as objectives. We then extend this formulation to utilize the IFT-based approach within the MIP formulation, after which we discuss applications also in personalization and adaptation of menus. We finally present a user study comparing our algorithms' results with commercial baseline designs. Finally, data from the controlled user evaluation is presented.

## 2. Problem definition

This paper addresses the problem of finding an optimal layout of commands in a hierarchical menu structure. The instance of a hierarchical menu examined here is the popular tabbed menu system in which commands are organized into groups, which are arranged into tabs. The most common menu types (linear menus, ribbon menus, etc.) are special cases of this general formulation that can be modeled by changing costs in the evaluative function. Our underlying objective with this paper – as reflected in the evaluative functions we explore – is to minimize the overall time consumed by the user in selecting commands.

The key terms utilized in our definition are illustrated in Fig. 1. We call a menu item a *Command*. A clearly demarcated set of commands (separated from other groups by a physical boundary) is termed a *Group*. Multiple groups arranged in vertically aligned structures constitute a *Menu Tab* (or tab). Individual tabs have their specific identifier text or titles; all of these titles collectively constitute the horizontally aligned structure denoted as a *menu bar*. For clear disambiguation, we note that this paper addresses single-level hierarchy within menus; this means that one *Menu Tab* in the menu bar will involve one vertically aligned structure of commands that must all be visible or hidden as an indivisible set. This paper does not support any further internal roll-over within the *Menu Tab*. While multiple groups are permitted within any *Menu Tab*, they must all be visible or hidden together.

Given $n$ commands, the problem first requires these commands to be organized as ordered sets, each representing one group. Then, these groups are themselves organized into ordered sets that form individual tabs. Lastly, the tabs, in turn, are ordered in terms of the labels to be shown in the menu bar. This means that the overall layout is a problem of reorganizing the unordered set $N$ into an ordered set of all $n$ commands.

The $n$ commands are characterized by frequency of use $\mathbb{F}$, associations $\mathbb{A}$, and (optionally) location preferences $\mathbb{L}$. We will denote the set of tabs as $T$, with individual tabs identified by index $\tau$. All groups (irrespective of their tabs) are assumed to constitute an unordered set of groups $C$. Individual groups will be identified by indices $c, \bar{c}$. We note that any tab $\tau$ is itself an ordered set of some groups. Further, any group $c \in C$ is also an ordered set of some commands. Finally, we use indices $i, j$ to denote commands. Hence, the frequency of usage of command $i$ is $\mathbb{F}_i$. The association between commands $i$ and $j$ is denoted as $\mathbb{A}_{ij}$. So, the menu design problem requires computation of an ordered set $T$ of ordered sets of groups of ordered commands, such that the objective function (defined via values of $\mathbb{F}$, $\mathbb{A}$, and optionally $\mathbb{L}$) is optimized.

### 2.1. Objective function: costs to minimize

The problem definition discussed above did not explicitly provide the objective function. Traditionally, menu design has involved an objective function that is a weighted combination of two factors: (1) Time required to access individual commands (2) Mutual association of commands placed within any group (see, e.g., Bailly et al., 2017; Bailly et al., 2014; Chen et al., 2015; Cockburn et al., 2007; Danilenko and Goubko, 2013; Goubko and Danilenko, 2010; Norman, 1991; Oulasvirta and Karrenbauer, 2018). This objective function helps in two ways. First, the time required by any user – who knows the location of a required command – is exactly captured using Fitts' law. So, the efficacy of an expert user (well conversant with the concerned application) is explicitly captured by the objective term. This ensures that the resulting menu system is indeed fast for usage by a well-experienced user. Secondly, the mutual association term helps a novice user in searching for any command quickly. Consider that a user is looking for the Save-As command. While exploring the menu, the user had encountered the Save command in the first (leftmost) tab. The user remembers this location and also associates the required Save-As command with the known Save command. So, the user is highly likely to search for the Save-As command in vicinity of the known Save command. The second objective term – representing the mutual association of commands – ensures that logically interrelated commands are collocated in close vicinity to each other.

In this paper, we do provide a classical formulation using this traditional objective function in Section 4. But then, we also propose a novel objective function based on the Information foraging theory in Section 5. For either formulation, our concept of cost encapsulates the efficiency of a broad set of users: While addressing the speed-up for

expert users who know the application well, we also wish to assist the exploratory efforts of novice users who are often searching for required commands with only a vague idea of requisite command names.

### 2.2. Scope

There exists a wide variety of interaction techniques for menus and menu-like paradigms for command selection (Bailly et al., 2017). This paper specifically focuses on non-adaptive menus with hierarchical structures. The positions of commands are not assumed to change during interaction. We target designs wherein commands are organized by groups and tabs. The objective function does not consider all aspects of menu design: selection of labels or shortcuts, item length, etc. For the purposes of this paper, navigation occurs by selection of a tab and then a command within the tab. In this instance of hierarchical menu systems, only a limited subset of a menu tree can be visible at a time. While it is possible to extend the techniques in this paper to cover some other types of hierarchical menus, that discussion is beyond the scope of the paper.

## 3. Related work

Our work builds on results in four areas of related work: (i) modeling of search performance for menus, (ii) meta-heuristic techniques, (iii) integer program (IP) approaches based on the assignment problem, and (iv) facility layout and next release problems in operations research and software engineering.

### 3.1. Search performance and predictive models

Human factors in selection performance have been studied extensively. There is evidence of the following empirical effects:

1. Shorter menus are faster to use (Byrne et al., 1999; Nilsen, 1996);
2. Targets that are closer to the top are faster to select (Byrne et al., 1999; Cockburn et al., 2007);
3. Linear menus with grouping ("semantic" or "systematic" menus) are much faster than random or alphabetical ones (Danilenko and Goubko, 2013; Hornof and Kieras, 1997; McDonald et al., 1983; Mehlenbacher et al., 1989);
4. It is faster to select a target that is present in the menu compared to determining that the desired target is not present in the menu (Bailly et al., 2014);
5. Users get faster with practice, and this positive effect influences other (non-target) items in the sub-menu also (Cockburn et al., 2007); and
6. Users often fixate on one of the first three items (Byrne, 2001).

Prior mathematical models (Bailly et al., 2014; Cockburn and Gutwin, 2009), typically using non-linear regression, capture some of these effects. Cognitive simulations such as EPIC (Hornof and Kieras, 1997), ACT-R (Byrne, 2001), and computational rationality (Chen et al., 2015) capture more effects but are very computationally expensive.

It has been found that users utilize various *search strategies* in menu navigation:

1. *Directed search* uses memory of element locations learned through practice for guessing where to search (Bailly et al., 2014);
2. *Serial search* progresses downward from the first or topmost command, after which items are sequentially examined one at a time (Byrne et al., 1999);
3. *Random search* selects an arbitrary position within the menu for fixation (Byrne et al., 1999; Hornof and Kieras, 1997); and
4. *Visually guided search* is based on sampling of visual landmarks such as the end of the menu or labels sharing visual features with the target (Bailly et al., 2014; Chen et al., 2015).

No previous mathematical model has precisely predicted the search behavior in a hierarchical menu. Sears and Shneiderman (1994) presented a model for *split menus*, where frequently used items are placed at the top of a linear menu, in their own group. Their model assumes that search time for frequently selected items is a logarithmic function of distance from the top and for low-frequency items is a linear one. Lee and MacGregor (1985) proposed that selection time follows the number of pages needed to access retrieval of a given item, number of items per page, time needed to assess one item, keystroking time, and system response time. These models do not account for the effects of any other design factor than the number of items on a page or in a group. Bailly and colleagues Bailly et al. (2014) introduced a free parameter to their model, indicating which of two search strategies is used. A gaze pattern was found wherein experienced users fixate on the first items in sub-groups and then either drill down or not. However, the pattern was not captured well by the model, and the authors identified this as a target for further improvement. A recently presented model (Chen et al., 2015) suggests that the optimal search strategy adapts to the semantic organization of a linear menu, allowing users to gaze more directly at relevant sub-groups. This model relies on reinforcement learning, and it has been tested for predicting the effect of menu organisation and menu length. on task completion time and eye movements.

With this paper, we investigate IFT's suitability for a computationally efficient model capturing a key aspect of hierarchy-related decision-making: the decision to keep reading a sub-menu (tab or group) versus jump to the next candidate.

### 3.2. Menu optimization using meta-heuristic techniques

Most prior work Troiano and Birtolo (2014); Troiano et al. (2008) on menu optimization has used a meta-heuristic technique. Meta-heuristic techniques do not make explicit assumptions about the objective function; rather, they consider it as an oracle that tells them the objective value of a given candidate. As a result, they can work with any objective function, including non-linear functions and even simulator models. In contrast to exact methods such as integer programming, meta-heuristic techniques cannot offer guarantees that the best design found is optimal. Moreover, meta-heuristic techniques, such as simulated annealing and genetic algorithms, include many hyperparameters, the tuning of these hyperparameters may affect their performance.

Troiano et al. (2016) used an index of accessibility and a user preference indicator to define objectives in a genetic algorithm (GA) solver that operates from the number of children of the item and depth of the menu hierarchy. Matsui and Yamada (2008) explored objective functions to address selection time that consist of search and pointing, a penalty term for functional dissimilarity with other designs, and a menu granularity regularizer. As the authors noted, the approach is brittle, because results can change dramatically with small adjustments to objective weights. The state-of-the-art approach at the moment is MenuOptimizer (Bailly et al., 2013). It uses a weighted sum from a selection time model (SDP Cockburn and Gutwin, 2009) and, as in Troiano et al.'s work, a structural metric. A weakness due to the use of the latter is that it produces results that are driven to be similar with previous designs. Further, the quality of the resulting solution changes with every execution. However, reasonable results can be produced quickly, within 5–15 min even on commodity hardware. The design of hierarchical menu systems has been restricted to the realm of meta-heuristic techniques, in areas such as simulated annealing (Matsui and Yamada, 2008), genetic algorithms (Golovine et al., 2010; Troiano et al., 2016), and ant colony optimization (Bailly et al., 2013).

Heuristic constructive approaches to optimization generate candidate designs using some heuristic that has been found to work in the domain. In menu optimization, this approach has been explored as a combination of exhaustive search and "drill-down" (Danilenko and Goubko, 2013; Goubko and Danilenko, 2012; 2010). That is, it initially explores the most relevant solution directions, applying a breadth-first paradigm, then chooses a few suitable candidates for in-depth

inspection. The methods presented thus far assume, however, that potential groupings of commands are stated *a priori*. This is a limitation, because the grouping predetermines the optimal solution. In this, the task becomes harder for the designer and relatively easier for the optimizer, which need only solve the ordering problem. Often, the grouping is a defining part of the problem at hand.

The method described in this paper requires only associations among frequent elements, without any pre-grouping of elements. However, if the *a priori* grouping were to be already available, the methods in this paper can make efficient use of this knowledge: Association values can be expressed using a binary mapping (value 1 when concerned commands are in same group and 0 otherwise).

### 3.3. Exact methods: menu optimization as an assignment problem

Unlike meta-heuristic techniques, exact methods are guaranteed to find the optimal solution in finite time. However, the time required may be an exponential function of the problem size (most interesting problems are NP-hard). The simplest exact method is *explicit enumeration*, wherein the objective value of each element in the solution space is evaluated and the current best solution (the so-called *incumbent*) is updated. In contrast, *implicit enumeration* makes use of relaxations that can be solved efficiently. One very popular form of implicit enumeration, which we use here, is *Branch & Bound*. This is one of the standard methods for solving Integer (Linear) Programs.

*Keyboard layout design* was defined in the 1970s as a quadratic assignment problem wherein the goal is to minimize the average time for movement between letters assigned to buttons. Relaxations to the quadratic assignment problem (QAP) formulation have reduced solution times with integer programming (IP) solvers to a permissible range even with realistic data (Karrenbauer and Oulasvirta, 2014).

The simplest application of this approach to menu design arrives at solutions for a *linear assignment task* (Oulasvirta and Karrenbauer, 2018); the formulation can be extended to full menu systems using hierarchical structures. Yet, designs created using assignment alone display systemic shortcomings. In particular, items are not assigned to groups, which precludes hierarchy. In addition, the most important items in each group are placed in the center (because of the associativity term). Moreover, though the design task has been formulated in integer programming terms, the problem has to be finally solved by means of meta-heuristic techniques due to the expensive nature of the evaluative functions required.

### 3.4. Selection problems in diet planning and service design

*Menu planning* (or diet planning) for the restaurant industry was defined as a linear programming problem in 1964 Balintfy (1964) and has received sustained interest since (Lancaster, 1992). The task here is to select food combinations that meet dietary, gastronomic, and production objectives. The *diet problem* is a variant of menu planning for an individual or group (Dantzig, 1990).

While food-menu planning is not relevant for the design of menus in computing systems, the selection of functionality is. Functionality selected for an application or service must be accessed via a menu system. What is known in software engineering as *the next release problem* refers to selection of new software features (Bagnall et al., 2001) that accounts for user preferences, dependencies among functions, and costs. In a recent paper (Oulasvirta et al., 2017), integer linear programming (ILP) was presented for the selection of functionality in interaction and service design. Here, we do not discuss the problem of how to select the menu commands; we assume the set $N$ to be known.

### 3.5. Facility layout problem

Facility layout problem (FLP) is a class of combinatorial decision-support system that deals with the location (placement) of facilities for factories (Drira et al., 2007). Facility location translates to the search for an optimal arrangement of non-overlapping indivisible entities within a given structure. The objective measure to be minimized is defined in terms of the weighted distance between the centroids of the entities being positioned. Within the broad FLP field, closest to menu design is the multi-row facility layout problems, or (Anjos and Vieira, 2017), a variant that allows for a layout with multiple rows (in a predetermined number) to which the entities can be assigned. The entities all have the same size (with height equal to the common row height), distances between adjacent rows are equal, and entities can be assigned to any row in general. We note a parallel between FLPs and our problem. The arrangement of commands into unique rows within tabs matches the multi-row facility layout problem (MRFLP). The literature discusses mixed-integer programming and also semi-definite optimization approaches for MRFLP (see Gen and Cheng, 2007). Regrettably, both approaches involve a highly non-linear objective function, which adversely affects computational performance. We cannot expect these prior formulations to address problems with sizes beyond 15–20 menu commands and with additional complicating design considerations. Moreover, their objectives are different and not directly to menu use, which emphasizes comprehensibility and fast access.

## 4. Reformulating the design task: a minimal representative formulation

The first part of our method is a flexible integer programming formulation for key decisions in designing a hierarchical menu structure. Intuitively the problem is defined as ordered sets at multiple levels. We pursue a formulation that is compact enough to produce results with limited computational effort. It is flexible also in the sense that it can support evaluative functions of numerous types. We designate the formulation as a *minimal representative formulation* (MRF).

The primary intention behind the MRF is to represent the structure of the *hierarchical* menu design problem, expanding from the assignment problem as dealt with by previous work. This lets us plug-in various evaluative functions and thereby benchmark existing approaches to menu optimization. The formulation relies on principles utilized in the relevant technical literature but not previously explored for menu optimization. These principles have made it possible to overcome some limitations of the assignment-based approach in the hierarchical case. In particular, we can now express an evaluative function that refers to a user's navigation behavior both at the level of commands and at the level of sets of commands. We exploit this to construct an information-foraging-based evaluative function that is more natural for hierarchical menu systems than the Fitts' law and associativity matrix approach (see the next section). The MRF is developed here as a mixed-integer linear program. We try out two evaluative functions to compare their results. The final outcome from the MRF is expected to represent the best possible results that can be obtained via techniques and approaches from prior literature.

Technically, the MRF employs *decision variables* that map specific commands to individual locations (row numbers within tabs). To address the set-cover-related consideration of determining the intra-tab grouping of commands, we define decision variables to represent the number of groups and also the constitution of any individual group. Finally, to ensure the integrity of groups and row numbers (i.e., avoid "holes" between row numbers), we use a general precedence variable that avoids non-linear terms while ensuring the expected computational performance. This results in a compact model with fewer variables and constraints.

**Decision variables:**

$$X_i^c = \begin{cases} 1 & \dots \text{ if command } i \text{ is placed in group } c \\ 0 & \dots \text{ otherwise} \end{cases}$$
$$Y_i^\tau = \begin{cases} 1 & \dots \text{ if command } i \text{ is placed on tab } \tau \\ 0 & \dots \text{ otherwise} \end{cases}$$
$$Q^{c\tau} = \begin{cases} 1 & \dots \text{ if group } c \text{ is placed on tab } \tau \\ 0 & \dots \text{ otherwise} \end{cases}$$
$$Z_{ij} = \begin{cases} 1 & \dots \text{ if commands } i,j \text{ are placed in the same group} \\ 0 & \dots \text{ if commands } i,j \text{ are placed in different groups} \end{cases}$$
$$W_{ij} = \begin{cases} 1 & \dots \text{ if commands } i,j \text{ are placed on the same tab} \\ 0 & \dots \text{ if commands } i,j \text{ are placed on different tabs} \end{cases}$$
$$R_i^r = \begin{cases} 1 & \dots \text{ if command } i \text{ is placed on the } r\text{th row of some tab} \\ 0 & \dots \text{ otherwise} \end{cases}$$
$t_i$ = The time required to reach command $i$ as indicated by Fitts' law

$$S^{c\bar{c}} = \begin{cases} 1 & \dots \text{ if group } c \text{ immediately precedes group } \bar{c} \text{ on some tab} \\ 0 & \dots \text{ otherwise} \end{cases}$$
$$\mathbb{S}^c = \begin{cases} 1 & \dots \text{ if group } c \text{ is the first (topmost) group on some tab} \\ 0 & \dots \text{ otherwise} \end{cases}$$
$$\xi^c = \begin{cases} 1 & \dots \text{ if group } c \text{ is used (has a non-zero number of commands)} \\ 0 & \dots \text{ otherwise (empty, with no commands)} \end{cases}$$
$$\beta^\tau = \begin{cases} 1 & \dots \text{ if tab } \tau \text{ is used (has a non-zero number of groups)} \\ 0 & \dots \text{ otherwise (empty, with no groups or commands)} \end{cases}$$
$$\Theta^{c\bar{c}} = \begin{cases} 1 & \dots \text{ if group } c \text{ is placed anywhere before group } \bar{c} \text{ on some tab} \\ 0 & \dots \text{ otherwise} \end{cases}$$
$P^c$ = Starting position (row number) of group $c$ within its tab

Decision variables $X, Y, Q$ define the unordered structure of the groups and tabs. Decision variables $S$ and $\Theta$ offer alternative ways to enforce the relative ordering of groups. The absolute positioning of commands is provided by $R$. The variables $W, Z, \xi, \beta$ ensure that the decisions enforced by the other variables are in synchronization with each other. In the absence of $W, Z, \xi, \beta$, the results implied by the other variables can be infeasible or non-unique.

The decision variable $\Theta$ requires more explanation. Classical mixed-integer programming formulations handle sequencing of elements by using an immediate precedence variable; this variable typically specifies that some element $i$ *immediately* precedes some element $j$. The immediate precedence variable inherently dictates the relative ordering and also the collocation; there must not be any element $k$ between $i$ and $j$. In contrast, $\Theta$ specifies the relative order alone and not the collocation. Hence, one or more elements may be present between $i$ and $j$. This general precedence variable provides several logical distinctions from the immediate precedence approach, and it will be required for row numbering. The full set of constraints applied to these decision variables is covered in the (Appendix A.2).

### 4.1. Example application: an evaluative function based on previous work

A key benefit of the MRF is that it can be used with a broad range of evaluative functions. Here, we replicate the "two-fold-objective" function of Bailly et al. (2013), which balances (i) the time required to reach the commands (weighted by the frequency of use) against (ii) the association of commands placed in a specific group or on a certain tab.

$$\text{Maximize: } \sum_{i\in N}\sum_{j\in N}\mathbb{A}_{ij}\left(\lambda_c Z_{ij} + \lambda_m W_{ij}\right) - \lambda_f \sum_{i\in N}\mathbb{F}_i t_i \tag{1}$$

The parameters $\lambda_f$, $\lambda_c$, and $\lambda_m$ are the relative weightages for access time (from Fitts' law), intra-group associations, and inter-group (intra-tab) associations. We will discuss values for these weightages in Section 5.3. We designate this as the two-fold objective because we are capturing two different metrics here: the term in $\mathbb{A}$ implies the effort in guessing the location of a command; the term in $\mathbb{F}$ captures the time required to reach a particular command when the user already knows its location. The results obtained for this objective function are presented, in Section 6, for comparison of the results to those obtained from the evaluative function based on the information foraging theory.

## 5. The information foraging approach

We develop a new evaluative function based on the information foraging theory (IFT). IFT models search behavior as utility-maximization in a patch world. The theory is an application of the optimal foraging theory in biology, which describes the hunting and food search behavior of animals. An adaptive agent is assumed to change the patch as soon as the gain decreases to a level that it would make more sense to move elsewhere. Consider a wolf that has nearly exhausted the food in its current forest. Should it stay there, go to a nearby forest with rabbits (representing few calories per day), or travel a bit further to reach a different patch with deer (which are harder to catch but offer a greater gain)? Application to menu interaction follows analogous reasoning. Just as in food foraging, the (information) ecology of a menu is *patchy*; that is, information about the target is unevenly distributed to patches such as groups and tabs. Because some of the patches are not fully visible and are accessed only via higher nodes in the tree, the agent must decide what to attend under uncertainty. From what is locally visible (e.g., the first items in a group), the user must infer what the rest of the region may carry. Hence, the key to menu design is not how close an item is to the top, or to related items, but how economically the user can decide how well it represents the rest of the menu.

To model this kind of "information scent" Pirolli and Card (1999), as is done in IFT applications in general, we assume that an agent's environment consists of patches indicative of a target to varying degrees and connected with distances (or time costs). Each patch is associated with a gain: a function describing how quickly information is extracted when the user is in that patch. In IFT, a non-linear (logarithmic) function is used to model gain. It is continuous and has the property of diminishing returns: as more time is spent in a patch, less information becomes available, and a rational forager moves to another patch. The theory further posits that the user must make a decision for every set of commands attended. In the case of a menu, the user looks at the leading (top) commands in the set (e.g., under "File") and then makes a guess as to whether or not this set of commands may contain the desired command (e.g., "Zoom In"). If the user guesses that the current set should contain the required command, then the user will investigate further by reading (exploring) within this particular set. Otherwise, he or she discards the current set and moves on to the next *without really analyzing the content of the current set.*

To make IFT amenable to mixed-integer programming, we have formulated a sample–discard–explore paradigm that allows avoiding non-linearities (e.g., logarithmic gain functions) but retains the essence of this foraging behavior. Intuitively, the sample–discard–explore function captures four logical outcomes possible during search:

1. **True positive**: The user guesses that the set contains the target command, and it indeed contains that command. In this case, the cost during search within the set is the time consumed (by Fitts' law) to scan the list and move the pointer over the required command in the set.
2. **True negative**: The user guesses that the current set does *not* contain the required command, and the set indeed does not contain it. No further cost is incurred.
3. **False positive**: The user guesses that the set contains the required command, but it actually does not. The additional cost incurred for this set is the time consumed (under Fitts' law) to navigate all commands in the set. This cost is proportional to the size of the set.
4. **False negative**: The user guesses that the set does not contain the required command, but in reality it does. Now the user must (fruitlessly) analyze all succeeding sets, such as subsequent groups on the tab.

We assume that the user begins the search by sequentially analyzing (sampling) the lead elements of every set encountered. On the basis of the decision made to discard or explore any specific set, the user invests the corresponding effort for that set. This process repeats until a true positive (target) is reached. This logic can be applied recursively at any level of a hierarchy where multiple options (sets) are available. In our application we assume two levels: Tab and Group. The insight is that the total time expended in locating a specific command is the summation of time spent in the four possible scenarios, weighted by the probability of the user making the corresponding decision for the relevant set. To obtain an estimate for the entire menu structure generated by an optimizer, the estimated times are further weighted by the frequency of use of individual commands.

In addition to the search-related time components, motor selection efforts must be considered. Consider the case where the user already knows or remembers that some command $i$ is in group $c$ on tab $\tau$ at row number $r$. There is no search effort at all, yet the user still takes some time to traverse to row $r$ on tab $\tau$. As in previous work, this time is as computed from Fitts' law, and it depends on $r$ and $\tau$ only. We address this time via the decision variable $t_i$.

We also need to quantify the user's expectation of a specific group or tab featuring command $i$. This expectation depends solely on the user's current knowledge of the presence/absence of other commands (such as $j$) in the group or on the tab. We quantify this expectation as follows:

1. If the desired command ($i$) is the leading (topmost) element of any group, the expectation is 100% for that group.
2. If leading element $j$ of any group has a very high score (above 80%) for association with command $i$, then the expectation is 100% for that group. Conversely, a very low score (below 20%) for association between $i$ and $j$ leads to a negligible expectation.
3. For intermediate, unexceptional association score values, the expectation is scaled in proportion to the relative value of the association score with respect to the median one.

Given a group $c$ and an element $i$, we can judge the expectation of the presence of $i$ in $c$ by looking at lead element $j$ of group $c$. Hence, the expectation of $i$'s presence depends solely on the association between $i$ and $j$. We denote this expectation as $E^{ij}$. We note that $E^{ij}$ can be computed in advance through a pre-processing step, so it can be treated as a known parameter in the mixed-integer programming formulation. We use $E^{ij}$ to scale the efforts in every group for every command.

### 5.1. Mathematical formulation

To develop an integer programming formulation based on IFT, we require decision variables that can uniquely specify the solution (the resulting layout) while enabling computation of the various efforts mentioned above. The specific decision variables (including computation of specific efforts) are explained below. We note here that the decision variables described in the previous section are required too, along with their constraints.

$$U_c^j = \begin{cases} 1 & \dots \text{ if command } j \text{ is the topmost (lead) element of group } c \\ 0 & \dots \text{ otherwise} \end{cases}$$

$\Phi_i^c$ = Total time / cost for command $i$ computed for group $c$

$\alpha_i^c$ = True-positive time / cost for command $i$ computed for group $c$

$\sigma_i^c$ = False-positive time / cost for command $i$ computed for group $c$

$\delta_i^c$ = False-negative time / cost for command $i$ computed for group $c$

$\Omega_i^\tau$ = Penalty incurred for command $i$ if it is placed on (non-standard) tab $\tau$

The decision variable $U_c^j$ assists in locating the lead element in any group. Our IFT approach is based on this lead element, so its knowledge is critical. The variable $\Phi_i^c$ encapsulates the expected total time, cost or effort required to reach the command $i$ if it were to be placed in group $c$. The value of $\Phi_i^c$ will be a function involving the location of group $c$, its lead element and overall composition. This value of $\Phi_i^c$ is expressed in terms of the three possible navigations – true-positive, false-positive and false-negative as covered by $\alpha_i^c, \sigma_i^c, \delta_i^c$ respectively.

The objective is to minimize weighted cost $\Phi$ (weighting is by frequency of use) for the time taken to reach any command placed within any set.

$$\text{Minimize} \quad \sum_{i \in N} \mathbb{F}_i \left( \sum_{c \in C} \Phi_i^c + \sum_{\tau \in T} \Omega_i^\tau \right)$$

such that:

$$\Phi_i^c \ \geq \ \lambda_0 t_i \ + \ \lambda_1 \alpha_i^c + \lambda_2 \sigma_i^c + \lambda_3 \delta_i^c \ \dots \forall \ i \in N, c \in C \tag{2}$$

This constraint is the key to the IFT approach and requires more explanation. Here, $\Phi_i^c$ does not intend to capture the exact time spent by a specific user in finding a specific command during a specific single session of usage. Rather, it intends to encapsulate the weighted estimate of sum of searching effort and accessing effort for that command. The first term of this constraint represents the time (computed via Fitts' law) needed to navigate to command $i$ if its location is known in advance. But the wasted efforts from false positive or false negative – and even the searching effort from true positive – must also be counted while optimizing the location of $c$. The remaining terms indicate the relevant costs incurred in searching for command $i$ with respect to group $c$. Hence, $\Phi_i^c$ is the summation of all concerned costs. The $\lambda$ values are weight factors to be specified by the designer. We introduce constraints to ensure that exactly one command is marked as the lead element of every group.

$$X_i^c \geq U_i^c \dots \forall i \in N, \forall c \tag{3}$$

$$\sum_{i \in N} U_i^c = \xi^c \dots \forall c \tag{4}$$

In addition, we need to ensure that command $i$ marked as the lead element of any group $c$ has its row number equal to $P^c$. Next, we look at constraining the values of individual cost components. To calculate an individual cost component, we take the scalar product of the expectation value (probability of exploring the set) and the time expended in exploration of this set. For example, the expected expense of a false positive for command $i$ in group $c$ is as follows:

$$\sigma_i^c \geq E^{ij} \sum_{k \in N} X_k^c - \nabla \left(1 + X_i^c - U_j^c\right) \dots \forall i, j \in N, \forall c \tag{5}$$

Here, $\nabla$ is a suitably chosen sufficiently large constant. If the $\nabla$ related term is neglected, then the false-positive cost is computed in terms of the size of the group that was needlessly explored. The $\nabla$ related term voids the constraint if $j$ is not the lead element or if $i$ actually is present. Thus, the constraint addresses the case of a false positive occurring when group $c$ is led by element $j$ and the user is exploring $c$ to search for $i$ when $i$ is not, in fact, present in $c$. Next, let us consider the case of a false negative for command $i$ in group $c$:

$$\delta_i^c \geq \left(1 - E^{ij}\right)(|C|) - \nabla \left(2 - U_j^c - X_i^c\right) \dots \forall i, j \in N, \forall c \tag{6}$$

If the $\nabla$ related term is neglected, then the false-negative cost is computed in terms of the number of sets that will be needlessly explored. The $\nabla$ related term voids the constraint if $j$ is not the lead element or if element $i$ is not really present. Thus, the constraint addresses the false-negative case when group $c$ is led by element $j$ and the user discards $c$ to search for $i$ because of a low value for $E^{ij}$. This means a high value for $(1 - E^{ij})$. Next, we examine handling of the penalty related to the tab locations where certain commands are normally expected.

$$\Omega_i^\tau \geq \lambda_4 \left(1 - Y_i^{\tau *}\right) \cdots \forall i \in N \tag{7}$$

Here, $\tau*$ refers to the preferred tab as specified by $L_i$. If the command is not on its preferred tab, a penalty of $\lambda_4$ is incurred.

### 5.2. Handling of loners

"Loner" commands have poor association with other commands but may have a high frequency of usage. To avoid disturbing the cohesion of other (well-associated) groups, we strive to put all such loners in a separate group of their own. However, the loner group itself becomes contentious when the association for a specific command is not at either extreme – that is, when the command is not associated strongly with other commands but does not actually have an average association low enough to denote it as a loner. Putting such commands in the loner group makes this loner group too large.

We introduce a new hypothetical (invisible) command $\kappa$ to the set $N$. This command will not be placed in the actual menu; it is only introduced temporarily to serve as a focal point for association of all loner commands. The new command has the lowest non-zero usage frequency and no location preference, but it will still be constrained to be the lead element for its group, specifying $\sum_c U_\kappa^c = 1$. The association of this command with all other commands is computed as follows:

1. For any command $i \in N$, compute the sum of the association scores for $i$ with all other commands $j \in N$, and designate this sum as $\Sigma_i$
2. Find the largest value of $\Sigma_i$ among all $i \in N$, and designate this maximum as $\nabla$
3. For any command $i \in N$, compute the value $\nabla - \Sigma_i$, to be designated as loner factor $\mathbb{W}_i$
4. The association of command $i$ with $\kappa$ is calculated as $\mathbb{W}_i$ / the square root of $N$
5. If any command $i$ has greater than average association with any one command in $N$, the association of command $i$ with $\kappa$ is marked as zero

After this, the optimization problem is solved for the augmented set $N + \{\kappa\}$. Naturally, the association score for strongly loner elements with $\kappa$ is quite high. This ensures that $\kappa$ serves as a "loner magnet," attracting all loner elements to a single group. Further, the weight used to compute the association score from the loner factor in step 4 can be modified to control the size of the loner group. No other constraints are required for the loners. The loner factor automatically handles the balance of populating the relevant group with commands while avoiding the problems of very large or very small groups.

### 5.3. Parameter values

The results from IFT (and even from the minimal representative formulation) are sensitive to the values set for $\lambda$, as direct weights in the objective function, any gross modifications to these values result in different menu layouts. For the results reported on in this paper, we have aimed for rough equivalence in settings among the objectives: no single term dominates in the overall objective function. To achieve this equivalence, we still need some rough information about the expected resultant layout of menus. So, we used the following as initial indicative intentions regarding layouts:

1. We prefer the menu layouts such that the number of commands (rows) in every tab is not varying too much. As an example, we note that the baseline or commercial menu structure in Notepad application has a very strong variation. While the *Edit* tab has 11 commands, the *View* tab has just one command. We would not prefer such a huge variation in our menus.
2. The maximum number of permitted tabs scales up as a logarithmic function of the number of commands to be placed.
3. We prefer not to use single-element sets. If some command is so unrelated to all other commands, it is a suitable candidate for being in the loner set. A single element group – as used in the *View* tab of the baseline version of Notepad application – will create several problems. Primarily, it hampers the access time of all subsequent commands. Secondly it also increases the total cost of all false negatives for previous commands.

The intentions listed above are not enforced as hard constraints and are not included in the objective functions. Rather, they are used as indicative guidelines while designing the values of parameters $\lambda$.

Consider the values of $\lambda_f$ and $\lambda_c$ as presented above in Eq. (1) for the minimal representative formulation. For a data instance involving $n$ elements, we first expect that roughly $\log(n)$ tabs are allowed and every tab has $\frac{n}{\log(n)}$ commands on average. Assuming that the overall User-Interface is of width $w$ and height $h$, the term $\sum_{i \in N} \mathbb{F}_i t_i$ in Eq. (1) will have a value of the order of magnitude of roughly $\frac{wnh}{2\log(n)}$. The term $\sum_{i \in N} \sum_{j \in N} \mathbb{A}_{ij} Z_{ij}$ will be in the general vicinity of $n^2/2$. For a sufficiently large canvas and for lower value of $n$, value of $\frac{wnh}{2log(n)}$ is substantially larger than $n^2/2$. To ensure that the two terms have comparable impact in the overall objective function, the value of $\lambda_c$ should be around $\frac{2wh}{nlog(n)}$ times that of $\lambda_f$. A similar approach is used to initially set the lambda values for IFT.

However, we expect that the designer will fine-tune the results further. It is expected that the final relative values will be ascertained through trial-and-error over a large number of results.

## 6. Results

In this section, we assess the quality of the optimized designs and report on performance quantitatively.

### 6.1. Task instances

The approach was tested with three realistic design scenarios, selected for their representativeness, size, and diversity as cases. Our goal was to use a wide range of often-used applications, where the names and meanings of most commands are clear to typical users. Users who are already conversant with the commands will also have some ideas and expectations as to mutual associations, preferred placements, etc. With these objectives, we chose the following applications as test cases:

1. The classic *Windows Notepad*™ application is a widely used compact, elementary text editor. The existing design involves 23 commands, distributed across five tabs.
2. The *Adobe Acrobat Reader*™ application is a commonly used reader for files in PDFformat. Version 11 of this application has a menu system with 46 commands, distributed over five tabs.
3. *Mozilla Firefox*™ 3.6 is a well-known browser application. Its menu system has 51 commands, spread across seven tabs.

These instances are represented by means of the following parameters: (i) names (text strings) and relative frequency values $\mathbb{F}$ for all commands, (ii) association score $\mathbb{A}$ for every pair of commands, and (iii) (optional) location preference $\mathbb{L}$ for any of the commands.

To ensure the practicality of the instances, the authors implemented a short exercise in which two external designers (students enrolled in an human-computer interaction program) were asked to rate two parameters independently: (i) how many times in the course of a typical usage session is a specific command required and (ii) how closely are the two given commands related to each other. The answers were quantified to yield the frequency and the association score, respectively. We found the disagreement to be below one percent, so the values from the two raters were accepted as the data instance specification.

In the discussion below, the incumbent menu is the menu that exists in the commercial applications as of the time of writing. This is referred

**Table 1**
Computation times for optimal solutions from the optimizer, for two distinct evaluative functions.

| Application | Number of elements | Average time to find the optimal solution, in minutes | |
|---|---|---|---|
| | | Two-fold-objective approach | Information foraging approach |
| Windows Notepad | 23 | 17 | 22 |
| Adode Acrobat Reader | 46 | 960 | 1201 |
| Mozilla Firefox | 51 | 1145 | 1633 |

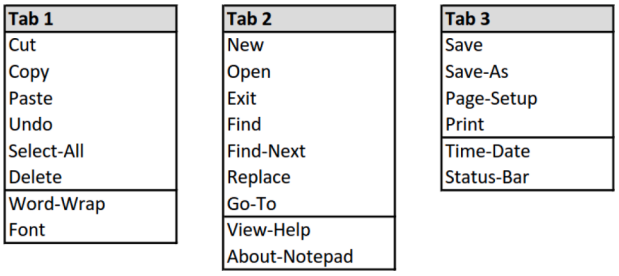



**Fig. 2.** A menu optimized for the Windows Notepad™ text editor with the two-fold-objective minimal representative formulation approach.

to as the "baseline design." The baseline designs for all data instances are provided in Appendix A.3. The layouts produced via the optimization method proposed in the paper are denoted as the "optimized design.

### 6.2. Implementation and numerical performance

The mixed-integer programming (MIP) formulations were coded in the Java™ SE (build 1.8) plarform. These formulations were solved using IBM™ Cplex™ 12.6.2 solver on an eight-core 64-bit Intel™ i7™ processor running at 2.8 GHz with 16 GB of RAM. The MIP solver was used with Concert™ technology in conjunction with several customized callbacks.

Computation times for the three cases are given in Table 1. These times are reported as averages over multiple computational executions, with different values for weight functions $\lambda$ and $\Phi$ (for values as computed in Section 5.3).

We note that for extreme values of $\lambda$ and $\Phi$, much shorter computation times were observed. For example, if $\lambda_f >> \lambda_c$ and $\lambda_f >> \lambda_m$ are set, then the minimal representative formulation objective does not really consider the association between commands at all. For such an extreme setting, the corresponding problem can be solved within a small fraction of the computational time listed. So, the computational performance strongly depends on the chosen parameters.

The listed performance is for the default versions of the MIP formulations (where the raw model is passed to the solver without making any attempts to improve performance). In practice, there exist several MIP techniques to improve computational performance. Further, the performance improves substantially when we start with a known (existing) solution instead of starting from scratch. In our case, one feasible solution (the existing layout) is always known, so this technique can be easily utilized. However, the discussion of such MIP techniques is beyond the scope of the current paper; our focus is to demonstrate the efficacy of the menu design approach and not the numerical method.

Although the computational effort excludes the use of these solvers in interactive design tools, they appear satisfactory with regard to the problem considered here, especially for one-shot optimization. Developers of professional software should be ready to invest a few hours or even two weeks of computational effort to ensure the quality of their menu design. This can be further speeded up by using dedicated hardware: our computations were done using a commodity laptop. Secondly, the data shows that using IFT does not significantly impair performance relative to that with the earlier, two-fold-objective approach.

### 6.3. The two-fold-objective approach

We now present designs generated via the algorithms discussed earlier for the data instances specified in Section 6.1. The results for the three data instances with the two-fold-objective approach are provided in Figs. 2–4.

It should be noted that the optimized menu for Notepad has fewer tabs than the baseline (commercial) design. This is because the Notepad data instance involved high scores for association between commands. The association values for pairs of commands for Acrobat and Firefox were comparatively low. The sparse association led to a wider layout with more tabs and smaller groups. The argument extends to the Acrobat menu also: two large groups can be seen in the Acrobat design, in the first and the second tab. The constituent commands of these groups have the strongest mutual associations. In addition, the association scores of these commands represent the only case of associations being relatively large for that data instance. Hence, the two-fold-objective formulation is strongly driven by the relative association values. If we had set the $\lambda$ values in Section 4.1 differently, that formulation would not have valued the associations scores so highly. The two-fold-objective formulation is quite sensitive to the $\lambda$ values. Further trials with the optimizer showed that multiple, quite different feasible solutions could be found within a narrow range of objective values. However, the specific relative weightage of the two primary terms in the objective function led to a situation in which an odd/unexpected placement was linked with a slightly higher objective value. We conclude that the two-fold-objective nature of the minimal representative formulation approach (with weighted performance from Fitts' law and mutual association of commands) led to a more opaque objective function.

Subjectively, it is difficult to justify the placement of a specific command in the location suggested by the optimal solution. Some observations can be made nonetheless. For example, the lead elements of most groups in the Adobe Acrobat menu are quite esoteric and unrepresentative. We note also that the two-fold-objective approach did not,

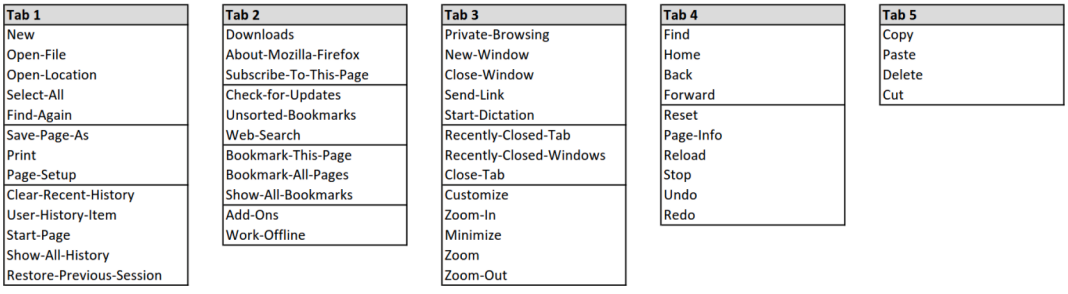



**Fig. 3.** A menu optimized for the Mozilla Firefox™ browser with the two-fold-objective minimal representative formulation approach.

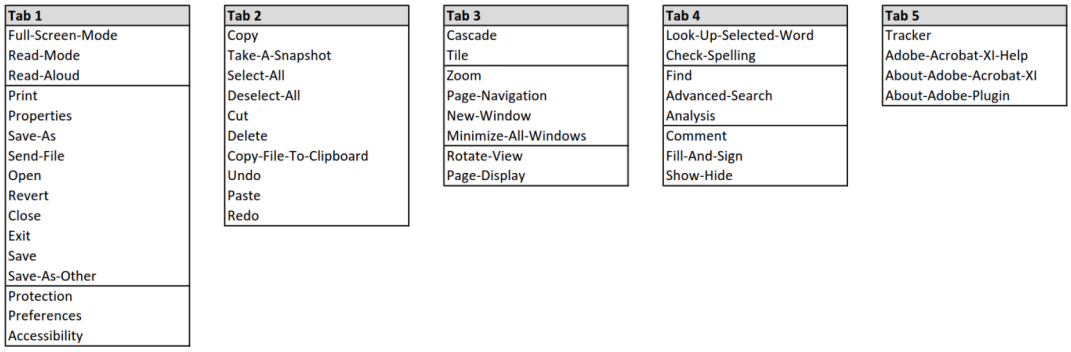



**Fig. 4.** A menu optimized for the Adobe Acrobat™ PDF viewer with the two-fold-objective minimal representative formulation approach.

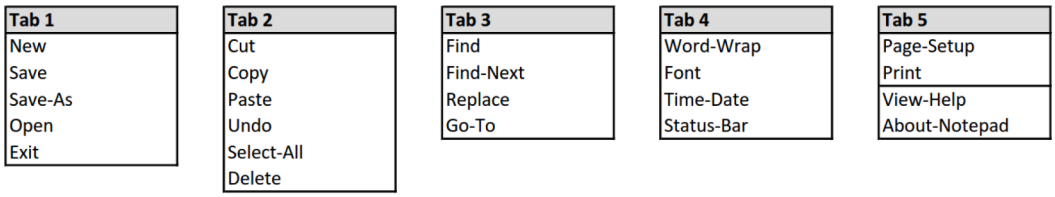



**Fig. 5.** The IFT-based approach for a menu optimized for the Windows Notepad™ text editor.

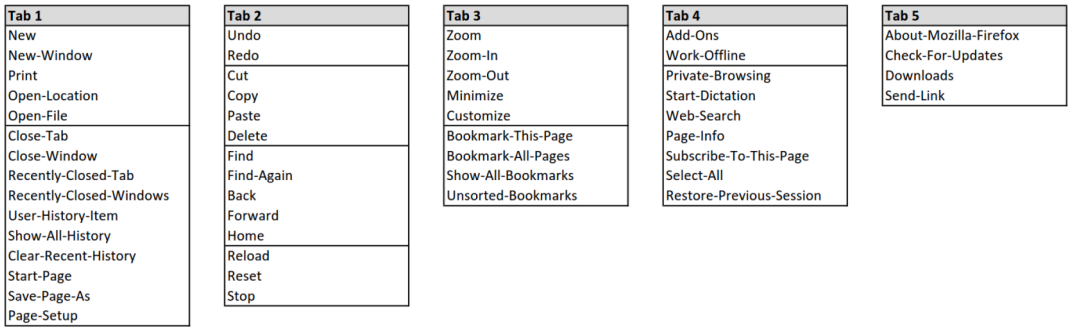



**Fig. 6.** A menu optimized for the Mozilla Firefox™ browser via the IFT-based approach.

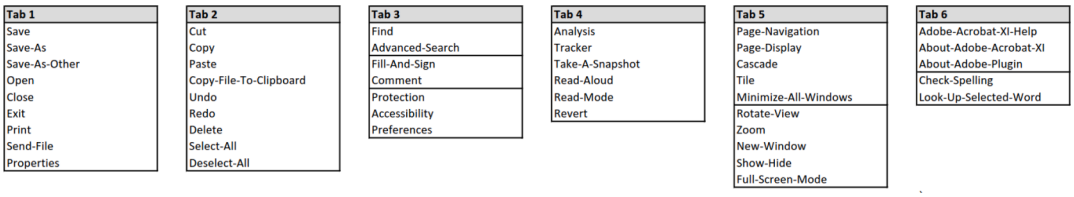



**Fig. 7.** A menu optimized for the Adobe Acrobat™ PDF viewer by means of the IFT-based approach.

without further information, address preferential placement of commands on desirable tabs. For example, the last/rightmost tab in the Firefox menu contains the *Cut, Copy*, and *Paste* commands and not the *Help* and *About* commands commonly expected here.

### 6.4. The IFT-based approach to optimized menu designs

Next, we consider the results obtained with the information foraging approach. The layouts generated are depicted below, in Figs. 5–7.

A few subjective observations can be made about the results. Firstly, the results for Notepad and also for Adobe Acrobat show a larger number of tabs. However, the tabs appear more balanced than with the previous method. In particular, there is less deviation among the tabs in the number of commands. An exception is the first tab in the Firefox menu, which is large relative to the others, but the remainder of the menu is well-balanced. Also, the groups appear reasonable in light of usage frequency (e.g., *Undo, Cut*, and *Copy*), associations between commands (e.g., *Undo* and *Redo*), and expectations for command position (e.g., *About*). In addition, groups' first items (lead items) are generally more indicative of the rest of the group here than with the two-fold approach (e.g., "Full-screen-Mode" in Fig. 4). Finally, there is good control of loners and also of preferential placement of elements on tabs, thanks to these being explicitly addressed in the formulation. With the next section, we explore whether these observations correlate with empirical results for user performance.

We should note that the IFT-based approach also turned out to be less sensitive than the two-fold one. There are relatively few *different* feasible solutions in close proximity to the optimal one. The IFT approach is

**Table 2**
Statistical results for Notepad, Mozilla Firefox, and Adobe Reader.

| Application | Dependent variable | Type | Statistical values | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | Average | SD | Median | U | *p*-Value | Effect size |
| Notepad | Selection time | Optimized | 1.99 | 1.12 | 1.63 | 88,984 | $< 0.001$ | 0.16 |
| | | Baseline | 2.38 | 1.58 | 1.93 | | | |
| | Number of tabs selected | Optimized | 1.43 | 1.15 | 1 | 59,448 | $< 0.001$ | 0.40 |
| | | Baseline | 1.64 | 1.29 | 1 | | | |
| Mozilla Firefox | Selection time | Optimized | 3.01 | 2.64 | 2.11 | 360192.5 | $< 0.001$ | 0.17 |
| | | Baseline | 3.36 | 2.74 | 2.52 | | | |
| | Number of tabs selected | Optimized | 1.84 | 2.18 | 1 | 417387.5 | $< 0.001$ | 0.08 |
| | | Baseline | 2.07 | 2.25 | 1 | | | |
| Adobe Reader | Selection time | Optimized | 3.37 | 2.63 | 2.42 | 341381.5 | $< 0.001$ | 0.20 |
| | | Baseline | 4.59 | 4.14 | 3.14 | | | |
| | Number of tabs selected | Optimized | 2.22 | 2.10 | 1 | 421775.5 | 0.080 | 0.04 |
| | | Baseline | 2.68 | 2.72 | 1 | | | |

affected less by minor variations in the relative weights of the terms in the objective function. This should be advantageous in cases wherein the weights cannot be deduced *a priori*.

## 7. Empirical evaluation

A controlled laboratory study was carried out in line with established practices in research on menu interaction (Bailly et al., 2014). We compare average selection times between optimized and non-optimized designs when everything else is kept equal. In the conditions used, the name of a command (the *target*) is shown on the display and the user is to find and select it as swiftly as possible. Three optimized designs (the task instances described above) were compared with commercially deployed designs (baselines). A non-uniform Zipfian distribution of command selection frequency was used, as in earlier research (Liu et al., 2017). We applied the same distributions used in the task instances, which were obtained via data from the external designers (see above). To avoid interference effects, each user experienced either the optimized or the non-optimized version of a menu, not both.

### 7.1. Method

*Participants* Twenty-four participants were recruited by means of email advertisements and personal networks. Their average age was 29.71 (SD: 3.14). Eight of the participants were female. One subject was left-handed. All were non-native English-speakers and familiar with computers' mouse and menu systems. At the end of the experimental study, we asked each participant whether he or she had seen these menus before or not, whereupon about 80% claimed to be familiar with these menus but not to have any idea of whether the locations of the commands had been changed. All were compensated with a movie ticket.

*Experiment design* Three applications were studied, as explained in the description of *task instances* above: Notepad, Adobe Reader, and Mozilla Firefox. Each user used an application either in the optimized or in the baseline condition (again, not in both). This yielded eight unique combinations (baseline or optimized for Notepad multiplied by baseline or optimized for Adobe Reader multiplied by baseline or optimized for Mozilla Firefox: $2 * 2 * 2 = 8$). Participants were assigned to conditions by rotation.

*Task and procedure* The study started with a brief introduction to the purpose of the study and the tasks to be performed. Demographic data was collected with regard to gender, age, native language, and level of familiarity with menus, via a questionnaire. After this came the main part of the experiment: The label of every target command (to be searched for) was displayed at the outset, after which the menu was shown once the participant had pressed a *Start* button. The task now was to find and click the target command as quickly as possible. Selection time was measured as the duration between pressing the *Start* button and clicking the command within the menu.

For Notepad, Mozilla Firefox, and Adobe Reader, this sequence of steps was completed 40, 80, and 80 times, respectively. The participants explored one layout at a time before proceeding to the next application. They had to find several commands within the given candidate layout, and then the next candidate layout was used. The complete procedure took approximately 30 min per user.

*Materials* For the 40, 80, and 80 commands (again, presented for Notepad, Mozilla Firefox, and Adobe Reader, respectively), the baseline designs were obtained from the latest Microsoft Windows version of the application at the time of the experiment (in January 2018). We used the Roulette wheel method (Goldberg and Deb, 1991) to sample from the frequency distribution of commands in the menu (see the description of the task instances). Because the optimizer does not choose tab labels, we used the first command on each tab as the label for that tab. For fair comparison, this was done in both conditions.

### 7.2. The apparatus and setup

The experimental software was implemented in Python with the Tkinter module for the menu system. Tkinter was used for presenting the menus and for recording selection times, mouse trajectories, and background data. The experiment was carried out on a computer running Windows 7, with 8 GB of RAM and a 20-inch LCD display. A mouse was used as the pointing device. The transfer function and other settings of the input device were specified by the experimenter and kept constant across all participants.

### 7.3. Results

Twenty-seven out of the 960 trials with Notepad, 27 out of the 1920 with Mozilla Firefox, and 42 out of the 1920 with Acrobat Reader were removed from the final dataset, for two main reasons: (i) selection of the wrong command (slip) and (ii) taking excessively long to find the target. We found that selection times (STs) were not normally distributed, as is common in reaction and choice reaction studies, and we used Mann–Whitney U testing (Mann and Whitney, 1947) for the statistical tests (Table 2).

Average ST was 1.99 s (SD: 1.12) for the optimized Notepad and 2.38 s (SD: 1.58) for the baseline Notepad design, 3.01 s (SD: 2.64) for the optimized and 3.36 s (SD: 2.74) for the baseline Firefox design, and 3.37 s (SD: 2.63) for the optimized and 4.59 s (SD: 4.14) for the baseline Acrobat Reader design. Moreover, the *p*-value for all three applications was less than 0.05, showing that there is a statistically significant difference between the optimized and baseline STs. In other words, our method was able to decrease STs for these menus.

We also examined the number of tabs selected before finding of each command. The average was 1.43 tabs (SD: 1.15) for optimized Notepad and 1.64 (SD: 1.29) for baseline Notepad. The corresponding figures for

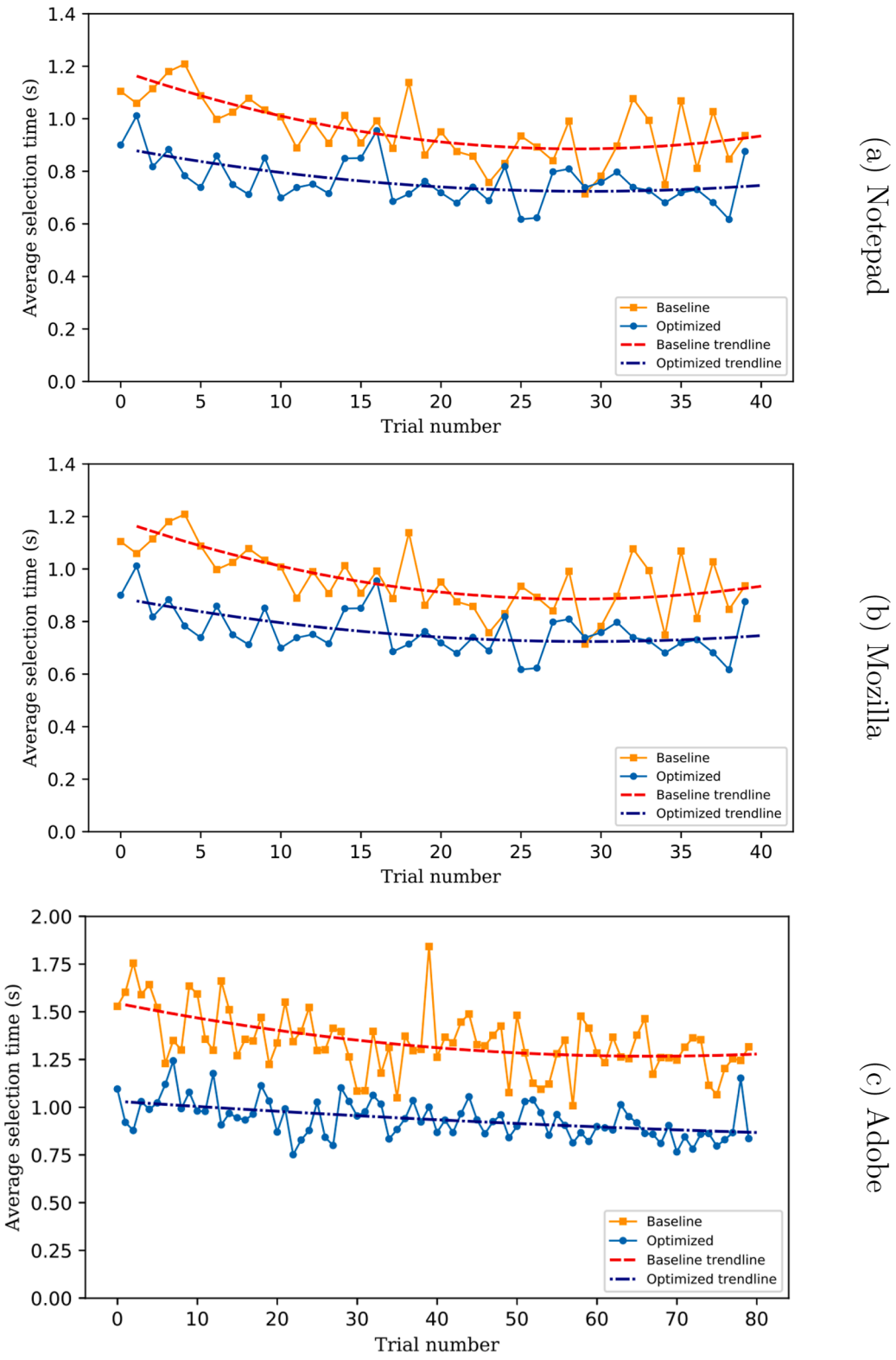
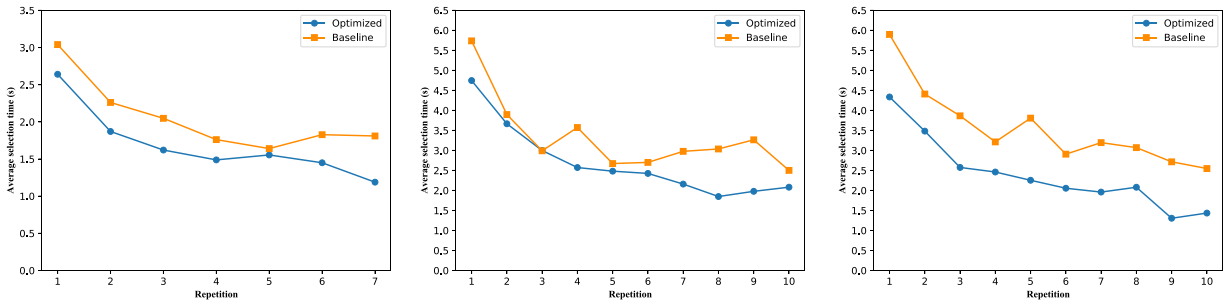



**Fig. 8.** Average selection times.

Firefox were 1.84 (SD: 2.18) for the optimized and 2.07 (SD: 2.25) for the commercial design, and those for Adobe Reader were 2.22 (SD: 2.10) for the optimized and 2.68 (SD: 2.72) for the baseline design. Statistical testing yielded a significant difference in favor of the optimized design in the case of Notepad and of Firefox. The effect was not significant for Adobe Reader. Nevertheless, the average ST for the optimized Adobe Reader was 1.22 s less than the baseline value.

The change in selection performance over time is depicted in Fig. 8. All trend-lines in these Figures are default second order polynomial functions. They consistently show a decrease in average selection time for both the baseline and the optimized menu. The optimized one showed better performance at the end of the experiment and in some cases also initially.

To understand learning effect more closely, we report average




(a) Notepad (b) Mozilla (c) Adobe

**Fig. 9.** Average selection time per command as a function of number of repetitions.

| Tab 1 | Tab 2 | Tab 3 | Tab 4 | Tab 5 |
|---|---|---|---|---|
| New | Undo | Word-Wrap | Status-Bar | View-Help |
| Open | Cut | Font | | About-Notepad |
| Save | Copy | | | |
| Save-As | Paste | | | |
| Page-Setup | Delete | | | |
| Print | Find | | | |
| Exit | Find-Next | | | |
| | Replace | | | |
| | Go-To | | | |
| | Select-All | | | |
| | Time-Date | | | |

(a) Classic (default) Menu layout for Notepad™ application

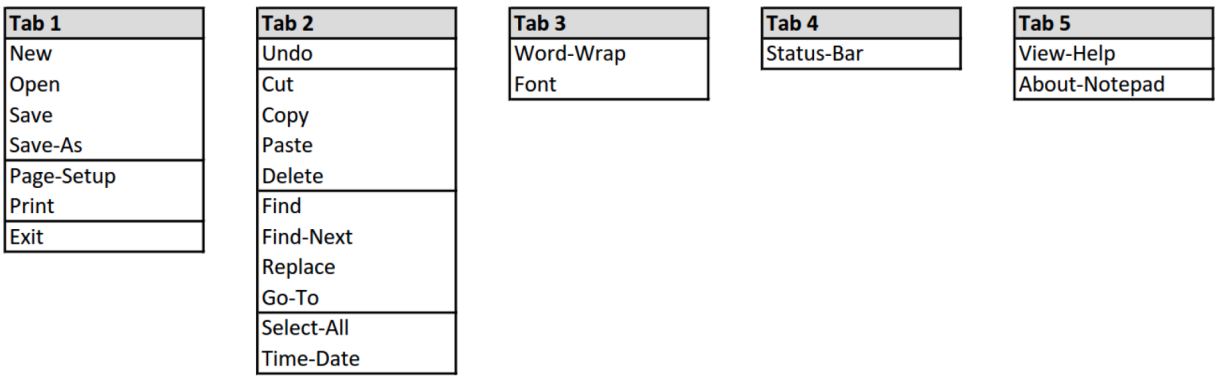
| Tab 1 | Tab 2 | Tab 3 | Tab 4 |
|---|---|---|---|
| New | Undo | Word-Wrap | Status-Bar |
| Open | Cut | Font | View-Help |
| Save | Copy | | About-Notepad |
| Save-As | Paste | | |
| Page-Setup | Delete | | |
| Print | Find | | |
| Time-Date | Find-Next | | |
| Exit | Replace | | |
| | Select-All | | |
| | Goto | | |

(b) Menu adapted for a "Novice" user profile.

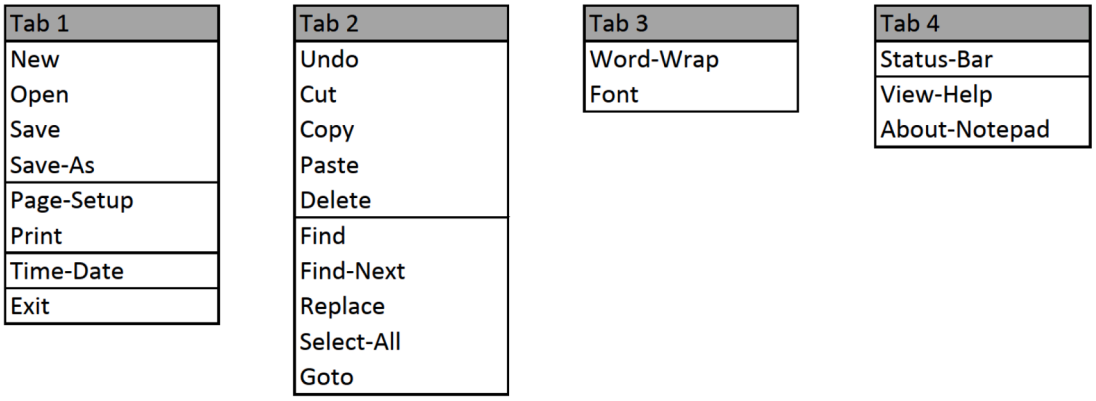
| Tab 1 | Tab 2 | Tab 3 | Tab 4 | Tab 5 |
|---|---|---|---|---|
| Find | Cut | Word-Wrap | Copy | View-Help |
| Paste | Replace | Font | Undo | About-Notepad |
| Save | Find-Next | Open | Status-Bar | |
| Save-As | Select-All | | | |
| Page-Setup | Delete | | | |
| Print | Goto | | | |
| New | Time-Date | | | |
| Exit | | | | |

(c) Menu adapted for a "Expert" user profile. Note the substantial rearrangement where association is sacrificed for performance

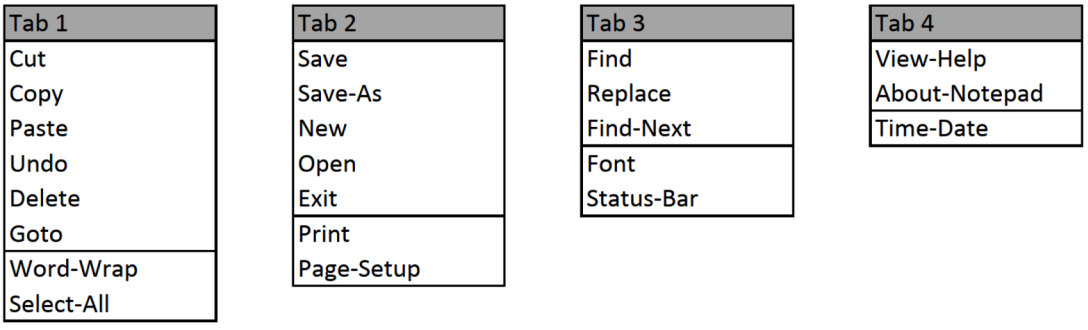
**Fig. 10.** Menu transition from Novice to Expert for Notepad™ application..




(a) Default menu layout for Notepad™ application




(b) Close-to-original layout. Performance does not improve significantly, but changes are more modest




(c) A farther-from-original layout. Performance improves significantly

**Fig. 11.** Gradual adaptation for Notepad™ application.

selection time *per command* in Fig. 9. With increasing repetitions per command, the optimized and baseline menus exhibit similar drop in performance, however the optimized menu shows a more stable trend and an overall lower ST.

## 8. Personalization and adaptation

The approach is not limited to one-shot computational design. In this section, we discuss two further applications in personalization and adaptation of a menu system.

### 8.1. Personalization

In personalization, a menu layout is custom-designed for an individual or a group of users with shared characteristics. Some software allows manually customizing menus. For example, the Eclipse software system arranges its commands and menus in one way (Perspective) for a "developer" and in a very different way for "tester". Our approach makes it possible to do personalization automatically when user data is available.

We illustrate this point using the Notepad application. A novice and casual user of Notepad would presumably use the common elementary

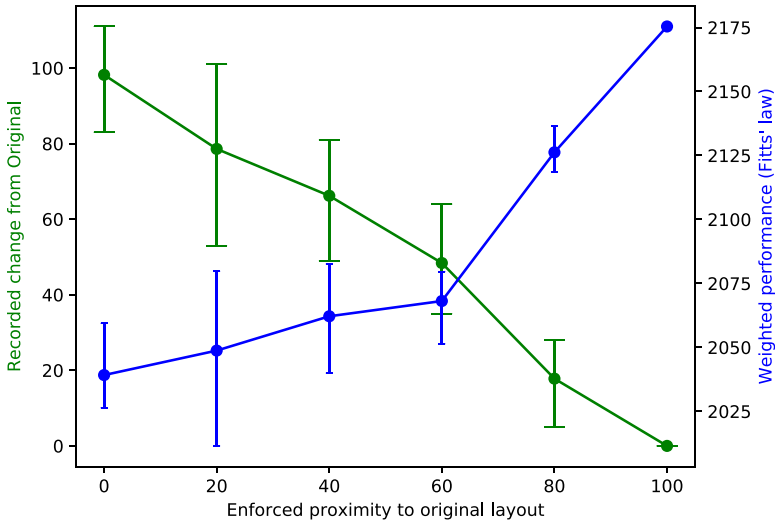



**Fig. 12.** Comparing benefit gained (performance) versus Change from original (loss of familiarity).

features such as 'Open, Save, Cut, Copy, Paste' and may also need 'Help' often. On the contrary, an expert would possibly know keyboard shortcuts for most common commands (due to extensive experience and familiarity).

So, we expect that experts would rather only need to use the Menubar to pick rarely used advanced commands such as 'Word-wrap' and 'Font'. This observation leads to two entirely different usage patterns for the two sets of users – this manifests as two different sets of frequency values for usage of commands. Fig. 10 shows the menu layouts recommended for a novice and an expert.

### 8.2. Adaptation

We can also support gradual adaptation of menus. Given a menu and user data, we propose local changes to it in order to improve usability and simultaneously make it learnable. Consider that a specific user is very well acquainted with the menu layout of an application. If we propose a new layout that is drastically different from the existing one, the user will require substantial effort in 'unlearning' the previous layout and learning a new one.

To minimize this retraining effort and still provide a better layout, we propose an adaptation method building on the IFT-based optimizer. Consider that we quantify the logical difference between two different menu layouts (that involve the same commands). We propose this logical difference as the weighted sum of the tab position change and the row number change of every command. Further, we constrain our computational menu design procedure to search within a specified vicinity of the previous layout. The resulting formulation is based on the following decision variables:

$\Pi_i$ = Change in tab position for command $i$ from original layout to new layout
$\Xi_i$ = Change in row position for command $i$ from original layout to new layout

In conjunction with $\Pi$ and $\Xi$, we also use decision variables from Sections 4 and 5. Then we compute the changes in the menu layout by using constraints such as:

$$\Xi_i >= \sum_r rR_i^r - \overline{R}_i \ \forall i \in N \tag{8}$$

$$\Xi_i >= -\sum_r rR_i^r + \overline{R}_i \ \forall i \in N \tag{9}$$

Here $\overline{R}_i$ is the row position of command $i$ in the original layout. The objective function includes as additional term to minimize the change from the original formulation, such as:

$$w\left(\sum_{i\in N}(\Pi_i + \Xi_i)\right) + (1-w)(\text{Performance objective from MRF or IFT})$$

Here $w$ is a preference term showing proximity to the original layout. This formulation forces the optimizer to find better-performing layouts that are not too much different from the existing layout. Fig. 11 illustrates this point by showing the Notepad menu gradually changed with increasing distance from the original layout.

To appreciate the impact of this gradual adaptation of menu layouts, consider Fig. 12. This shows the gradual adaptation of the menu layout where we balance the proximity to the original layout against the objective of getting better performance. In this Figure, we intend to demonstrate that our approach supports controlling the amount of change made for adaptation of a menu layout. We vary the proximity of an adapted layout to the original layout (OX axis), and see what performance improvement we can achieve with this proximity constraint (OY axis). The blue line shows the improvement in performance value (Fitts' law). The green line shows the loss of familiarity (change from original). The results reported here are for the Notepad menu.

## 9. Discussion and conclusions

This paper has contributed to the study of algorithmic methods for computational design of menus. The design of menu systems strongly impacts the usability and learnability of the computing application. However, currently menu design remains a manual activity almost exclusively; there is no well-established or commonly accepted computational technique to automatically generate or refine menus. The absence of an effective mathematical model for a menu hierarchy – such

as the minimal representative formulation (MRF) proposed here – has made it difficult to test for reliable objectives and to distinguish a good solution from a poor one. To the best of our knowledge, no earlier approach provides guarantees regarding the solution quality, yields a hierarchically organized menu suitable for large command sets, is not over-determined by previous designs (e.g., on account of a data-driven approach to the objective function), does not require much input (relative to the frequency of each command and pair-wise association scores), can be used for one-shot design as well as adaptation, and is computationally efficient for large menus.

Our contribution through the MRF is to enable a compact, flexible and purely linear IP formulation for solving the assignment and set covering problems simultaneously and within reasonable computation effort, thereby warranting application in regular menu designs. The MRF approach provides enough flexibility in defining the objective function to cover the wide variety of factors involved in menu design (the total number of tabs, the length of individual tabs, the number of groups on a tab, the length of individual groups, intra-group and intra-tab associativity, frequency of usage of the commands, etc.). The MRF supports diverse evaluative functions; this is valuable for researchers because various hypotheses can be tested with relative ease. Specifically, evaluation functions can be expressed that refer not only to the position of any item (like in the assignment-based approach) but also to its membership of a set such as a tab. We implemented a 'classical' objective from the previous literature and showed that solutions for the resulting mixed-integer programs can be found within reasonable computational effort. The results yielded by the two-fold-objective approach – while not entirely unreasonable or impractical – suffer from a few shortcomings, such as non-intuitive placement of commands.

Our novel information-foraging-based approach addresses these problems; it addresses users' decisions in zooming in versus skipping menus when searching for a target. The layouts resulting from the IFT-based (information forage theory) approach appear to be more balanced, better organized (especially lead items), and aligned more closely with expectations regarding command placement. Although the original form of IFT involves non-linear models of user behavior, we have demonstrated that a simpler, purely linear, MIP-based (mixed-integer programming) approach yields good results within reasonable computational effort. Empirical evaluations suggest that computationally produced menus can be on par with commercial designs as long as the input data (here, frequencies of selection) reflects actual usage. However, our empirical evaluation assumed that the usage frequencies match those used to run the optimizer. Future work should explore how sensitive the outcomes are to a mismatch between inputs and actual use. Moreover, the problem of how to computationally produce labels for formed groups and tabs remains an open problem.

The immediate conclusion is that IFT can be used as an evaluative function in computational menu design with good results. In conjunction with the MRF, this offers a rigorous, coherent yet flexible new framework for computational menu design. Our decision variables and the resulting constraints work with standard commercial MIP solvers and do not require any specialized contributions; for example, we do not require any specialized decompositions, relaxations, or column generation techniques (which are often used to enhance MIP performance). This opens possibilities for utilizing more complex evaluation functions with relatively lesser effort. In future, to make the method available for interactive design tools, we will explore heuristic variants and relaxations of the IFT-based approach, which may allow interactive-level performance with large task instances. Another limitation to be overcome is related to the nature of the task instances. Even if filling-in an association matrix constitutes only about an hour's work for a reasonably large menu system, it may not be practical for some use cases (for instance, in agile or rapid development cycles). We will explore word embeddings and other machine learning approaches that can automatically discover command-pair associations from data. Moreover, the scope of design decisions covered needs more attention. Further work should examine other decisions in menu design, such as label selection and shortcut assignment, and expand from tabbed/grouped menus to other types.

In our opinion, the general class of exact numerical optimization techniques holds promise for hierarchically organized user interfaces more generally. More research is needed to build on this finding. Many user interfaces are organized as trees or graphs navigated by selecting from proximally available options (Pirolli, 2007). The MRF offers a natural representation for the key decisions, such as which item to assign to which display, in which group. Deepening hierarchies can be addressed by recursively adding decisions. We found also that the sample–discard–explore formulation of IFT captures an essential aspect of a navigating user's decision-making. This is an improvement over previous work in user interface optimization, which has utilized mainly non-hierarchical evaluation functions (Oulasvirta and Karrenbauer, 2018). The linear reformulation proposed here is also sufficiently efficient and avoids resorting to meta-heuristic techniques. However, more work is needed to address the different modalities of menu access, such as short cuts and context menus, which add redundancy to the optimization problem and require users to learn more complex strategies of menu use. Moreover, the naming of tabs remains an open problem. While word embeddings produce good results for pair-wise association scores, finding a descriptive label for a tab may require considering other types of semantics, such as part–whole relationships.

The code presented in this paper is made available via our project page.

## CRediT authorship contribution statement

**Niraj Ramesh Dayama:** Conceptualization, Methodology. **Morteza Shiripour:** Software. **Antti Oulasvirta:** Conceptualization, Supervision, Writing - review & editing. **Evgeny Ivanko:** Investigation, Writing - review & editing. **Andreas Karrenbauer:** Supervision.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Appendix A

### *A1. Terminology*

This section provides a glossary for the terminology used in the paper (Table 3).

### *A2. Details of the MRF constraints*

The following constraints apply to the MRF:

**Table 3**
Glossary of terminology used in the paper.

| Term used | Interpretation |
|---|---|
| a,b | Fitts' law constants |
| X | Position of a command to a group |
| Y | Position of a command to a tab |
| Q | Position of a group on a tab |
| Z | Commands in the same group |
| W | Command on the same tab |
| R | Row position of a command in a tab |
| t | The time required to reach a command |
| S | Proceeding groups |
| $\mathbb{S}$ | First group on each tab |
| $\xi$ | Used groups |
| $\beta$ | Used tabs |
| $\Theta$ | Position of groups before a specific group |
| P | Starting position of a group within a tab |
| $\lambda$ | Weight of importance of each objective function |
| $\mathbb{A}$ | Pairwise association of commands |
| U | First command of a group |
| $\Phi$ | Searching effort for finding a command |
| $\alpha$ | True-positive time of finding a command |
| $\sigma$ | False-positive time of finding a command |
| $\delta$ | False-negative time of finding a command |
| $\Omega$ | Penalty for a command if it is placed on a non-standard tab |
| $\Pi, \Xi$ | Change in position for command $i$ from original layout to new layout |

$$t_i = \left(a + b\log_2\left(\sum_r rR_i^r + 1\right)\right) + \left(a + b\log_2\left(\sum_\tau \tau Y_i^\tau + 1\right)\right) \ldots \forall\ i \in N \tag{10}$$

This constraint is used to calculate the selection time according to the Fitts' law. The first part on the right equation is the Fitts' law for rows and the next part is for tabs.

$$|N|\xi^c \geq \sum_{i \in N} X_i^c \ldots \forall\ c \tag{11}$$

$$\sum_{i \in N} X_i^c \geq \xi^c \ldots \forall\ c \tag{12}$$

$$|N|\beta^\tau \geq \sum_{i \in N} Y_i^\tau \ldots \forall\ \tau \tag{13}$$

$$\sum_{i \in N} Y_i^\tau \geq \beta^\tau \ldots \forall\ \tau \tag{14}$$

The above constraints ensure that any group or tab will be marked for use if and only if it actually includes at least one command.

$$W_{ij} \geq Z_{ij} \ldots \forall i, j \in N \tag{15}$$

This constraint ensures that if two commands are in the same group, they share the same tab.

$$\sum_\tau Q^{c\tau} = \xi^c \ldots \forall c \tag{16}$$

$$\sum_\tau Y_i^\tau = 1 \ldots \forall i \in N \tag{17}$$

$$\sum_c X_i^c = 1 \ldots \forall i \in N \tag{18}$$

Every command must be placed on exactly one tab and should be present in exactly one group. Similarly, every group (if marked for use) should be placed on exactly one tab.

$$\beta^\tau \geq Q^{c\tau} \ldots \forall c, \tau \tag{19}$$

If a group is to be placed on a specific tab, then that tab must be marked as non-empty.

$$\sum_r R_i^r = 1 \ldots \forall i \in N \tag{20}$$

Every command must be placed on exactly one row.

$$Q^{\bar{c}\tau} \geq Q^{c\tau} + S^{c\bar{c}} + S^{\bar{c}c} - 1 \ldots \forall c, \bar{c}, \tau \tag{21}$$

Two groups marked for immediate precedence must be placed on the same tab.

$$\sum_i R_i^r \leq \sum_\tau \beta^\tau \ldots \forall r \tag{22}$$

The total number of commands on any row is less than or equal to the number of tabs being used.

$$\sum_i R_i^r \leq \sum_i R_i^{r-1} \ldots \forall r : \; r \geq 1 \tag{23}$$

No intermediate rows should be left unoccupied (avoid holes).

$$\sum_\tau \beta^\tau = \sum_c \mathbb{S}^c \ldots \forall c \in C \tag{24}$$

The number of tabs for use is equal to the number of groups that can be the starting (topmost) groups.

$$\xi^c = \sum\nolimits_{\bar{c} \in C} S^{\bar{c}c} + \mathbb{S}^c \ldots \forall c \in C \tag{25}$$

If a group is used, it must either be the topmost group on its tab or be preceded by another group.

$$R_i^r + R_j^r + W_{ij} \leq 2 \ldots \forall i, j \in N \tag{26}$$

No two commands on a tab may share the same row number. We can enforce a similar constraint for any pair of commands within a single group.

$$\sum_r r R_i^r \leq P^c + \sum_j X_j^c + |N|\left(1 - X_i^c\right) \ldots \forall i \in N, \forall c \in C \tag{27}$$

$$\sum_r r R_i^r \geq P^c - |N|\left(1 - X_i^c\right) \ldots \forall i \in N, \forall c \in C \tag{28}$$

The above constraints keep the row number for any given command within the bounds of its designated group.

$$P^{\bar{c}} \geq P^c + \sum_{i \in N} X_i^c - |N|\left(1 - \Theta^{c\bar{c}}\right), \forall c, \bar{c} \in C \tag{29}$$

This constraint ensures that no two groups from any tab can overlap each other.

$$X_i^c \geq X_j^c + Z_{ij} - 1 \ldots \forall i, j \in N, \forall c \in C \tag{30}$$

This constraint interconnects the $X$ variable with the $Z$ variable to ensure the logical constitution of groups. A similar constraint is enforced for the $Y$ and $W$ variables.

### *A3. The baseline designs for all data instances*

This section presents the baseline (existing) menu designs as seen in the commercial versions of Notepad, Acrobat, and Firefox applications (Figs. 13–15).

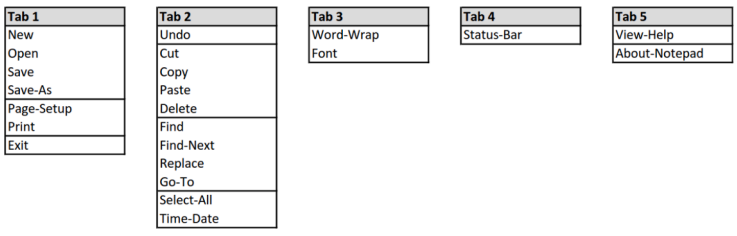



**Fig. 13.** Notepad baseline design.

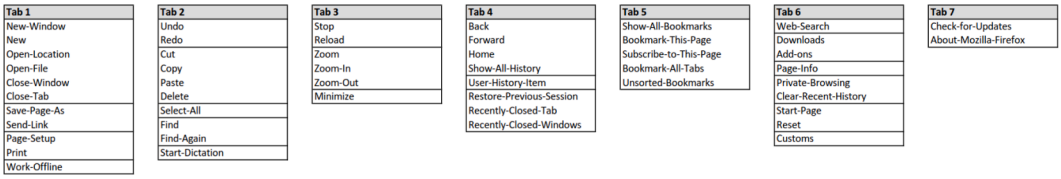



**Fig. 14.** Mozilla Firefox baseline design.

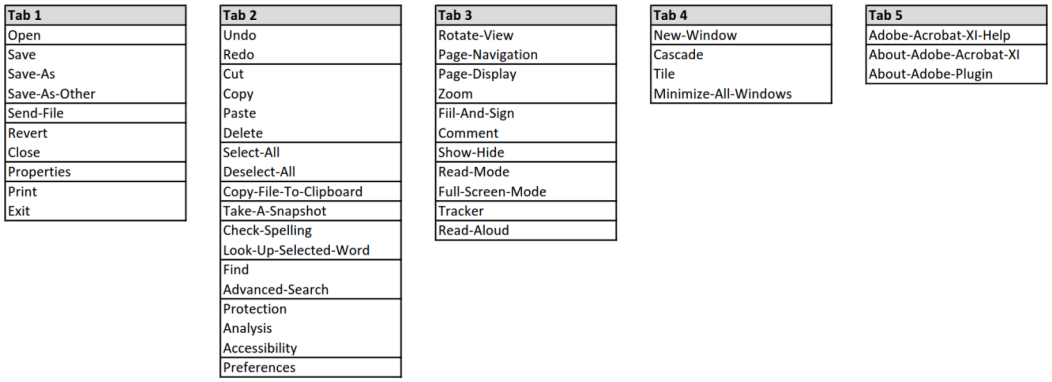



**Fig. 15.** Adobe reader baseline design.

### *A4. Preferred locations for specific commands*

Menu systems in software applications have traditionally followed some unwritten norms regarding placement of certain key command groups in specific tabs. For example:

1. Commands to open/create a new session, file or activity are predominantly located in the first (leftmost) tab of the menu.
2. Commands to save/close an ongoing session, file or activity are predominantly located in the first (leftmost) tab of the menu.
3. Commands to manipulate the clipboard by Cut/Copy/Paste some parts of an ongoing file or activity are never located in the first (leftmost) tab or the last (rightmost) of the menu. Rather, these commands are typically in the tab that is second from left.
4. Commands to access 'Help' topics, to read information *about* the current software application, find its version or to update that application are predominantly located in the last (rightmost) tab of the menu.

While the four norms written above have not been formally documented in standard design guidelines, the authors posit that it is rare to find common professional software application which do not follow these norms. We postulate two reasons behind such norms: (1) Designers of some seminal software applications may have logically chosen the placement of these command groups. (2) The ingrained practice has continued unchallenged and become an essential part of user expectation. Effectively, a practice was started and no one saw any major reason to change it.

The norms written above are extremely generic and are not restricted to any specific domain or topic. It is conceivable that specific domains, topics or business areas will have more such norms specific to practitioners of that topic. We assume that such information is available as input parameter for our menu design process. For a few (say around 5–10% of total) commands, we assume that the preferred location specification $\mathbb{L}$ is provided in terms of the tab number where the command be preferably expected.

## References


Ahlström, D., 2005. Modeling and improving selection in cascading pull-down menus using Fitts' law, the steering law and force fields. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 61–70. https://doi.org/10.1145/1054972.1054982. In this issue.

Anjos, M.F., Vieira, M.V., 2017. Mathematical optimization approaches for facility layout problems: the state-of-the-art and future research directions. Eur. J. Oper. Res. 261 (1), 1–16.

Bagnall, A.J., Rayward-Smith, V.J., Whittley, I.M., 2001. The next release problem. Inf. Softw. Technol. 43 (14), 883–890.

Bailly, G., Lecolinet, E., Nigay, L., 2017. Visual menu techniques. ACM Comput. Surv. (CSUR) 49 (4), 60.

Bailly, G., Oulasvirta, A., Brumby, D.P., Howes, A., 2014. Model of visual search and selection time in linear menus. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 3865–3874.

Bailly, G., Oulasvirta, A., Kötzing, T., Hoppe, S., 2013. MenuOptimizer: interactive optimization of menu systems. Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology. ACM, pp. 331–342.

Balintfy, J.L., 1964. Menu planning by computer. Commun. ACM 7 (4), 255–259.

Brumby, D.P., Howes, A., 2004. Good enough but i'll just check: web-page search as attentional refocusing. ICCM, pp. 46–51.

Byrne, M.D., 2001. ACT-R/PM and menu selection: applying a cognitive architecture to HCI. Int. J. Hum.-Comput. Stud. 55 (1), 41–84.

Byrne, M.D., Anderson, J.R., Douglass, S., Matessa, M., 1999. Eye tracking the visual search of click-down menus. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 402–409.

Card, S.K., Moran, T.P., Newell, A., 1980. The keystroke-level model for user performance time with interactive systems. Commun. ACM 23 (7), 396–410.

Chen, X., Bailly, G., Brumby, D.P., Oulasvirta, A., Howes, A., 2015. The emergence of interactive behavior: a model of rational menu search. Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. ACM, pp. 4217–4226.

Cockburn, A., Gutwin, C., 2009. A predictive model of human performance with scrolling and hierarchical lists. Hum.-Comput. Interact. 24 (3), 273–314.

Cockburn, A., Gutwin, C., Greenberg, S., 2007. A predictive model of menu performance. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 627–636.

Danilenko, A., Goubko, M., 2013. Semantic-aware optimization of user interface menus. Autom. Remote Control 74 (8), 1399–1411.

Dantzig, G.B., 1990. The diet problem. Interfaces 20 (4), 43–47.

Drira, A., Pierreval, H., Hajri-Gabouj, S., 2007. Facility layout problems: a survey. Annu. Rev. Control 31 (2), 255–267.

Fu, W.-T., Pirolli, P., 2007. SNIF-ACT: a cognitive model of user navigation on the world wide web. Hum.-Comput. Interact. 22 (4), 355–412.

Gen, M., Cheng, R., 2007. Facility Layout Design Problems. John Wiley & Sons, Inc., pp. 292–329
Goldberg, D.E., Deb, K., 1991. A comparative analysis of selection schemes used in genetic algorithms. Foundations of Genetic Algorithms, 1. Elsevier, pp. 69–93.
Golovine, J.-C., McCall, J., Holt, P.O., 2010. Evolving interface designs to minimize user task times as simulated in a cognitive architecture. Evolutionary Computation (CEC), 2010 IEEE Congress on. IEEE, pp. 1–7.
Goubko, M., Danilenko, A., 2012. Mathematical model of hierarchical menu structure optimization. Autom. Remote Control 73 (8), 1410–1423.
Goubko, M.V., Danilenko, A.I., 2010. An automated routine for menu structure optimization. Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems. ACM, pp. 67–76.
Hornof, A.J., Kieras, D.E., 1997. Cognitive modeling reveals menu search in both random and systematic. Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 107–114.
Karrenbauer, A., Oulasvirta, A., 2014. Improvements to keyboard optimization with integer programming. Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology. ACM, pp. 621–626.
Lancaster, L.M., 1992. The history of the application of mathematical programming to menu planning. Eur. J. Oper. Res. 57 (3), 339–347.
Lee, E., MacGregor, J., 1985. Minimizing user search time in menu retrieval systems. Hum. Fact. 27 (2), 157–162.
Liu, W., Bailly, G., Howes, A., 2017. Effects of frequency distribution on linear menu performance. Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. ACM, pp. 1307–1312.
Mann, H.B., Whitney, D.R., 1947. On a test of whether one of two random variables is stochastically larger than the other. Ann. Math. Stat. 50–60.
Matsui, S., Yamada, S., 2008. Optimizing hierarchical menus by genetic algorithm and simulated annealing. Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation. ACM, pp. 1587–1594.
McDonald, J.E., Stone, J.D., Liebelt, L.S., 1983. Searching for items in menus: the effects of organization and type of target. Proceedings of the Human Factors Society Annual Meeting, 27. SAGE Publications Sage CA, Los Angeles, CA, pp. 834–837.
Mehlenbacher, B., Duffy, T.M., Palmer, J., 1989. Finding information on a menu: linking menu organization to the user's goals. Hum.-Comput. Interact. 4 (3), 231–251.
Nilsen, E.L., 1996. Perceptual-Motor Control in Human-Computer Interaction. Technical Report. Michigan Univ Ann Arbor Div of Research Development and Administration.
Norman, K.L., 1991. The Psychology of menu Selection: Designing Cognitive Control at the Human/Computer Interface. Intellect Books.
Oulasvirta, A., Dayama, N.R., Shiripour, M., John, M., Karrenbauer, A., 2020. Combinatorial optimization of graphical user interface designs. Proc. IEEE 108 (3), 434–464.
Oulasvirta, A., Feit, A., Lähteenlahti, P., Karrenbauer, A., 2017. Computational support for functionality selection in interaction design. ACM Trans. Comput.-Hum. Interact. (TOCHI) 24 (5), 34.
Oulasvirta, A., Karrenbauer, A., 2018. Combinatorial optimization for user interface design. Comput. Interact. 97–121.
Paap, K.R., Cooke, N.J., 1997. Chapter 24 - design of menus. In: Helander, M.G., Landauer, T.K., Prabhu, P.V. (Eds.), Handbook of Human-Computer Interaction, second ed. North-Holland, Amsterdam, pp. 533–572.
Pirolli, P., 2007. Information Foraging Theory: Adaptive Interaction with Information. Oxford University Press.
Pirolli, P., Card, S., 1999. Information foraging. Psychol. Rev. 106 (4), 643.
Pirolli, P., Fu, W.-T., 2003. SNIF-ACT: a model of information foraging on the world wide web. In: Brusilovsky, P., Corbett, A., de Rosis, F. (Eds.), User Modeling 2003. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 45–54.
Sears, A., Shneiderman, B., 1994. Split menus: effectively using selection frequency to organize menus. ACM Trans. Comput.-Hum. Interact. (TOCHI) 1 (1), 27–51.
Troiano, L., Birtolo, C., 2014. Genetic algorithms supporting generative design of user interfaces: examples. Inf. Sci. 259, 433–451. https://doi.org/10.1016/j.ins.2012.01.006.
Troiano, L., Birtolo, C., Armenise, R., 2016. Searching optimal menu layouts by linear genetic programming. J. Ambient Intell. Humaniz. computing 7 (2), 239–256.
Troiano, L., Birtolo, C., Armenise, R., Cirillo, G., 2008. Optimization of menu layouts by means of genetic algorithms. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). In: LNCS, 4972, pp. 242–253. https://doi.org/10.1007/978-3-540-78604-7_21.Conference of 8th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2008 ; Conference Date: 26 March 2008 Through 28 March 2008; Conference Code:72583