Optimization Approaches to Adaptive Menus

Zhengwu Lu

Aalto University, School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology

Espoo 18th May 2017

Supervisor:

Professor Antti Oulasvirta

Advisor:

Professor Antti Oulasvirta

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY	ABSTRACT OF THE
SCHOOL OF ELECTRICAL ENGINEERING	MASTER'S THESIS

Author: Zhengwu Lu

Title: Optimization Approaches to Adaptive Menus

Date: 18.05.2017 Language: English Number of Pages: 82 + 7

Department: Department of Communications and Networking

Professorship: Communications Engineering

Supervisor: Prof. Antti Oulasvirta

Advisor: Prof. Antti Oulasvirta

Graphical menus perform as vital components and offer essential controls in today's graphical interface. However, few studies have been conducted to modelling the performance of a menu. Furthermore, menu optimization methods previously proposed have been largely concentrating on reshaping layout of the whole menu system.

In order to model menu performance, this thesis extends the Search-Decision-Pointing model by introducing two additional factors, i.e. the cost function and semantic function. The cost function is a penalty function which decreases the user expertise regarding a menu layout according to the degree of modification done to the menu. The semantic function is a reward function which encourages items with strong relations be positioned close to each other. Centered on this menu performance model, several optimization methods have been implemented. Each method focuses on improving menu performance by applying distinctive strategies, such as increasing item size or reducing item pointing distance.

Three test cases have been exercised to evaluate the optimization methods in a simulated software which displays graphical user interfaces and emulates the menu utilization of real users. The results of test cases reveal that the menu performance has been successfully improved in all test cases by the fundamental heuristic search algorithm. Moreover, other optimization methods have been able to further increase menu performance ranging from 3% to 8% depending on test cases. In addition, it is identified that increasing the size of an item offers surprisingly little benefit. Conversely, reducing item pointing distance has greatly improved menu performance. Moreover, positioning items by their semantic relations may also enhance group saliency. On the other hand, optimization methods may not always succeed in providing usable menus due to design constraints. Hence, menu performance optimization shall be carefully exercised by considering the entire graphical user interface.

Keywords: user interface, graphical menu, model-based design and optimization, heuristic algorithm, multi-objective function

Acknowledgements

The experience of this thesis has been a fantastic journey. I would like to give my gratitude and thanks to all who have supported me in this study. First and foremost, I would like to thank my supervisor and advisor, Professor Antti Oulasvirta, for offering me this opportunity to work on this topic and aiding me with excellent guidance and valuable comments whenever I need them.

I would like to also express my heartfelt thanks to my family for their unweaving supports. My parents, who have been providing great support throughout the master's study of mine. My son and my daughter, who embrace me with joy and happiness. My wife, who lends me the greatest strength and grants me with everlasting love.

Espoo, 18th May 2017

Zhengwu Lu

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

This thesis studies the fundamental aspects regarding menu performance. Furthermore, this thesis applies a multi-objective function in order to evaluate performance of menus. In addition to this, several optimization methods have been designed based on menu performance models for the purpose of evaluating the effects regarding altering essential elements of a menu structure. Moreover, this thesis work introduces two simulator software on which the experiments of this thesis are performed. The overall goal of the simulators is to execute experiments according to predefined parameters in order to evaluate the feasibility and performance of optimization methods.

1.1 Research background and motivation

Graphical menus have been performing a vital role ever since the introduction of graphical user interface (GUI) for software and applications. Over the last years, applications are continuously expanding and providing an increasing amount of features and functionalities. In order to resonate in harmony with the rapid advancing applications, graphical menus have also been growing regarding various aspects, such as size, complexity and representation style. Because of this, utilization of graphical menus have been increasingly becoming difficult. Hence, it has been essential to deliver a graphical menu with clear layout and fast access to menu items, especially for experts whose work is largely depends on the utilization of graphical menus.

Due to the wide utilization of graphical menu, numerous studies have been conducted regarding novel menus [2], however, few researches have been performed to evaluate and predict menu performance by changing its layout or structure. In addition to this, these researches are largely based on the theoretical study of Fitts' Law regarding menu item pointing time [7]. As a result, various significant menu performance factors are rather neglected, such as visual search time, decision time, menu learnability as well as semantic relations between menu items [2] [26]. Furthermore, few studies have been performed to improve menu performance by implementing optimization algorithms. Consequently, today's menu layouts are largely constructed based on designer's point of view without mathematical modelling [23]. However, designing an explicit menu system may reveal many challenges because the amount of alternative designs for a menu grow as a super-exponential function of the number of items in a menu systems [11]. That is, a menu which holds n commands may theoretically be organized in n! ways. Because of this, optimization approaches which attempt to reorder menu items by relocating them according to their frequency of usages, may appear to be rather sufficient for menu system with few items. However, these approaches begin to rapidly lose their efficiency for large sized menu systems. [2] [11] [26]

1.2 Research objective

The research objective of this thesis is to improve menu performance of linear menu in an editor software. A linear menu is based on linear layout where menu items are linearly organized and close to each other. Because of this, linear menu may reduce eye movements incurred by utilizing serial search strategy [12] [13]. The editor software here is assumed to be the Notepad++ [9]. On the other hand, the study of this thesis may be also applied to other software with similar graphical user interface, such as the LibreOffice.

Concentrating on the research objective, this thesis evaluates and improves menu performance based on a multi-objective function composed of the Search-Decision-Pointing (SDP) model [2] and a semantic model [11] [26]. More specifically, this thesis aims to design and implement menu optimization methods which may decrease the average time expenditure for searching and executing a menu item in a menu.

Ultimately, this thesis evaluates the feasibility of menu optimization methods for the editor software based on the objective function of this thesis in simulated environments.

1.3 Research approach

In order to accomplish the research goals, this thesis initially studies various menu performance factors. Based on these factors, several menu performance models are also analyzed. Subsequently, these menu performance models form the objective function of this thesis. Centered on the objective function, several optimization methods are designed based on a variety of menu optimization strategies. These methods attempt to identify the most appropriate menu design by modifying various menu structural factors, such as menu layout, item positioning, item size and item saliency.

Two simulator software are programmed to perform and visualize this thesis work. They are referred to as the *Background Simulator* and *Graphical Simulator*. Additionally, three distinctive test cases are crafted and executed by the simulator software in order to evaluate the feasibility and performance of these optimization methods.

Ultimately, this thesis study is performed centered on model-based design approach by which menu performance and optimization methods are evaluated.

1.4 Research scope

Centered on the research objective, this thesis defines the following research scopes,

- A multi-objective function [8] which is utilized to evaluate menu performance. This function differs with the ones in previous studies by considering both the SDP model and the semantic model. A weight factor is applied to these models in order to balance the overall menu performance.
- A linear menu where menu items are horizontally positioned as buttons.
- A graphical user interface simulator which presents a linear menu and holds a similar complexity as a medium-sized editor software. For instance, Notepad++. [9]
- The optimization methods are designed according to menu optimization strategies and are able to perform the following operations,

- Change the position of items
- Regroup items
- Change the display size of items
- Increase the saliency items

1.5 Outline

- **Chapter 2 Related Work** discusses some previously conducted researches regarding menu performance optimization.
- **Chapter 3 Theoretical Framework** discusses the fundamental theory studied. Moreover, the objective function of this thesis is also presented.
- **Chapter 4 Software and Optimization Design** describes design and operation principle of optimization methods. Furthermore, the execution process of simulator software is also introduced.
- **Chapter 5 Evaluation** performs the evaluation of optimization methods by conducting three distinctive test cases. The results of optimization methods are analyzed and discusses in detail.
- **Chapter 6 Conclusion and Future Work** displays the key findings of Chapter 5 and discusses potential future research works.

Chapter 2. Related work

Numerous researches have been studied regarding menu performance optimization, including item frequency based optimization, pointing time (Fitts' law) based optimization and heuristic algorithm based optimization. This chapter discusses some of the previously conducted studies related to this thesis work.

2.1 Item frequency based optimization

In 1994, Sears and Shneiderman have introduce an approach to increase menu performance by adapting menus. This approach is defined as *split menu*. [1] In their research, a menu is separated into two partitions by a separator line to improve menu performance. Items that are frequently selected are relocated to the top part of menu before the split line for faster access while rarely accessed items are positioned in the lower part of menu. This approach results in improved menu performance. Furthermore, in later commercially implemented version of *split menus*, promoted items are copied to the top part of menu instead of being moved, hence, these items may be accessed from both the top and lower part of menu. As a result, this design has further resulted in improved user satisfaction. An example of this kind of improved version of *split menu* may be observed in the "Font" menu in Microsoft Office. [1][13]

Sears and Shneiderman have modelled the expected benefit offered by the *split menu* based on a log-linear formula which is demonstrated as the following,

$$Expected\ Benefits = S_{HF} \cdot \sum_{HF\ items} f(x) * [\log_2\left(L_T(x)\right) - \log_2(L_S(x))]$$

$$+ S_{LF} \cdot \textstyle\sum_{LF\ items} f(x) * [L_T(x) - L_S(x)]$$

Where,

- $f(x)$ is the frequency or probability of being selected of item x. The probability sum of all menu items equals 1.
- $L_T(x)$ and $L_S(x)$ represent the location of item x in traditional (lower) part of menu and in the split part of menu respectively.
- S_{HF} and S_{LF} are defined as the slopes of regression equations for items with high and low frequency respectively. They are centered on $\log_2(L_T(x))$ and $t_T(x)$, where $t_T(x)$ is the time required to select item x in the lower part of menu.

An example of the *split menu* designed by Sears and Shneiderman is demonstrated as the following,

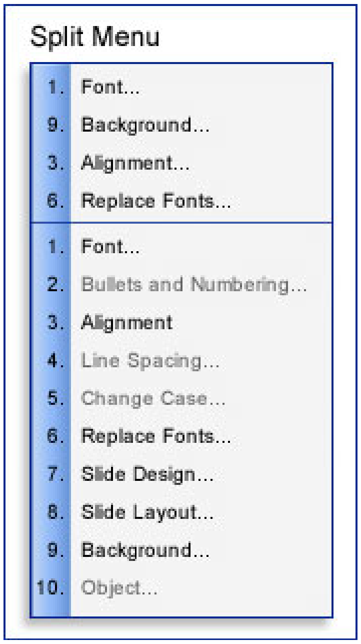


Figure 1: Split menu [13]

Based on the design of the *split menu,* Sears and Shneiderman have also observed that users may quickly memorize the locations of frequently accessed items. This allows them to select items without searching for the desired item. Because of this, Sears and Shneiderman have suggested that the selection time of familiar items are dominated by the time needed to move the cursor to the desired item, which may be estimated by the Fitts' law. On the other hand, items that are infrequently selected require users to iterate through the whole menu from top to bottom until the desired item is identified. As a result, the selection time of rarely accessed items may be estimated by a linear function with the length of menu and position of the desired item in the menu. [1] [13]

2.2 Fitts' Law based optimization

Ahlström has proposed another approach which attempts to improve menu selection time. In his research, the mouse cursor is attracted to a force field which assists users in navigating cascading pull-down menus in item selection tasks. The force field is implemented based on a warping algorithm which attempts to pull the cursor to the most appropriate force point depending on the item the cursor is currently pointing to. The software which utilizes the force fields tracks the movement of cursor and extract event information regarding mouse positions. These information is then continuously processed by the warping algorithm to generate force fields in real time. The coordinates for new cursor positions influenced by the force field may be calculated as the following [7],

$$n = a + s \cdot ||a - p|| \cdot \frac{f - a}{||f - a||}$$

Where,

- n is the new cursor coordinates influenced by the force field. It consists of n_x and n_y.
- a is the last active cursor position.
- p is the previous active cursor position before a.
- f is the force field location.
- s is the strength of force field.

In addition to this, Ahlström has introduced a menu performance model which is based on a combination of the Fitts' law pointing requirements and the steering law. The Fitts' law is a well-known performance model studied in information theory and has been wildly applied in the field of HCI [1] [2] [4] [11] [12] [13] [27]. Moreover, based on the Fitts' law, the steering law which is introduced by Accot and Zhai, has also been utilized in this research [7]. The menu performance model which estimates the selection time T for selecting an item of a n^{th} submenu in a cascading pull-down menu may be demonstrated as the following,

$$T = a + b\frac{nh}{w} + a + b\frac{w}{h}$$

Where a and b are empirically determined variables. Parameters w and h represent the width of parent menu and height of menu items respectively. The time incurred by movements inside a menu is divided into two parts where the first part of the formula $a + b\frac{nh}{w}$ estimates the vertical movements while the second part $a + b\frac{w}{h}$ horizontal movements. [7]

An example of force field menu may be demonstrated as the following,

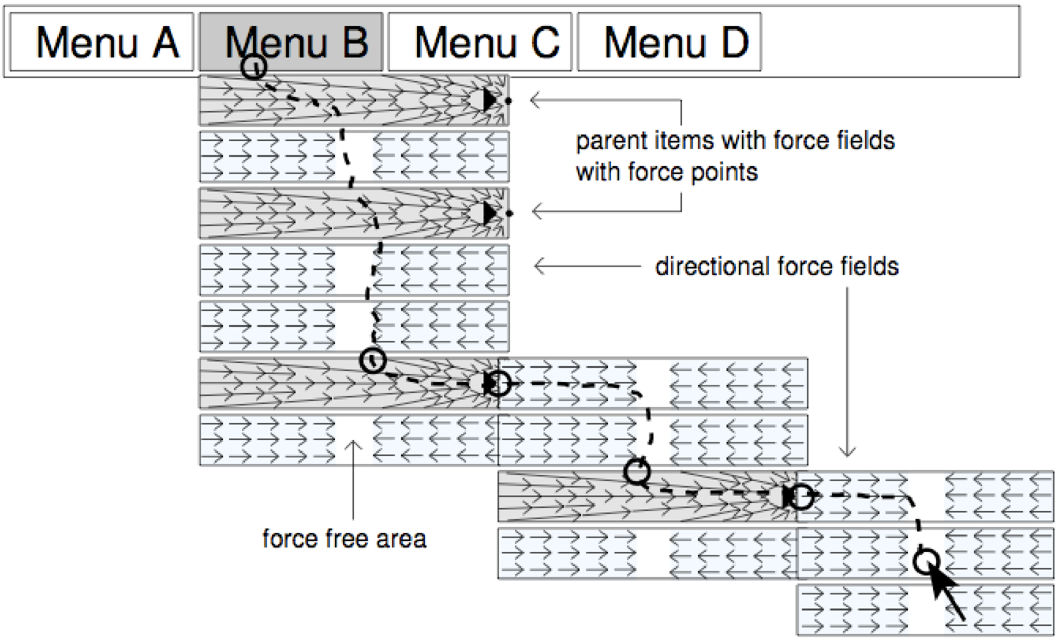


Figure 2: Force field menu [13]

According to Ahlström, the implementation of force field has been able to improve menu performance for users by average 18% compared to normal menus. However, the model utilized by Ahlström neglects the search time required to identify a desired time. As a result, the effect of user expertise has not been taken into consideration in this research. [7] [13]

2.3 Heuristic algorithm based optimization

Matsui and Yamada have designed a method to optimize hierarchical menus. Their method is based on heuristic algorithms, including simulated annealing and genetic algorithms. In their research, menu is mapped by tree structure and menu items are determined as nodes. Each node may only hold a finite number of items. The fundamental research problem defined in their study is to identify the most appropriate allocation of items in a tree structure which may minimize the objective function. The objective function utilized in their research is based on the Search-Decision-Pointing model known as the SDP model introduced by Cockburn et.al. [2] In addition to the original factors of the SDP model proposed by Cockburn et.al, Matsui and Yamada also have also introduced two new factors known as the *functional similarity* and *menu granularity*. The parameter *functional similarity* is used to as a penalty factor which attempts to minimize the distance between semantically related items. Additionally, *menu granularity* is used to balance the tree structure of a menu. That is, it attempts to prevent a node to have children of very distinctive types, for instance, to place a node that has no child together with another node that has many children into the same parent node. [2] [27]

The objective function proposed by Matsui and Yamada may be demonstrated as the following,

$$minimize \qquad f = T_{avg} + \alpha P^s + \beta P^g$$

Where T_{avg} is the average menu selection time. P^S and P^g are the penalty functions for *functional similarity* and *menu granularity* while parameters α and β are constants use to balance the performance for these two functions.

An example of menu layout comparison before optimization and after may be demonstrated as the following,

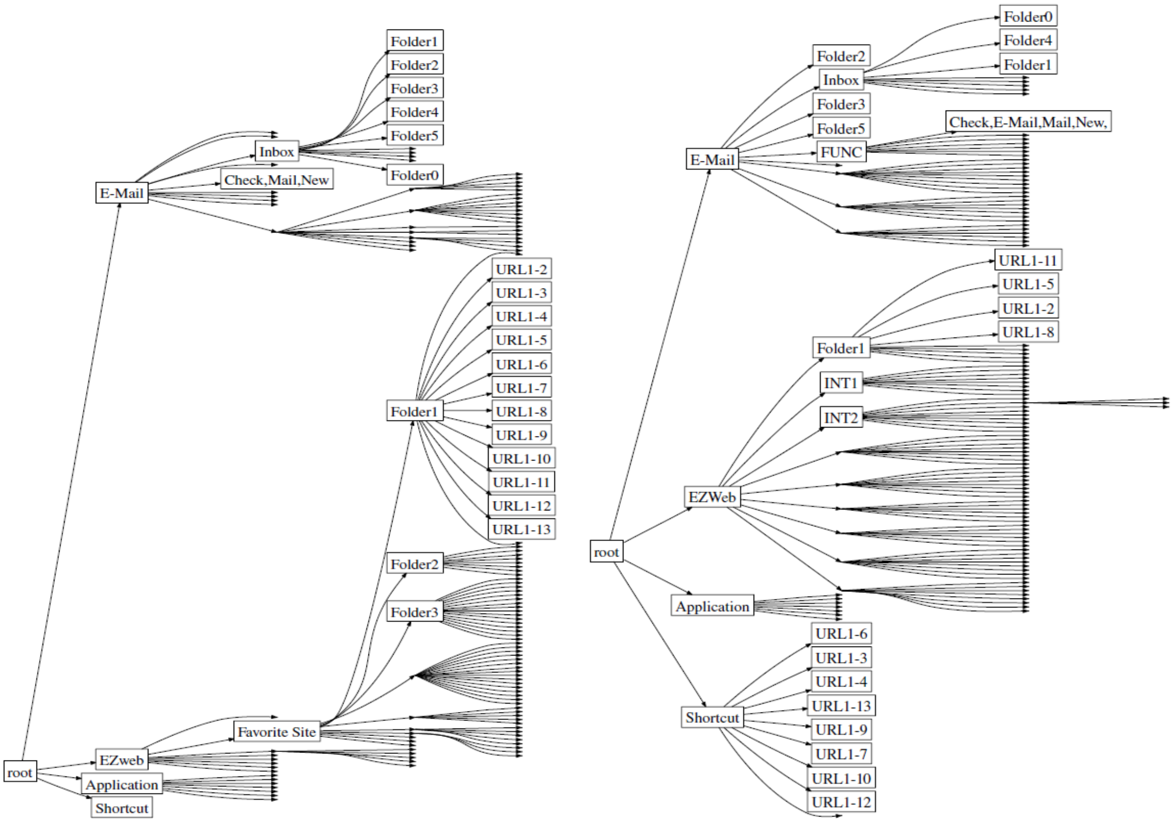


Figure 3: Menu layout comparison before optimization and after optimization [13]

Matsui and Yamada report that their algorithms have been able to decrease average menu item selection time by at least 40%, which is a significant improvement [27]. Furthermore, they indicate that their algorithms are suitable for hierarchical menus in other devices as well instead of limited to only cell phones. However, the menu performance model utilized by them does not consider the cost incurred by switching positions of an items. Additionally, the effect of menu structure modification regarding menu learnability is also ignored in their research.

Chapter summary

This chapter has discussed some of the previously conducted work regarding modelling and improving menu performance. Each model and method designed and implemented in these researches provide their own advantages as well as disadvantages. Some models focus on some of the most significant menu performance factors, such as item access frequency or item pointing time requirement. On the other hand, other models attempt to consider a variety of design factors, such as the SDP model.

Previous studies have displayed a huge amount of potential in improving menu performance. However, few research has been conducted to study the influence of considering semantic relations between menu items in combination with a predictive menu performance model. Moreover, the negative effect of modifying menu structure is also not explained. Because of these, the model performance model utilized in this thesis work is based on the SDP model. In addition to the original factors introduced by the SDP model, this thesis work proposes also several other factors, including item cost function and semantic relation function. The SDP model, objective function and other related theoretical frameworks are discussed in detail in the next chapter.

Chapter 3. Theoretical framework

This chapter discusses the theoretical framework of this thesis. The framework is composed of several distinctive concepts, including menu performance factors, the menu performance models as well as the objective function of this thesis work.

3.1 Menu performance factors

Menu performance are significantly affected by various factors, including menu length, item positioning, user expertise, pointing strategy, menu learnability as well as item semantic relations.

1. Menu length: The length of menu is defined by the number of items reside within a menu. It is unambiguous that users are naturally quicker at searching and selecting items in menus with short length. On the other hand, the search time for a given item increases as the number of items in that menu increases. As a result, long menu length may negatively affect user performance. [2] [11] [12]

2. Item positioning: The positioning of item affect both the search and selection time of user. That is, it has been studied that items located in the top of menu are easier to be selected compared to the ones in the bottom. However, in unfamiliar menus, users may have to iterate through whole menu one item a time in order to locate the desired item. Hence, the item positioning may significantly affect the search time of an item for users in menus with little to no previous experience. On the other hand, users may completely rely on their spatial memories regarding item positions in familiar menus. [2] [10] [11]

3. User expertise: Users increasingly gain experience regarding a menu layout through continuous utilization of the given menu. As a result, the performance of such menu is improved. On the other hand, users may have to relearn the menu in situations where the menu layout rapidly varies. Consequently, user expertise concept is less valuable if the menu layout is unstable. [2]

4. Pointing strategy: Pointing strategy defines the approach regarding cooperation between eye movements and mouse control. In general, users may utilize two distinctive strategies; [12]

1) Users may move the mouse cursor according to their eye movements. That is, the mouse cursor is simultaneously moved with the eyes by tracking and following the eye movement.
2) Users may initially attempt to identify and locate desired item, subsequent to this, the move cursor is moved in single movement in order to select the given item.

In the first approach, the mouse cursor is located near eye trace during searching phase, hence, the pointing time may be relatively decreased. In this thesis project, all item selections are performed by a simulator software. Because of this, it is assumed that the second approach is utilized.

5. Menu learnability: The learnability of a menu may be realized as a parameter which indicates the degree of modification caused by adjusting the menu layout or structure. A menu with high degree of learnability suggests a more immutable menu layout. That is, the menu layout has incurred few to no changes compared to a previous one. Conversely, a low degree of learnability suggests that the current menu has performed various layout changes, hence, the appearance of the current menu is significantly different compared to a recent one. [2]

Users are continuously acquire knowledge regarding a menu layout while executing menu items, as a result, they may eventually remember the entire menu layout. However, some menu layouts are rather difficult to be learned compare to others, hence, the menu learnability factor serves as a vital role to illustrate the consequence of changing parameters of menu items, therefore, it may enhance the precision of the menu performance model. [2]

6. Semantic relations: The semantic relations between items may largely affect a menu performance [11] [27] [30]. More specifically, menu items that are semantically positioned offer better performance regarding visual search compared to unordered ones [1] [30]. Because of this, the semantic parameter is considered as an influential factor in menu performance.

3.2 Menu performance model

As discussed, various menu performance models have been proposed in previous studies [1] [7] [22]. Indeed, numerous performance factors have been evaluated in these models, however, few of them consider different performance factors as a joint function on a comprehensive level. Hence, based on the menu performance factors presented in previous section, the menu performance model utilized in this thesis work is based on the Search-Decision-Pointing (SDP) model introduced by Cockburn et.al. [2]. The SDP model is referred as a predictive menu performance model which may estimate the selection time of a menu item. It consists of a procedure of three steps, defined as search, decision and pointing step. Each step contributes a set of mathematical equations which may be utilized to calculate the corresponding cost of an item in time unit. In addition, the SDP model has been applied and developed in various menu optimization studies, such as hierarchical menus, circular and square layouts. [6] [27] Because of these, the SDP model appears to be an appropriate candidate regarding menu performance study and is applied as a fundamental theory in this thesis work.

Search is the procedure where users attempt to seek and identify desired items in a menu. The items may be located by utilizing serial search approach and by utilizing the spatial memory of users. These two strategies may be described as the following,

- **Serial search:** is also known as visual search, is a search strategy where users attempt to search the desired item by iterating through the whole menu by a top-to-bottom approach [2] [5] [10]. Additionally, items subsequent to the desired item are not considered. Moreover, the time cost incurred by serial search may continuously decrease as users are increasingly gain experience of the given menu. On the other hand, a long menu length may negatively affect the time cost for serial search strategy as users are more likely to visit more items priori to locating the desired one. [2] [3] [11]
- **Directed search:** is a search strategy where users attempt to locate an item from a cluster of items by the probability of the desired item. This strategy requires users to largely rely on their spatial memory instead of serial search.

As a result, the effect of directed search is rather weak for novice users as they have gained little to no experience yet. However, an expert user may fundamentally rely on directed search strategy by utilizing their spatial memories, hence completely abandon the procedure of serial search [3] [11] [12]. In this thesis work, the **directed search** procedure is also referred to as the **decision phase.**

Some researchers have argued that the goals of a user may be multiple when searching in a menu. [15] That is, the user would like to access various items or any of the items in one search procedure. However, in this thesis study it is assumed that the goal of user consists of one item at a time.

Decision, the decision (directed search) time of an item is estimated by the Hick Hyman law which is discussed in the following section. [2] [14]

Pointing, also known as target acquisition, is referred to as the phase initiated subsequent to locating the desired item. This thesis project assumes that the starting location for a pointing phase is from the left top corner of the user interface. The time cost of the pointing phase is calculated with the Fitts' law which is discussed in the following section. [2] [11] [26] [27]

3.3 Mathematical formulation

This section discusses several fundamental equations regarding the actual realization of the SDP model as well as other relevant mathematical parameters, including search time, decision time, pointing time, item probability, user expertise, cost function as well as item semantic function.

3.3.1 The Search-Decision-Pointing model

The Search-Decision-Pointing (SDP) model [2] is utilized to predict the selection time of an item. If i is the desired item, then the selection time for item i may be illustrated as the following,

$$T_i = T_{dsi} + T_{pi}$$

Where T_{dsi} and T_{pi} are referred to as the search & decision time and pointing time of item i respectively.

Pointing time: As previously discussed, pointing is initiated subsequent to locating of a desired item. Hence, the pointing time may be calculated by the Fitts' law:

$$T_{pi} = a_{pi} + b_{pi} \cdot log_2(1 + ID), \quad ID = \frac{A}{W}$$

Where ID is defined as the Index of Difficulty. The term A is referred to as the amplitude of movement from cursor to the desired item, and W represents the width of the item. Additionally, both a_{pi} and b_{pi} are empirically defined constants. [2] [11] [27]

Search & Decision time [2] [11] [12] [13]**:** Novice users frequently attempt to locate a desired item by utilizing the serial search strategy. On the other hand, expert users are considerably more familiar with the menu interface, hence, they may rather rely on spatial memory locations. As a result, it is suggested that the decision and search time may be understood as the sum of serial search (visual search) and Hick-Hyman decisions. Hence, the decision and search time may be defined as the following,

$$T_{dsi} = T_{ssi} + T_{hhi}$$

Where T_{ssi} is determined to be serial search time and T_{hhi} Hick-Hyman decision time. As previously discussed, it is assumed that novice users initiate item searching with a top-to-bottom search strategy. In addition to that, the search process is immediately terminated subsequent to locating the desired item. Hence, the serial search time involves the total number of items n in a menu. That is,

$$T_{ssi} = a_{ssi} + b_{ssi} \cdot n$$

Where both a_{ssi} and b_{ssi} are empirically defined constants. On the other hand, when users have acquired adequate knowledge regarding the positioning of items in a given menu, the search time transforms from linear to logarithmic. As a result, decision time may be presented by the Hick Hyman decision model [14],

$$T_{hhi} = a_{hhi} + b_{hhi} \cdot H_i$$

Where a_{hhi} and b_{hhi} are empirically defined Hick Hyman decision constants. Moreover, H_i represents the entropy of item i as,

$$H_i = log_2 \cdot ({}^{1}/_{p_i})$$

Where p_i is referred to as the probability of a given item.

Item probability: This thesis assumes that the probabilities of items are initially equal. That is, it may be calculated by dividing one with the number of items in a given menu, such as

$$p_i = {}^{1}/_{n} \quad \text{and} \quad \forall_i$$

However, the probabilities of items may repeatedly vary to resonate with the number of selections generated by users. Hence, the probabilities of items are re-calculated subsequent to each menu layout modification,

$$p_i = {}^{t_i}/_{t} \text{ , for every n items selected}$$

Where $t_i$and t are referred to as number of selections of item *i* and total number of selections respectively at the moment of probability recalculation.

User expertise [2]: The user expertise term e_i is utilized in order to balance the weight of search & decision item between serial search and Hick Hyman decision model. The user expertise term may acquire value range from 0 to 1. A user with experience close to 1 indicates that such user is an expert concerning the positioning of items. In such circumstances, the search & decision time for an item is largely dominated by Hick Hyman decision model. Hence, the original search equation may be redefined as the following,

$$T_{dsi} = (1 - e_i) \cdot T_{ssi} + e_i \cdot T_{hhi}$$

Where the user expertise e_i may be modelled as the following,

$$e_i = L \cdot (1 - {}^{1}/_{t_i})$$

Where t_i is referred to as the number of selections of item *i* while L is determined as the learnability of a menu.

Menu Learnability [2]: The menu learnability L is referred to as a parameter which indicates the degree of modification. By default configuration, it is calculated by one minus movements incurred by switching the positions of items *i* and *j* divided by the menu size, it is expressed as the following,

$$L = 1 - (\frac{\Delta M_{ij}}{Menusize})$$

Where **movement** M_{ij} is calculated as the absolute value of position of item i minus position of item j, that is,

$$\Delta M_{ij} = \left| Position_i - Position_{ij} \right| \quad \text{where}$$

i = 1, 2…, menu size, j = 1, 2…, menu size, and i != j

By default, the *Background Simulator* calculates the value of menu learnability priori to each modification attempt. On the other hand, the *Graphical Simulator* also calculates the corresponding menu learnability for each method it executes. The *Background Simulator* and *Graphical Simulator* are discussed in detail in the next chapter. [29]

Cost function: The cost function is also referred to as the penalty function which is utilized to estimate the cost regarding altering the positioning of menu items. More specifically, subsequent to repositioning a menu item, user loses a part of knowledge regarding the positioning of that particular item. Because of this, the number of selections of that item shall decrease according to the amount of movements it has taken times a cost factor. That is,

$$C_i = \Delta M_{ij} \cdot CF_i$$

Initially, the cost factor CF_i is calculated as one divided by total number of items. That is,

$$CF_i = {}^{1}/_{n}$$

This value reflects the fact that an item loses approximately half of its selections if it has taken a movement equal to half of the menu length. Similarly, if an item originally located as the first item in the menu, has been repositioned to the end of menu (i.e. $\Delta M_{ij} = menu\ size - 1$), that particular item may lose most of its previous selections. As previously mentioned, item probabilities are recalculated after each successful menu optimization, hence, the cost function is executed prior to this procedure in order to reflect the cost incurred for modifying menu items. [29]

3.3.2 Semantic function

The semantic function is calculated based on predefined semantic table regarding the semantic relations between menu items. In contrast to the SDP model, the semantic function is defined as a reward function [4] [27] which encourages the grouping of semantically related items together in order to achieve high menu performance. The semantic value of a menu may be expressed as the following,

$$S = \sum_{i,j} R_{ij}$$

$$i = 1,2, \dots n - 1 \quad and \quad j = i + 1$$

Where R_{ij} is the semantic value between item *i* and *j* defined in the semantic table.

The semantic function is implemented together with the SDP model to form a weighted menu performance model, which is defined as the objective function in this thesis work.

3.4 Objective function

As mentioned above, the objective function is composed of the SDP model and the semantic function. On the other hand, these two models also contribute their own objective functions. That is, the objective function of the SDP model is illustrated as the following,

$$minimize \quad T_{avg} = \sum_{i}^{n} T_i p_i$$

Where T_{avg} is referred to as the average item selection time of the whole menu and p_i is the probability of menu item *i.* As noticed, the objective function of the SDP model is defined as a *cost function* [4] [11] [18] [23], hence, a better menu performance may be achieved by minimizing the average item selection predicted by the SDP model.

As previously mentioned, in contrast to the SDP model, the semantic function is defined as a *reward function* [4] [27], hence, the goal is to maximize the value of semantic model, as illustrated as the following,

$$maximize \quad S = \sum_{i,j} R_{ij}$$

As noticed, the overall objective function consists of two very distinctive objective functions. Because of this, a weight parameter is required to balance and adjust the optimization focus depending on situation needs. As a result, the overall objective function may be defined as a multi-objective optimization function [4] [8] [11] [18] [29], which is illustrated as the following,

$$PI = W_{sdp}T_{avg} + W_sS$$

Where PI is defined as the performance index for the joint objective function while W_{sdp} and W_s are referred to as the performance weights for the SDP model and the semantic model respectively. This objective function appears to be proper, however, a couple of key points are noticed. First, the values produced by these two models are in different domains. That is, the value calculated by the SDP model is in time domain while the value of semantic model is in a generalized domain (i.e. between 0 to 1). Second, the results these two models are on the opposite vectors as the goal is to maximizing the semantic relations between items while minimizing the average selection time predicted by the SDP model. Because of these, the performance models have to be modified. In order to achieve this, result of the SDP model is normalized in order to provide values between the same range as the semantic model. Additionally, instead of minimizing the selection time, the multi-objective function attempts to *maximize the reversed and normalized selection time* of the SDP model. As a result, this would allow the SDP model to perform on the same vector as the semantic model by retaining the original objective. That is, a lower selection time results in higher menu performance. The final form of the multi-objective optimization function is illustrated as the following,

$$maximize \quad PI = (1 - W_{sdp}T'_{avg}) + W_sS$$

Where T'_{avg} is referred to as the normalized average item selection time.

Chapter summary

This chapter has discussed the fundamental concepts of this thesis work, including menu performance models, objective functions and menu performance factors. A variety of menu performance factors have been presented, including menu length, item positioning, user expertise, menu learnability, pointing strategy as well as semantic relations between items. Based on these menu performance factors, this thesis extends the SDP model by introducing two additional factors known as the cost factor and the semantic parameter. The cost factor is applied by a cost function which reduces item selections based on the movement incurred for optimizing menu layouts. On the other hand, the semantic parameter is modelled by a semantic function which encourages the grouping of items with close semantic relations. Based on the extended SDP model and the semantic model, a joint multi-objective optimization function is formed. This objective function serves as the fundamental framework centered on which all the optimization methods are built. The design and operation principles of these optimization methods as well as the programmed simulators are discussed in the following chapter.

Chapter 4. Software and optimization design

This chapter discusses the *Background simulator and Graphical* simulator programmed in this work. Furthermore, this chapter presents the design and operation principles of optimization methods utilized in chapter 5. Both simulators are programmed the Java programming language JDK version 1.8. Additionally, The Graphical simulator is based on the Swing library in JDK. [25]

4.1 Background simulator

The *Background simulator* is a background process which executes a series of actions, including menu generation, probability table generation, semantic table calculation and real user simulation. Additionally, menus are processed by the *Background simulator* prior to be modified by the *Graphical simulator*.

The *Background simulator* consists of a variety of functionalities, its operation principles may be demonstrated as the following,

The initial step of a simulation execution is to configure a viable set of parameters for the *Background simulator*, including simulation duration, optimization frequency, skew of Zipfian distribution, cost factor, weight of probability models and menu size. Some parameters are utilized in actual optimization process while others are applied as predefined variables in early simulation phases. For instance, probability table are generated based on the Zipfian distribution. This is because, Cockburn et.al have argued that menu items are selected according to a non-uniform manner [2]. In fact, other researches have also indicated that the distribution of item selection in a menu follows the Zipfian distribution [13] [21], where a large proportion of the probability cluster is dominated by only few items. As a result, other low frequency items are left unselected which create a long tail effect [28]. On the other hand, menu size which is also referred to as menu length in chapter 3, defines the total number of items reside in the menu. The default menu size in the *Background simulator* is 26, however, the *Background simulator* fortunately supports a very large menu size, theoretically ranges from 1 to any finite number. For instance, a menu size of 1000 is still considered to be feasible in the *Background simulator*. In addition to this, if available, the *Background simulator* reads a semantic table based on the theme package configured in miscellaneous settings. If a semantic table is unavailable, the *Background simulator* skips this process and ignores the influence of semantic model in optimization phases.

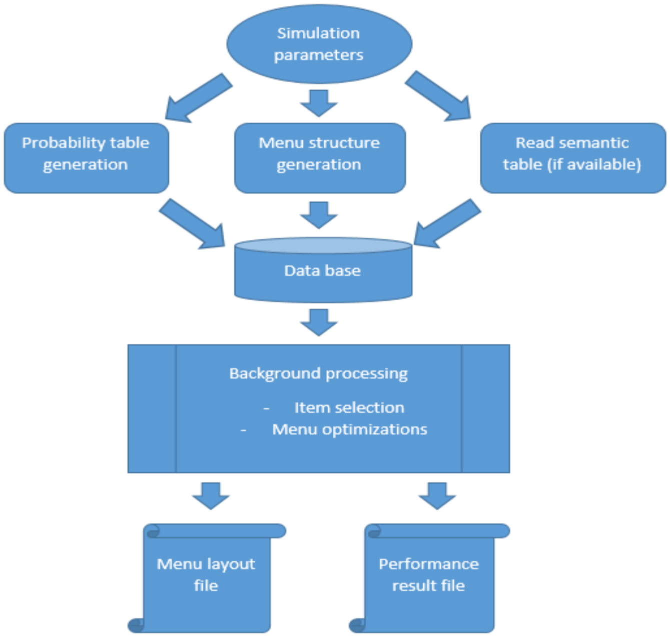


Figure 4: Background simulator operation process

The semantic table, menu layout and generated probability table construct a data base which is processed in actual optimization phases. Each test case holds its own data base which is utilized in all optimization methods. This is particularly important because the data base allows different optimization methods to be executed with their own configurations, such as a higher optimization frequency or more expensive cost factor.

The simulation begins by extracting data from the data base. Item selections are being generated over time and menu layout is continuously optimized based on an *exhaustive search algorithm* discussed later. This algorithm is implemented as method one in later sections. The *Background Simulator* performs this process until the desired number of selections have been generated. Subsequent to this, the *Background Simulator* produces two files, i.e. a menu layout file and a performance result file. The performance result file consists of performance indexes of successful optimization attempts calculated during simulation runs. The menu layout file is composed of menu item related information, such as item position, item selection, simulator calculated probability table and actual user selection probability. It is worth noticing that items are selected based on the user selection probability, which is the probability table generated based on the Zipfian distribution in parameter configuration phase. On the other hand, the *Background Simulator* calculates a real-time probability table for optimization models. This probability table is continuously being modified to reflect changes being made regarding item selections during runtime. The user probability table stays static during the whole simulation.

4.2 Graphical interface

This section discusses the operation principles of the *Graphical Simulator*. The *Graphical Simulator* is initialized by extracting data from menu layout file and performance result file delivered by the *Background Simulator*. Subsequently, the *Graphical Simulator* constructs a graphical representation of a menu based on its original menu layout prior to any optimization attempts and the menu theme package. This menu layout is referred to as the basic layout.

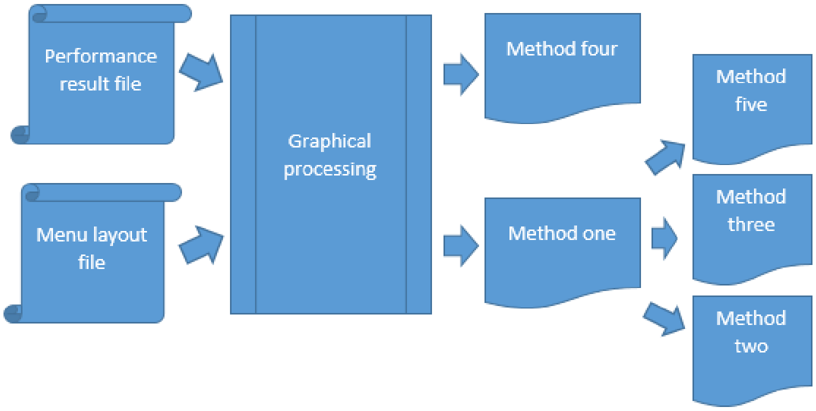


Figure 5: Graphical simulator operation process

Based on the basic layout, the Graphical Simulator attempts to further optimize the menu layout by applying several graphical optimization method. These methods are referred to as method two, three, four and five. Each method implements at least one of the optimization strategies discussed in the following section. However, due to the supplementary modifications, menu layout produced by these methods may potentially suffer from infeasibility design as well as additional menu learnability losses.

4.3 Optimization method design

This section describes the design principles and operation processes of each optimization method. In addition to this, menu optimization and adaptation strategies are also presented.

4.3.1 Optimization strategy

Bailly et.al have proposed a structure of taxonomy which attempts to categorize menu characteristics based on three dimensions, including menu item, menu and menu system. These dimensions may be presented as a hierarchy where menu system is composed of menus and menu consists of menu items. Bailly et.al have stated that this hierarch allows menu modifications to be performed on different levels. As a result, the flexibility of optimization method design is greatly enhanced. For instance, a designer may want to improve menu performance on partial level without altering the overall structure of the whole menu and menu system, e.g. by changing the characteristics of a menu item. On the other hand, the entire menu system may be optimized if desired, e.g. when designing user interface for new applications. [13]

Centered on the concept mentioned and the work of other authors [19], menu performance may be increased by focusing on a variety of strategies. In this thesis work, the following strategies are utilized,

- **Increase item size:** Analysis of the Fitts' law suggests that menu performance may be increased by enlarging the size of item. Additionally, an item with large size is more likely to attract user attention, hence also increases item recognition. [2]
- **Reducing pointing distance:** Based on theory of the Fitts' law. Menu performance may be increased by decreasing the amplitude of movement required for pointing to an item [26]. Moreover, this optimization strategy may also decrease the time needed to search for an item for inexperienced users because they tend to utilize the serial search strategy [1] [13].
- **Increasing saliency:** Menu performance may be optimized by enhance the saliency of the most significant items [2]. An item's saliency may be increased by enlarging its size. In addition to this, supplementary graphical components, such as item transparency and separator lines may also be utilized to increase item saliency.
- **Semantic grouping:** The semantic characteristics of items may greatly affect menu performance. Because of this, items may be grouped according to their logical relationships, thus increasing menu performance. [30]

4.3.2 Adaptation strategy

McGrenere et.al have conducted researches regarding the comparison between different menu variabilities, including adaptive, adaptable and static menu [21]. Based on their researches, these menu adaptations maybe explained as the following,

A **static** menu is defined as a menu which does not alter its layout during utilization. On the other hand, a **dynamic** menu changes its layout during usage. Additionally, **dynamic** menus may be further divided into **adaptive** menus and **adaptable** menus. An **adaptive** menu is a menu where layout modifications are controlled by systems, such as optimization method or algorithm. On the other hand, in an **adaptable** menu, the layout changes are initiated by users. [21]

Based on the design and operation processes of the *Background simulator* and *Graphical simulator*, method one and four are executed centered on the theory of the **adaptive** menu whereas method two, three and five follow the concept of the **adaptable** menu. All optimization methods are evaluated according to the objective function discussed in previous chapter.

4.3.3 Method design

Method one

The method one is executed by the *Background Simulator* and its operation is based on the *exhaustive search algorithm*. The *exhaustive search algorithm* is a search algorithm which iterates through all possible solution candidates. Each candidate is verified by the objective function. In this thesis work, the method one applies *exhaustive search algorithm* [11] [13] [23] in such way that it attempts to form new menu layouts by considering all the possible swaps between items. Each swap creates a new layout and its corresponding menu performance is calculated according to the objective function discussed in previous chapter. The highest menu performance index is logged to the result file and is used for comparison in the following optimization attempts. The menu layout with the highest performance index estimated by method one serves as basic result and is transferred to method two, three and five for further optimization.

The overall operation process of method one may be demonstrated as the following,

1. Initial parameter definitions, including simulation duration, optimization frequency, skew of Zipfian distribution, cost factor, weight of probability models and menu size.
2. While elapsed simulation time is less than target simulation duration, perform the following on each execution loop,
    - Generate item selection according to predefined user probability table
    - Based on optimization frequency, perform *exhaustive search algorithm* to identify new menu layouts with superior performance index than the current menu in usage.

- If better menu layouts are determined, log information regarding the menu with the best performance index of the current execution loop to the result file, including the number of optimization attempt and its performance index.
- Subsequently, this menu layout is presented to user

3. Continue the execution loop from step 2

Method two

Method two attempts to improve menu performance by utilizing the reducing pointing distance strategy. As a result, method two may shorten both item search and pointing time. As discussed in previous chapter, performance of menu is modelled as the following,

$$T_i = T_{dsi} + T_{pi}$$

Where T_{dsi}represents search and decision time while T_{pi}represents item pointing time which is governed by the Fitts' law,

$$T_{pi} = a_{pi} + b_{pi} \cdot log_2(1 + ID), \quad ID = \frac{A}{W}$$

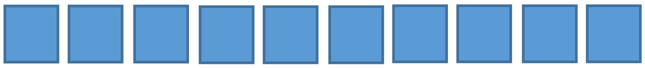
Where *ID* is referred to as the *Index of Difficulty* and A as amplitude of movement and W item width. The *ID* may be reduced by two strategies, i.e. either to reduce the amplitude of movement or increase the size of item. Method two utilizes the first strategy. That is, to reduce the pointing distance required to click an item. Method two realizes this strategy by dividing the menu into two parts and relocating the second part of menu to the beginning of menu layout on a parallel row with the first part. As a result, the pointing time of relocated items are greatly reduced, hence increases menu performance. Furthermore, novice users tend to search desired items by utilizing the serial search strategy. Because of this, by separating menu into two rows may potentially decrease the time incurred for locating an item, thus reduce average selection time. However, method two is limited by the following design constraints,

1. The second part of menu (i.e. relocated items) does not violate the overall user interface. That is, a sufficient amount of space must be available for relocating items in order to prevent overlapping.
2. The outcome (i.e. menu layout) is feasible to users regarding usability. That is, the final layout shall hold balanced rows where each row contains an equal or close to equal number of buttons.

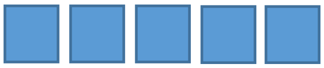
A new design variable is implemented in method two, it is referred to as the *separator*. The *separator* is utilized by method two in deciding the appropriate position from which the menu is divided into parts. More specifically, the *separator* attempts to separate menu layout by balancing the amount of selections of items reside on each row. The *separator* is calculated and applied as the following,

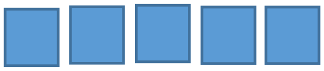
1. Configure initial *separator* value, the default value is 0.1. That is, the amount of selections of items on the first row is 0.1 of the total amount of selections of all items. Hence, the initial overall ratio of selections between first and second row is 1:9.
2. Configure step value, also referred to as cumulative value, the default value is 0.1. That is, the *separator* is moved (increased) by 0.1 on each step, up to 0.99. Each time the *separator* is moved, the performance index of current menu layout is calculated and compared to a previous (best) one. The menu layout with higher performance index is marked as the best layout for next comparison.
3. Configure maximum separator value, the default is 0.5. That is, the movement of *separator* is terminated upon reaching this value. The menu layout with highest performance index is then presented on the *Graphical simulator*.

As previously mentioned, method two may produce infeasible designs. A feasible design may be illustrated as the following, if the original menu layout is,



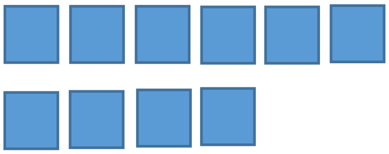
A feasible modified menu layout may be,





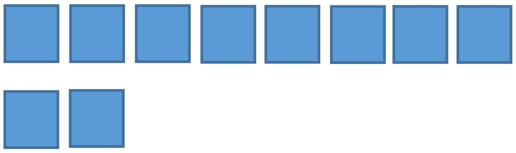
Balanced

Or



Slightly unbalanced

An infeasible menu layout may be,



Unbalanced

These graphs demonstrate an example regarding defining the feasibility of a menu layout, however, the final outcome may also be decided by actual designers.

In fact, the *separator* may be configured according to another convention. That is, the *separator* may be adjusted primarily based on raw button index, e.g. separate menu into two parts from the 12th button index. As a result, it is much easier to achieve a balanced menu layout, however, this may decrease the benefit of method two. This is because, method two attempts to further optimization the result of method one. As previously discussed, method one utilizes *exhaustive search algorithm*, however, due to limited simulation duration, method one may hardly process through every possibilities. As a result, some important items which holds a significant number of selections, are positioned in undesired locations which may potentially decrease menu performance index, such as in the middle of menu. Because of this, method two attempts to identify the locations of these items by evaluating with movements of the *separator* and relocate them on a separate row. Indeed, this may also be achieved by shifting menu layout based on raw item index, however, the efficiency of this approach is greatly reduced for menus with large size.

Method three

Method three attempts to improve menu performance by utilizing several strategies, including increasing item size, reducing pointing distance as well as enhancing saliency. These strategies are implemented as the following,

- Items are grouped into chunks based on the basic result obtained from method one, separator lines are added between chunks to increase saliency between groups [12] [13].
- The first item of each group has its size enlarged for better recognition.
- For each chunk, items of normal size are relocated onto two balanced rows for reducing pointing time.

Based on these strategies, method three essentially creates a new menu layout, hence, the overall menu learnability is relatively low compared to other methods. Because of this, method three shall be considered when the following requirements are satisfied,

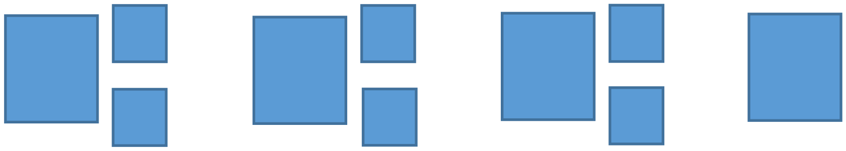
- The menu layout provided by method two is infeasible
- Semantic model holds a relative low performance weight
- Method five offers few improvements without violating its design constraints

Furthermore, method three is limited by similar design constraints as method two. That is, a sufficient space must be available for relocating items on another row and for enlarging the size of items.

The method three is executed as the following,

1. Configure the group size parameter *n*. This value is defined to be 5 in this thesis for testing purpose. A valid value for this parameter may be n = 2, 3, … (menu size – 1).
2. Extract information from the result file produced by method one
3. Separate items into groups based on parameter *n*
4. Enlarge the size of the first item in each group
5. Locate the rest of items onto two rows

A sample result of method three (n = 3) is demonstrated as the following,



Method four

Method four attempts to optimize menu layout by focusing on item semantic relations. Because of this, method four is only applicable if information regarding semantic relations between items are available.

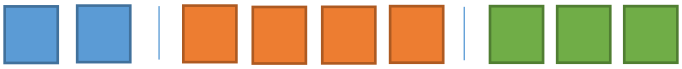
As discussed in previous chapter, if a semantic table is present, the overall menu performance model may be illustrated as the following,

$$P = W_{sdp} * P_{sdp} + W_{se} * P_{se}$$

Where in method four the performance weight of semantic model W_{se} largely dominates the one of the SDP model W_{sdp}.

Similar to method one, the operation of method four is also based on the *exhaustive search algorithm*. Moreover, method four utilizes the same data base of method one and is executed as a parallel process by the *Background Simulator*. Subsequently, the result of method four is transferred to the *Graphical Simulator*. Based on the semantic rating of items, the *Graphical Simulator* attempts to further optimize menu performance by adding separator lines between items to strengthen the saliency between distinctive groups [13].

A sample menu layout of method four may be illustrated as the following,



Where groups are represented by different colors.

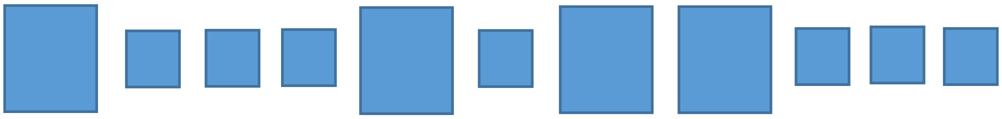
Method five

As previously mentioned, menu performance may be increased by decreasing the *index of difficulty* introduced by the Fitts' law. Hence, method five attempts to improve menu performance by applying the increasing item size strategy. An item is enlarged if the item is considered to be a *significant item*. An item is *significant*, if its number of selections is equal or greater than a percentage of the total number of selections. This percentage parameter is referred to as *acceptance*. In addition to this factor, it is understood that by increasing item size also increases the menu length in pixel, hence, the menu shall not exceed the maximum resolution of the *Graphical Simulator*. Based on the predefined parameter of the *Graphical Simulator*, the maximum number of enlarged item is eight.

Method five is execute as the following,

1. Define the value of *acceptance*, the default value is 0.05. That is, the size of item is enlarged if it holds equal or more than 5% of the total number of selections.
2. Extract data obtained from method one.
3. Identify the most *significant items*.
4. Attempt to enlarge the size of the most *significant items* without violating design constraint

A sample menu layout may be illustrated as the following,



Similar to method two, the objective of method five is to reduce the *index of difficulty* introduced by the Fitts' law. The different is that instead of reduce the amplitude of movement, method five increases the target size. As a result, method five may contribute less improvement compared to method two. However, method five rarely suffers from infeasible menu layout and retains the highest menu learnability of all other methods. Because of this, method five may be considered in most situations.

Chapter summary

This chapter has explained the fundamental operation principle regarding the *Background Simulator* and *Graphical Simulator*. Additionally, several optimization strategies presented by Bailly et.al have been discussed. Moreover, menu adaptation strategies proposed by McGrenere et.al have also been described. Based on these optimization and adaptation strategies, five optimization methods have been designed. For instance, method one is processed by the *Background simulator* as the initial optimization process. Moreover, its result serves as basic result to be further modified by several other methods applied by the *Graphical Simulator*. For instance, method two attempts to improve menu performance by focusing on decreasing movements required to reach an item by relocating items onto another parallel row. Moreover, this action may also potentially reduce the search time incurred by serial search strategy for novice users. However, method two may frequently suffer from infeasible menu designs, thus shall be carefully considered. On the other hand, method five attempts to increase menu performance index by enlarging the size of the most *significant items*. This method may contribute less compared to other methods, however, it holds the highest menu learnability and suffers little from infeasible menu designs. Method three on the other hand, attempts to improve menu performance by utilizing various strategies introduced by Bailly et.al. It may indeed provide great menu performance benefits, however, method three suffers from high menu learnability loss, thus shall only be considered when desired situations are satisfied. In addition to these methods, method four is executed in parallel with method one by using the same data base and algorithm. However, it focuses more on the semantic relationships between items instead of general menu performance. As a result, the semantic meanings are more prominent between items compared to other methods. The performance results of these optimization methods are evaluated by several test cases introduced in the nest chapter.

Chapter 5. Evaluation

This chapter evaluates the performance of optimization methods discussed in Chapter 5. Initially, a variety of essential simulator runtime parameters are presented. Subsequently, the simulation environment, hardware specifications and simulation assumptions are displayed. After which, several simulation test cases are conducted in order to perform the evaluation. Furthermore, a short review is provided in each test case for assessing the feasibility of the optimization methods.

5.1 Simulation parameter

The optimization algorithm supports a variety of distinctive and adjustable factors. Factors belong to the *Background Simulator* are presented as the following,

- **Simulation duration:** limits the duration of simulation by counting the number of items generated. This value is set to be *1000* by default. That is, a total amount of 1000 selections are generated during experiment.
- **Optimization frequency:** defines the rate of execution of optimization attempts. The default value is *1*. That is, the simulator attempts to perform optimizations subsequent to each item selection.
- **Semantical table:** defines whether the semantic table is utilized or not. A vital factor that reveals the effect of semantic meaning regarding menu performance. The default value is *TRUE*;
- **Performance model weight:** determines the weight of influence between the SDP model and semantic table. The default value is *0.5 to 0.5*.
- **Cost factor:** determines the cost incurred by changing the position of menu items. The default value is calculated as one divided by menu size. Note, this parameter requires the movement factor to calculate the final cost of changing the position of an item. The movement factor is discussed in Chapter 3.

- **Skew of Zipfian distribution:** This parameter defines the shape of probability curve generated by Zipfian distribution discussed in **chapter 4.** A large skew value may produce a significant long tail effect where few items possess a high probability rate while other items close to the minimal possible number. The default value of this parameter is set to be *0.8*.

Several other factors are introduced in the *Graphical simulator*, they are displayed as the following,

- **Separator:** determines the position of where the menu is divided into two representation rows. The default value is set to be *0.1*. This parameter is implemented in optimization method II and is valid between *0.1 to 0.5*.
- **Cumulative value:** defines the increment the *Separator* receives during each optimization attempt. This parameter is applied along with *Separator* in method II. Its default value is 0.1.
- **Acceptance:** determines the most significant items for which the size is enlarged. The default value is set to be *0.05*, i.e. items which contributes more than 5% of the total amount of item selections are considered to be *significant items*. This parameter is applied in optimization method V and is valid between *0.01 to 0.99*.

Factors discussed above perform as the most vital parameters in both *Background* and *Graphical simulator*. On the other hand, several other factors may also affect the result of simulations, such as simulator window size and space between buttons. However, these parameters are considered as static parameters thus altering their value contributes little to the study of actual menu performance model. Because of this, these values are preferably defined by user interface designers.

5.2 Simulation environments

Experiments are conducted on a laptop which hold the following specifications,

Operating System	Windows 10 Home 64-bit
System Manufacturer	Hewlett-Packard
System Model	HP ENVY 17 Notebook PC
Processor	Intel® Core™ i7-4710MQ CPU_2.50GHz (8 CPUs)
Memory	12GB

Furthermore, simulators assume that mouse cursor is always moved from the top corner of the screen. Moreover, the simulator is locked at a resolution of 1920 * 1080, i.e. resizing is not enabled. Additionally, the default size of buttons is 50 pixels and space between buttons is 5 pixels. Furthermore, theoretical constants utilized in this thesis are fetched from the previous research conducted by Cockburn et.al, including the empirically determined constants of the Fitts' law, Hick-Hyman law and visual search model [2].

5.3 Experiments

As previously discussed, the simulator offers five methods to optimize menu layouts. Each method is designed to improve the general menu performance while aiming to provide at least one of the following advantages, including short pointing distance, strong semantic relations between items and enhanced item saliency. In order to evaluate the advantages as well as disadvantages of these optimization methods, this section performs several experiments in simulated environments.

5.3.1 Test case I: Theme Office

Office like software are not strange to anyone, especially to engineers. Every engineer has used some kind of office software in his or her careers, such as Microsoft Office® or LibreOffice®. However, not every office software provides users with an optimized user interface. On the other hand, an optimized user interface may greatly enhance the usability of the software as well as the working efficiency of users.

This test case consists of three purposes. First, it aims to evaluate both the efficiency and stability of the *Background Simulator*. This is executed by setting the skew of Zipfian distribution to extremely high (i.e. 0.8). As a result, the generated user probability table shall be dominated by only few items. Second, this test case attempts to identify the influence of cost factor. In addition to this, the third purpose of this test case is to observe the effect of prolonged simulation as well as the effect of infrequent optimization attempts. In order to achieve these purposes, a secondary simulation is executed in parallel with the first one by utilizing the same data base.

The buttons used in test case I are displayed as the following,

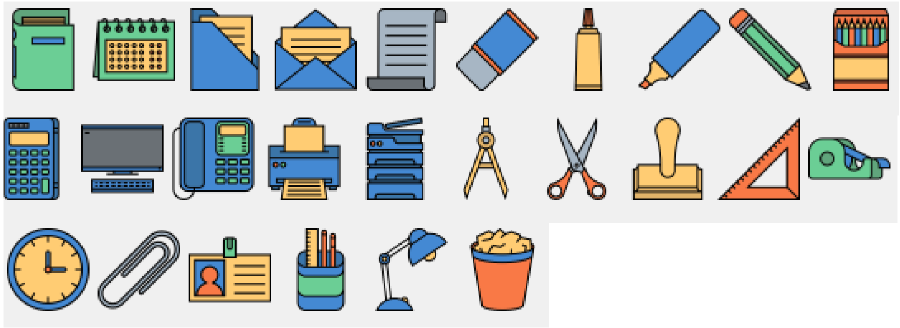


Figure 6: Buttons, test case I

It is obvious that these buttons may be categorized into several groups, such as documents, pencils and digital tools. However, in order to focus on the results of the *Background Simulator*, optimization methods are disabled which heavily rely on the *Graphical Simulator*. Moreover, the influence of semantic table is also ignored. Because of these, the result of method three and method four are neglected in test case I. On the other hand, both test cases

II and III are constructed for both simulators and also offer their own distinctive semantic tables in later sections. Hence, the results of these methods are analysed in those test cases.

As mentioned above, two simulation runs are executed in parallel in test case I by using the same data base. However, their simulation runtime parameters are somewhat different. Thus, two sets of parameter tables are defined, they are expressed as the following,

Simulation duration	1000
Optimization frequency	1
Performance model weight (SDP : Semantic)	1 : 0
Cost factor	0.04
Skew of Zipfian distribution	0.8

Table 1: Simulation parameter, Run 1, test case I

Simulation duration	10000
Optimization frequency	10
Performance model weight (SDP : Semantic)	1 : 0
Cost factor	0.08
Skew of Zipfian distribution	0.8

Table 2: Simulation parameter, Run 2, test case I

Several points are worth noticing from the parameter tables.

The **weight of performance models** is rather irrelevant due to the absence of semantic table in test case I. Hence, objective function solely considers the result of the SDP model.

The **skew of Zipfian distribution** has been adjusted to be 0.8. Because of this, the generated user probability table is dominated by only few items. This reflects the fact that buttons are most likely been selected according to the Zipfian distribution in an office software [2] [21]. The sorted probability table of items are illustrated as the following,

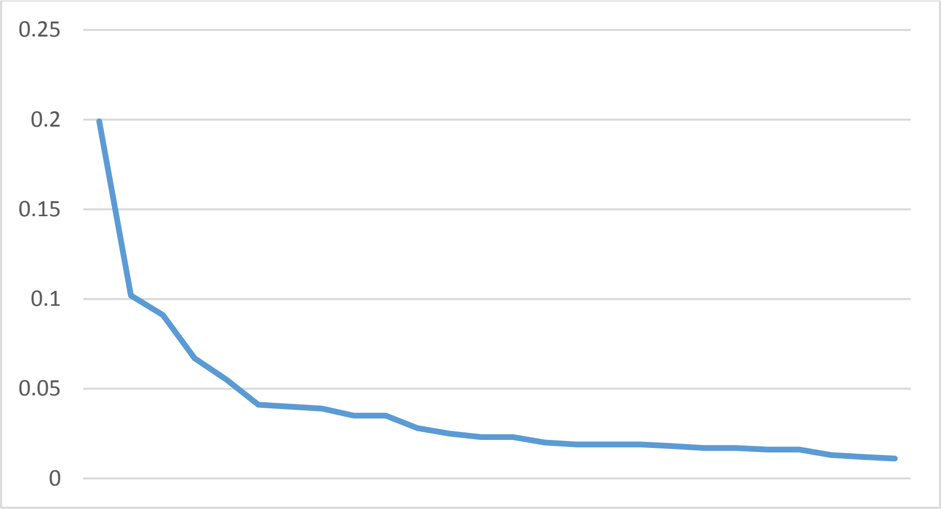


Figure 7: Item probability distribution sorted, test case I

The **simulation duration** and **optimization frequency** are set to be 10000 and 10 respectively in *Run 2*. Because of this, the duration of *Run 2* is much longer but optimization attempts are infrequently executed compared to *Run 1*. With this configuration, users are more likely to learn the menu layout. Thus, it is interesting to observe the potential advantage as well as disadvantage received by several key factors in a prolonged simulation run, such as the user expertise and optimization frequency. Additionally, it is also worth noticing that both runs process the same amount of optimization attempts (i.e. 1000) despite having different simulation durations.

The **cost factor** is 0.08 in *Run 2* which is doubled compared to *Run 1*. Consequently, a double amount of selections is removed from the selection history of the modified button. Hence, this change may further discourage the frequency of successful optimization attempts. An optimization attempt is considered successful if it has improved menu performance by executing optimization methods.

Based on the parameters given in test case 1, the following **hypothesizes** are presumed:

1. *The results of Method one are somewhat similar in both runs*
2. *Method two offers significant improvements compared to Method one in both runs*
3. *Method five provides better result in Run 2 than Run 1.*

Prior to execute test case 1. The button probability distribution and initial menu layouts for both runs are illustrated as the following,

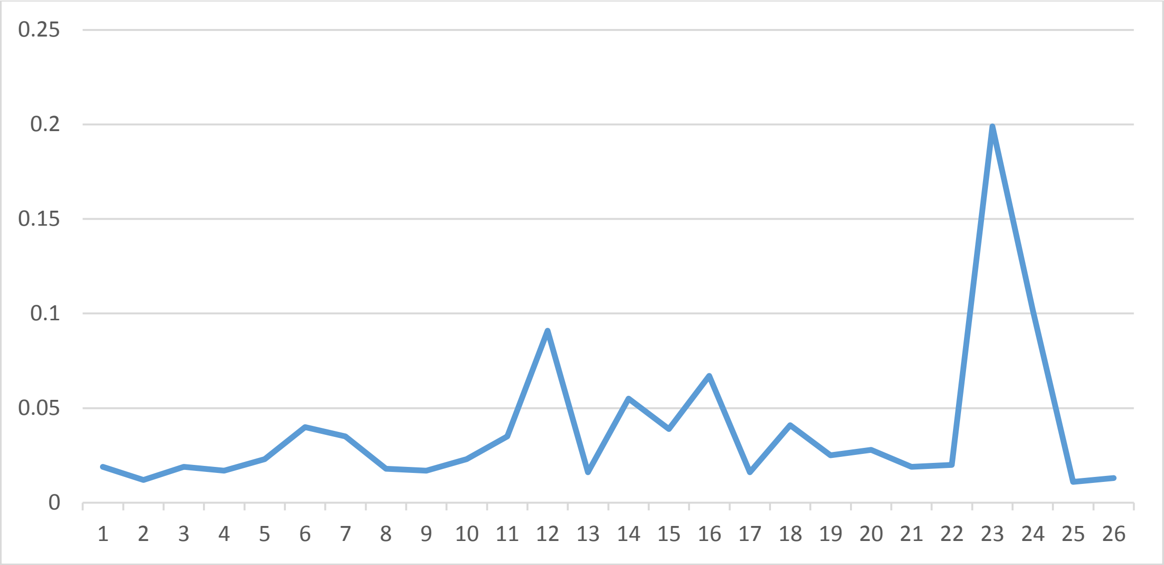


Figure 8: Button probability distribution, test case I

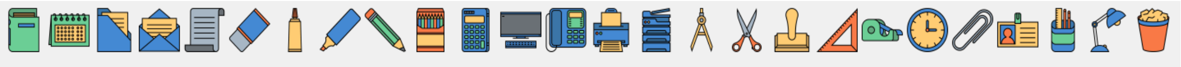


Figure 9: Initial menu layout, test case I

A first observation displays that button *ID-Card* holds the highest probability while button *Writing Tools* holds the second highest. Moreover, buttons are not placed in any sorted order. Hence, no previous optimization has been performed for this menu.

Method one

According to the *Background Simulator*, performance index of the initial menu layout is 0.075 in both runs. Furthermore, successful optimization attempts are displayed as the following,

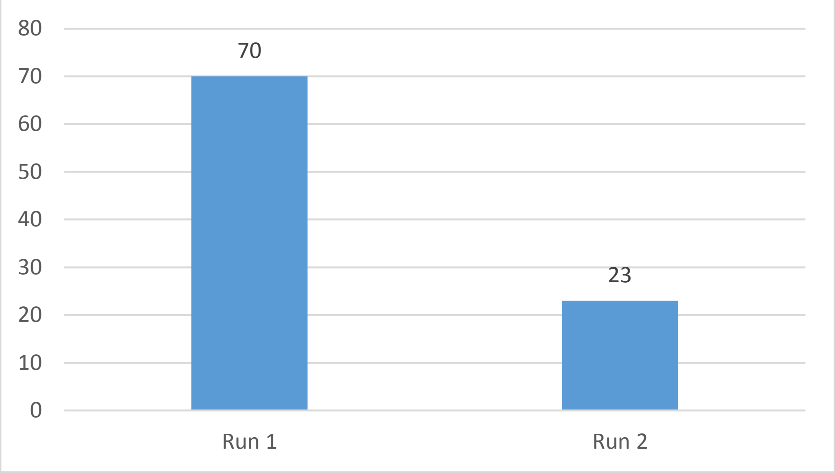


Figure 10: Successful optimization attempts, test case I

It is observed that, despite having both runs be optimized for the same amount of attempts (i.e. 1000), a total amount of 70 optimization attempts have been successful in *Run 1*, however, only 23 attempts have succeed in *Run 2*. This may indicate that optimization attempts are largely affected by both the cost factors as well as optimization frequency. As a result, only few attempts have been successful in *Run 2*. The button selection distribution and the menu layout of *Run 1* is illustrated as the following,

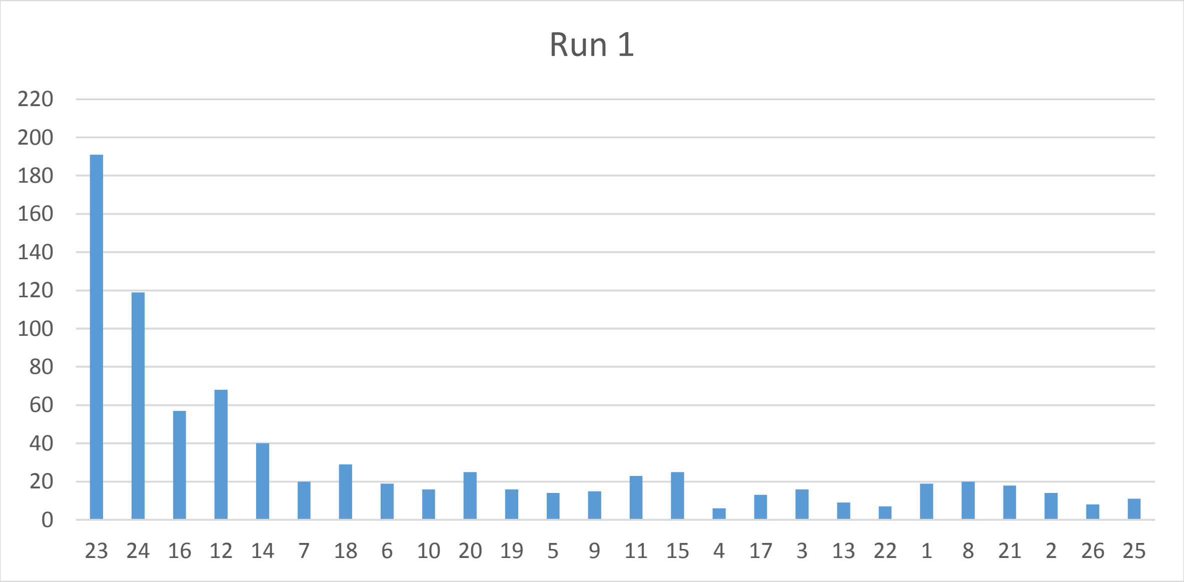


Figure 11: Selection distribution Run 1, test case I

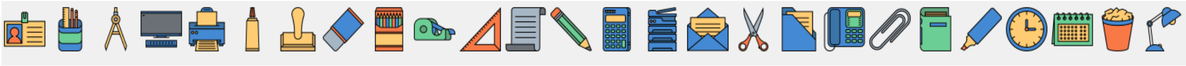


Figure 12: Method one, Run 1, test case I

Based on Figure 11, it is noticed that buttons *ID-Card* and *Writing Tools* which hold the highest amount of selections, have been positioned as the first and second button in menu. Furthermore, other buttons also have their positions modified primarily according to their corresponding number of selections. The *Background Simulator* reports a performance index of 0.927 for *Run 1*.

The button selection distribution and menu layout of *Run 2* are displayed as the following,

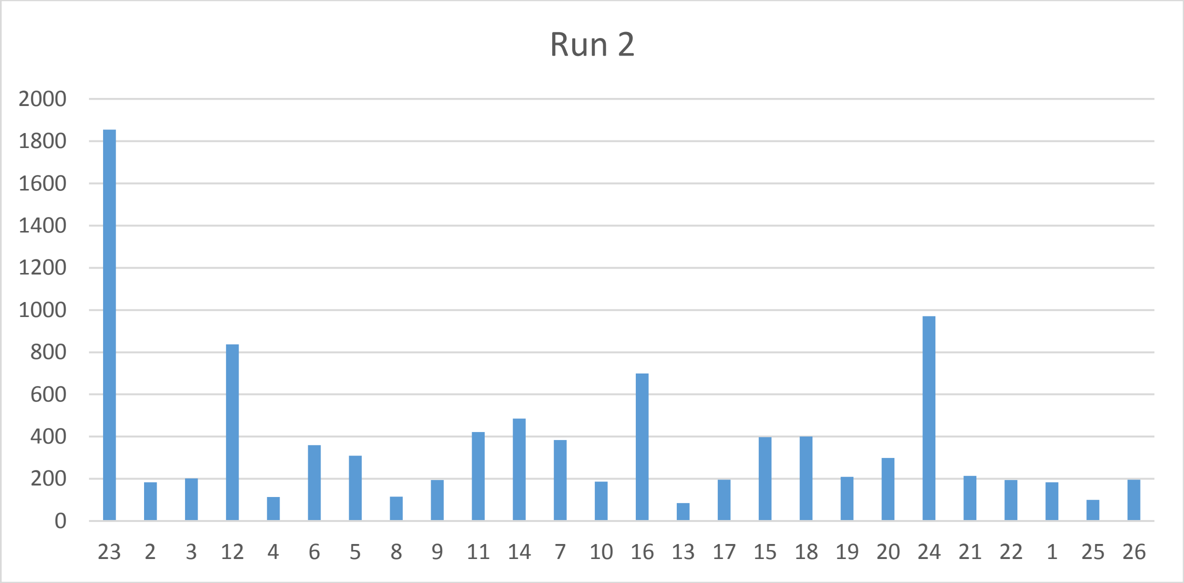


Figure 13: Selection distribution Run 2, test case I

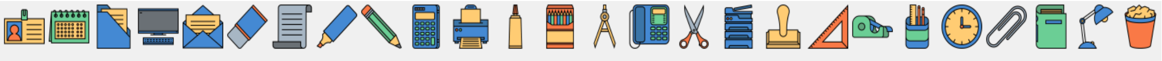


Figure 14: Method one, Run 2, test case I

It is obvious that the menu layout of *Run 2* is rather different compared to *Run 1*. Moreover, fewer buttons have been positioned according to the selection history, such as *ID-Card* or *Computer* . The positioning of other items, such as *Writing Tools* , has been far from optimal. As a result, the menu layout of *Run 2* appears to be rather incomplete. However, according to the *Background Simulator*, *Run 2* has achieved a performance index of 0.907, which is only slightly lower than *Run 1* and with less than half of the successful optimization attempts executed in *Run 1*. Several points exist which may explain the reasons behind this phenomenal and they are related to the following fundamental concepts,

1. **Menu learnability**. As discussed in previous chapters. Menu learnability is a factor which estimates the degree of menu layout modification. That is, a low degree of menu learnability is received if layout of the menu is rapidly being modified. Conversely, a menu yields high menu learnability if its layout remains static. Because of this, the overall menu learnability is higher in *Run 2* compared to the one in *Run 1*.
2. **User expertise.** The user expertise factor is utilized to balance the weight between visual search model and directed search model. Additionally, expert users may completely rely on their spatial memories regarding button positioning instead of visual search tactic. In *Run 2*, simulator user has generated ten times the selections of *Run 1*. Moreover, *Run 2* has executed only 23 menu modifications, i.e. 23 successful optimization attempts. Thus, fewer selections have been removed from history due to cost factor. Because of these, simulated user holds a higher degree of user expertise in *Run 2* than in *Run 1*.

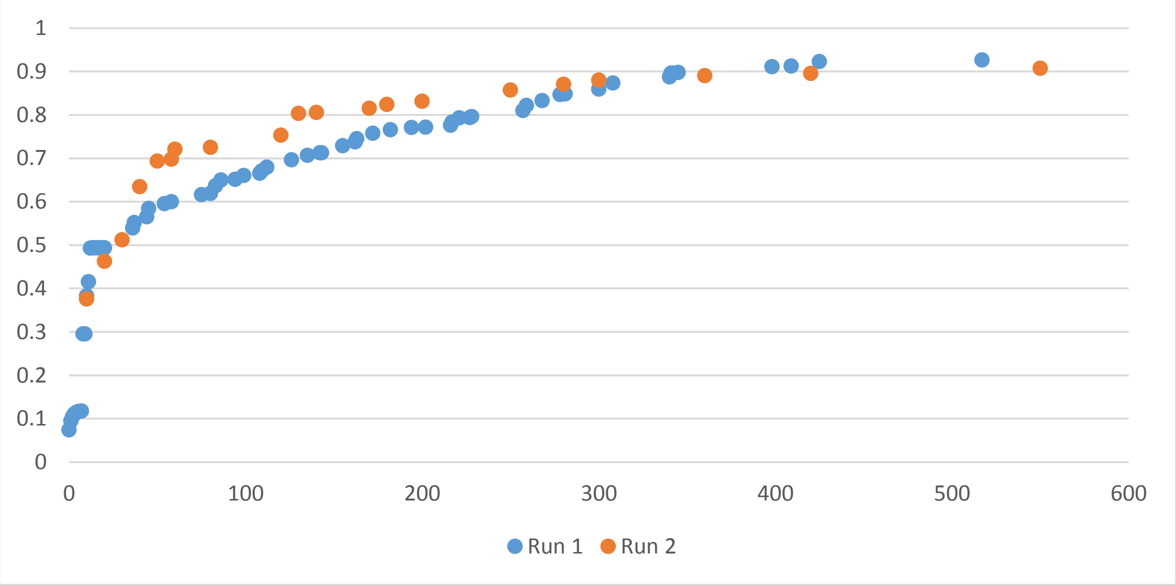


Figure 15: Optimization attempts, test case I

Based on Figure 15 it is noticed that the performance index of *Run 1* is actually lower than *Run 2* in early simulation phases. This is caused by the rapid changes done to menu layout, which results in lower menu learnability as well as user expertise. On the other hand, both runs receive little benefits in later optimization phases as optimization results are becoming closer to optimal, hence menu layouts have been rather static. Consequently, this increases the menu learnability and user expertise of simulated user in *Run 1*, which ultimately results in better performance compared to *Run 2*. However, the *Background Simulator* has executed a finite number of optimizations, thus, the results of both runs are in fact *Approximately Optimal* [4] and results beyond this simulation is unknown. Hence, it is likely that *Run 2* may in fact performs better than *Run 1* in extended simulation runs.

As previously mentioned, the *Graphical Simulator* attempts to further optimize the menu layout generated by the *Background Simulator* with several other methods. However, method three and four are not evaluated in test case I. The analysis for these methods are performed in test case II and III.

Method two

Method two attempts to increase menu performance index by dividing menu into two parts from an appropriate point. As a result, a proportion of menu items are relocated onto a secondary row for quick access. The modified menu layout of *Run 1* is displayed as the following,

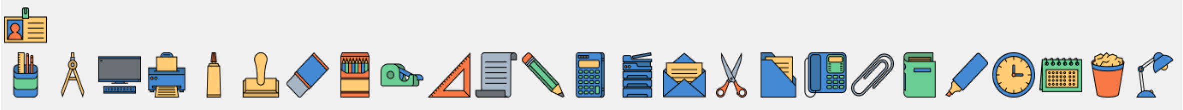


Figure 16: Method two, Run 2, test case I

The *Graphical Simulator* indicates that the performance index of this menu layout is approximately 5.7% better compared to the one in Method one. However, the ratio between two rows is 1:9 which is extremely unbalanced. Because of this, the menu layout constructed by Method two for *Run 1* is infeasible and shall not be considered.

The modified menu layout of *Run 2* is illustrated as the following,

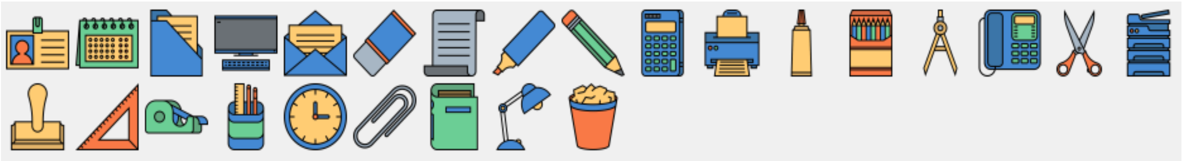


Figure 17: Method two, Run 2, test case I

According to the *Graphical Simulator*, this layout offers a performance index of 0.958 which is 5.6% better compared to Method one. Furthermore, Figure 17 shows the ratio between two rows is 7:3 which is slightly more balanced than in *Run 1*.

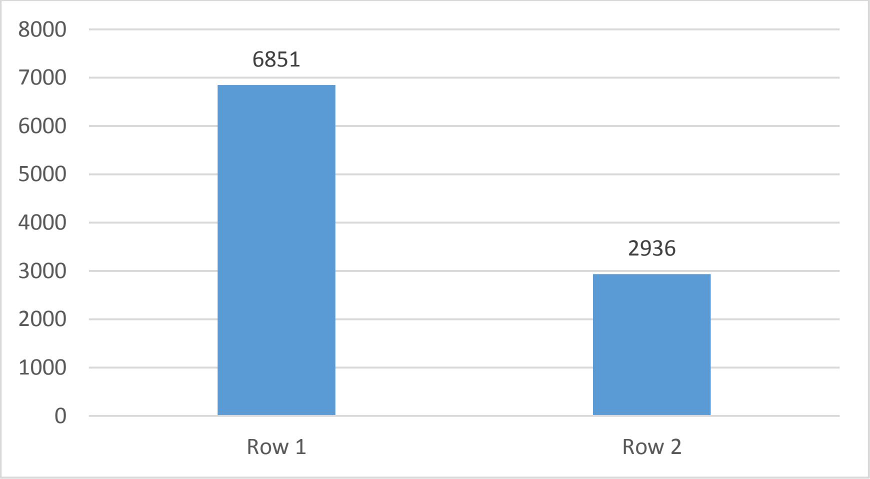


Figure 18: Row ratio, Run 2, test case I

Because of this, this menu layout may be considered.

Method five

Method five attempts to increase menu performance index by enlarging the most *significant buttons* without grouping or modifying the positioning of buttons. The modified menu layout of *Run 1* is displayed as the following,

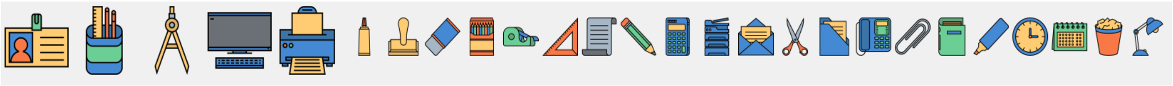


Figure 19: Method five, Run 1, test case I

The *Graphical Simulator* reports a performance index of 0.96 for this menu layout which is improved by 3.5% compared to method one. Additionally, method five allows the size of eight buttons to be enlarge without violating design constraints. Hence, method five may still improve performance index in longer simulations.

The modified menu layout of *Run 2* is illustrated as the following,

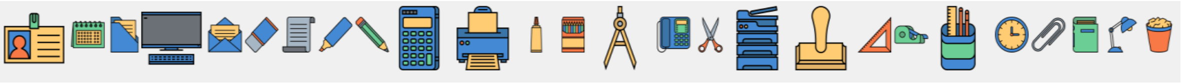


Figure 20: Method five, Run 2, test case I

The *Graphical Simulator* indicates that the performance index of this menu layout is 0.943, which is 4% better compared to method one. Moreover, more buttons have their size enlarged in *Run 2* compared to *Run 1* which results in more performance gain. This indicates method five benefits from long simulation duration as well as infrequent menu layout modifications.

Summary: test case I

This section discusses the hypothesizes presumed in the beginning of test case 1 and conclude results obtained from simulation runs. The overall results of both runs are illustrated as the following,

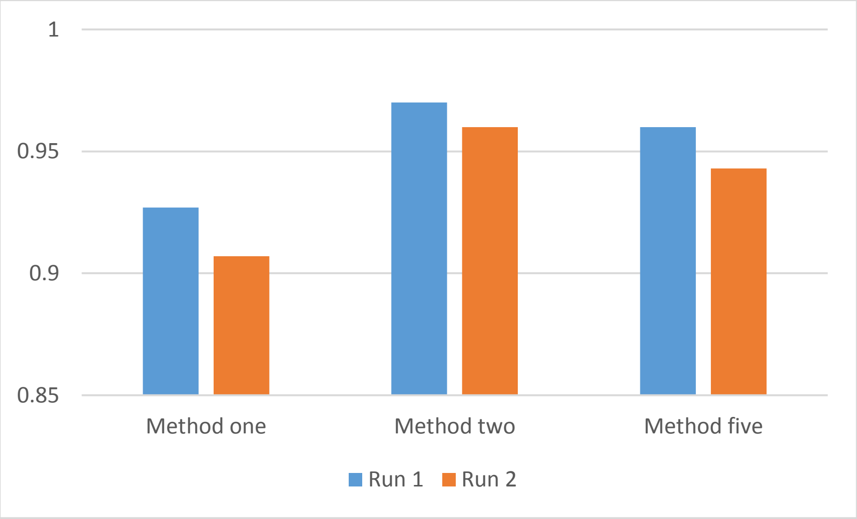


Figure 21: Overall results, test case I

1. The results of method one are somewhat similar in both runs

The menu layouts produced by method one are rather different for both runs. That is, the menu layout of *Run 1* appears to be more optimized compared to *Run 2*. This is caused by the high cost factor as well as infrequent optimization attempts configured in *Run 2*. In addition to this, *Run 2* holds only less than half of the successful optimization attempts performed in *Run 1*, which further proves that modification of menu layout has been greatly discouraged in *Run 2*. On the other hand, the performance index of *Run 2* is only slightly lower than the one of *Run 1* in method one. This indicates that simulated user retains higher grade of user expertise in *Run 2*. Because of these, the menu layouts provided by method one for these two runs are rather different.

2. Method two offers significant improvements compared to Method one in both runs

Indeed, method two are able to offer over 5% improvements in both runs. However, the menu layout of *Run 1* is rather unbalanced, hence it may be considered infeasible in real user cases. On the other hand, the menu layout of *Run 2* appears to be much more balanced, thus, the menu may be usable in real user cases.

3. Method five provides better result in Run 2 than Run 1

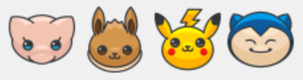
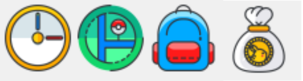
Method five provides no better result for *Run 2* than *Run 1,* however, method five is able to provide better improvements for *Run 2 (4%)* compared to *Run 1 (3.5%)*, however, *Run 1* still holds a better performance index, which is approximately 1.7% higher than *Run 2*. On the other hand, method five has decreased the overall performance gap between *Run 2 and 1* by around 0.5%, which indicates that method five may offer more benefits for prolonged simulation runs.

Due to decent improvements and high menu learnability, the result of method five may be considered as the final design for both runs in test case I.

5.3.2 Test case II: Theme Pokémon

Pokémon game has been a childhood dream for many of us, in fact, some of us still actively play it every day. Pokémon is designed and published by Nintendo® back in the 90s. It is a video game crafted around a kind of fictional companions called Pokémon and players perform as the role of trainers who catch, live and train with Pokémon for great achievements.

This test case consists of two purposes. Firstly, it is designed to observe and analysis the effect caused by decreasing the degree of skew of Zipfian distribution. Because of this, the user probability table is generated in such way that the distribution of item selections is slightly more even compared to the one in test case I. Secondly, this test case aims to validate the feasibility of optimization method four which focuses on the strategy of semantic grouping of items discussed in **Chapter 4**. In order to perform this, the buttons of user interface are categorized into 6 + 1 types. Moreover, Bailly et al. have argued that a logical group averagely contains 4 items [12], hence, each type is composed of four buttons belonging to the same category with the exception of the *Trainer* category, which only has two buttons, i.e. a pair of male and female trainer. The buttons and their classification are displayed as the following,

Pokémon	
Poke ball	
Medicines	
Battle badges	
Poke eggs	
Tools	

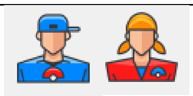
Trainers	

Figure 22: Buttons and categories, test case II

Centered on the classification, the relationships between buttons are described by different grades. Each grade is defined by a number range from 0 to 1. Two buttons possess poor relations if the grade between them is close to 0. Conversely, a grade close to 1 indicates a strong relation between two buttons, such as *Pikachu* and *Snorlax*. Furthermore, each button is strongly associated with other buttons in its own category, however they behave naturally, weakly or poorly with other categories.

For the sake of clarify, this test case defines four relationship grades, including strong (S = 0.8), natural (N = 0.4), weak (W = 0.2), and poor (P = 0.1).

In addition to this, the trainer category has a special grade (T = 0.5) which is slightly better than the natural grade and is also constant towards all other categories. This is done to reflect the fact that trainers are in the center of other categories and all other items are connected via trainers.

The semantic table may be demonstrated as the following,

	Pokémon	Poke ball	Medicine	Tools	Badges	Poke egg	Trainers
Pokémon	**S**	**N**	**W**	**P**	**P**	**N**	**T**
Poke ball	**N**	**S**	**P**	**P**	**P**	**P**	**T**
Medicine	**W**	**P**	**S**	**P**	**P**	**P**	**T**
Tools	**P**	**P**	**P**	**S**	**P**	**P**	**T**
Badges	**P**	**P**	**P**	**P**	**S**	**P**	**T**
Poke egg	**N**	**P**	**P**	**P**	**P**	**S**	**T**
Trainers	**T**	**T**	**T**	**T**	**T**	**T**	**T**

Table 3: Semantical table, test case II

In test case 2, simulator runtime parameters are defined as the following,

Total simulated selections	1000
Optimization frequency	1
performance model weight (SDP : Semantic)	8:2
Cost factor	0.04
Skew of Zipfian distribution	0.6

Table 4: Simulation parameters, test case II

As previously discussed, test case II aims to evaluate the influence of the skew of Zipfian distribution. Hence, the *Background Simulator* attempts to generate a more evenly distributed selection of items by reducing the value of skew to 0.6 from 0.8 of test case I. The sorted probabilities of items generated in test case II are illustrated as the following,

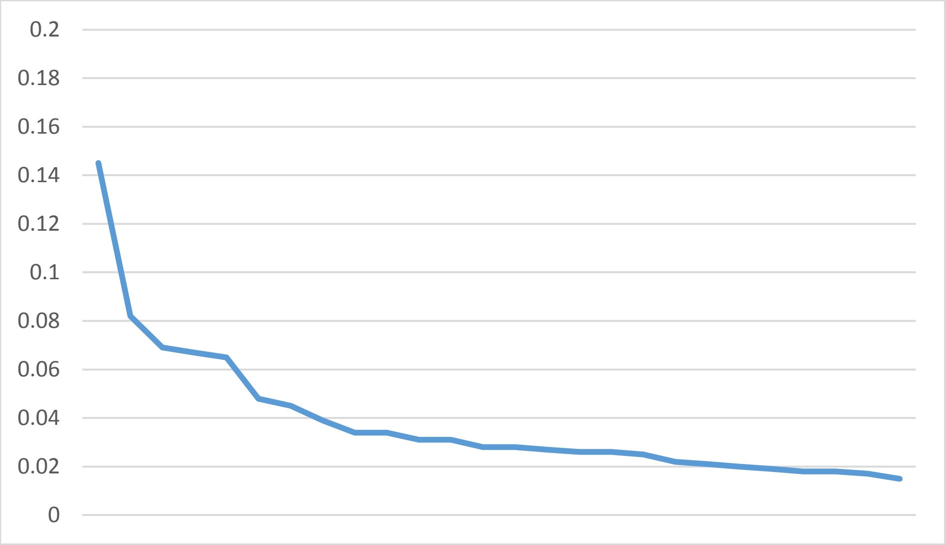


Figure 23:Item probability distribution sorted, test case II

It is noticed that the generated probability table appears to be less dominated by few items, instead, several items have obtained a decent amount of probabilities. Furthermore, the highest probability is perceived to be 0.145 which is significantly lower compared to the one in test case I.

The **weight of performance models** is chosen to be 8:2 in optimization methods one, two, three and five. That is, in these methods, the performance weight for the SDP model is 0.8 while the performance weight for semantic model is 0.2. On the other hand, in method four, the performance weight for semantic model is 0.8 while the SDP model holds a performance weight of 0.2. The weights of models are chosen in this convention to facilitate the performance models to their extreme points. Because of this, test case II may more appropriately fulfill its original purposes.

The **simulation duration** and **optimization frequency** are adjusted to their default values which are 1000 and 1 respectively. As previously mentioned, one of the objectives of test case II is to evaluate the effect of semantic table compared to menu performance model. Hence, a longer duration of simulation run may yield incorrect result. This is because, the semantic model receives little benefit subsequent to a point of optimization attempt due to predefined and fixed value of the semantic table. However, the SDP model constantly gains improvements as more items are being selected due to the learning effect. In addition to this, the SDP model may continuously be optimized for a tremendous amount of times as a menu with 26 items may be ordered in 26! different ways. The *Background Simulator* may still be able to process this, however, the hardware utilized in this thesis is incapable to compute such large number of simulation runs. Hence, the values of **simulation duration** and **optimization frequency** are selected in this convention in order to reduce the likelihood of inappropriate result as well as balance the variance of performance models.

The **cost factor** is reversed to its default value which is 0.04. This adjustment may encourage optimization methods to more frequently execute optimization attempts, e.g. change the position of items. Similar to *Run 1* in test case I. The purpose of this change is twofold. First, it also attempts to reduce the negative effect caused by long simulation duration. Second, it causes the *Background Simulator* to produce more optimization results which enhance the accuracy of experiments.

Based on the purpose and simulation parameters of test case 2, the following **hypothesizes** are presumed:

1. *Method two offers a significant improvement compared to Method one*
2. *The performance of Method three is considerably worse than Method four*
3. *Method five performs better than Method three*

Prior to execute test case II. The initial menu layout and item probability distribution are illustrated as the following,

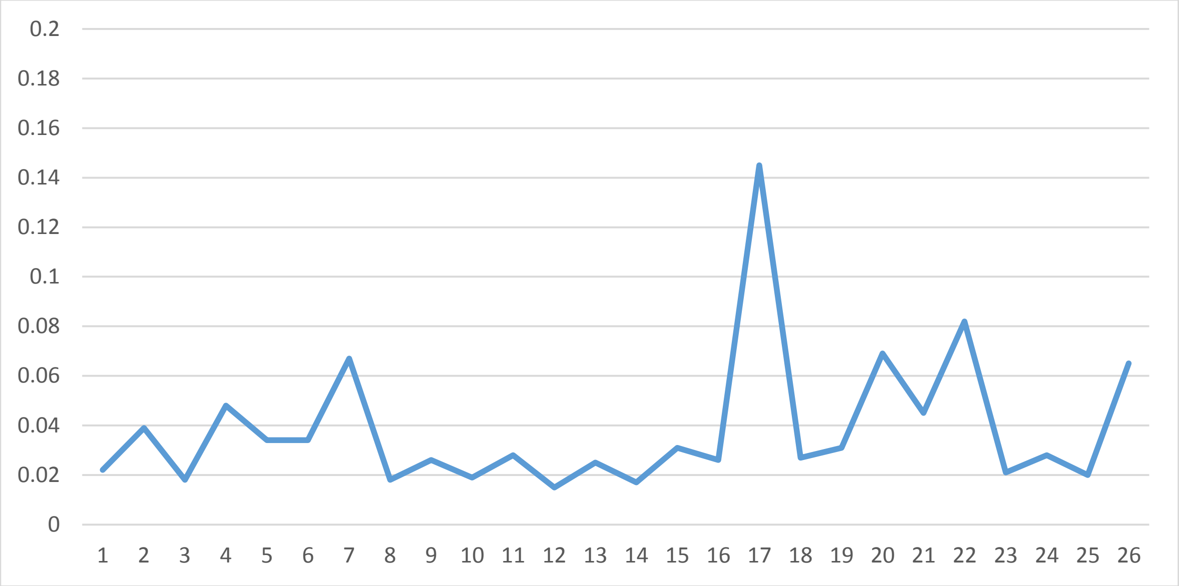


Figure 24: Item probability distribution, test case II

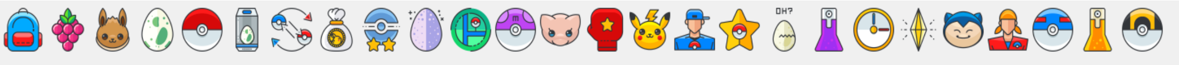


Figure 25: Initial menu layout of theme Pokémon, test case II

A rough observation indicates that the button *Star Badge* holds the highest probability. Additionally, no items with *strong* relationships are located close to each other. Moreover, items are not positioned in any sorted order, hence this menu layout has not been optimized previously.

Method one

Optimization method one is able to improve menu performance index from initial value 0.077 to 0.8305. Furthermore, runtime result indicates that method one has successfully improved menu performance in 54 optimization attempts of 1000. The menu layout optimized by method 1 may be displayed as the following,

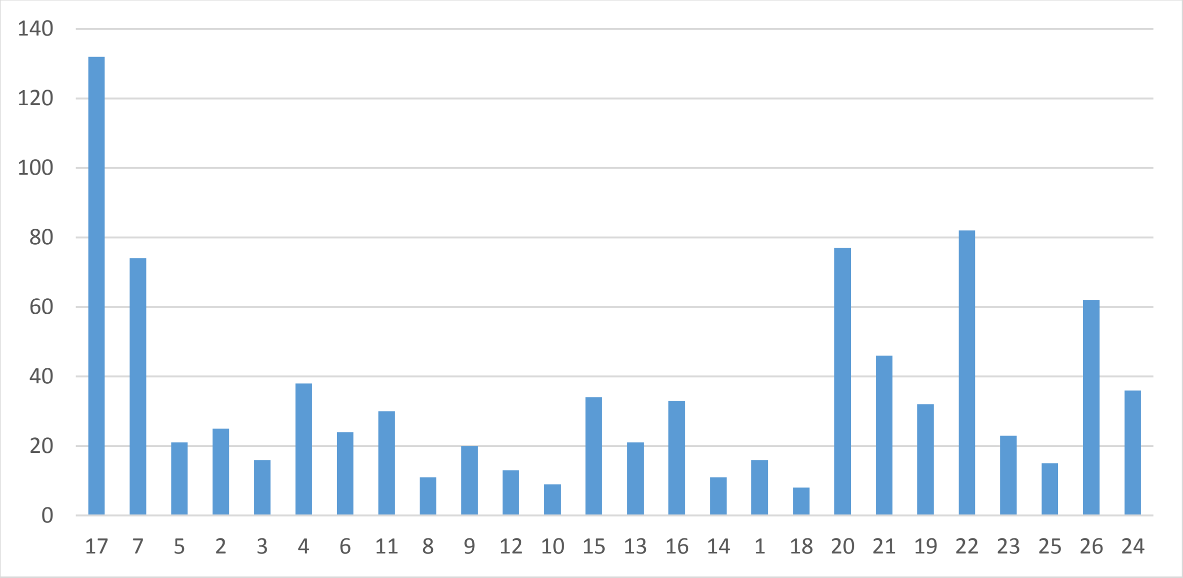


Figure 26: Selection distribution, test case II

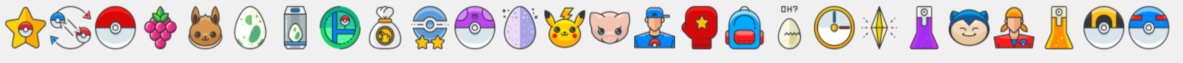


Figure 27: Method one, test case II

Based on Figure 27, it is observed that *Star Badge* ⭐ which has the highest probability and received the most selections, has been positioned as the first item in menu. On the other hand, several items which also hold a considerable amount of selections, such as ◊ and ◓, have not been placed to the top of menu for quicker access, instead they have been located close to another related button in order to achieve semantic improvements, such as ◊ ⚗ and ◓ ◓. However, due to the low weight adjusted to semantic table, method one has primarily optimized menu according to the SDP model, hence, not all items hold close relationships with each other.

In order to analyze further, probabilities of buttons calculated by simulator during test case II runtime are illustrated as the following,

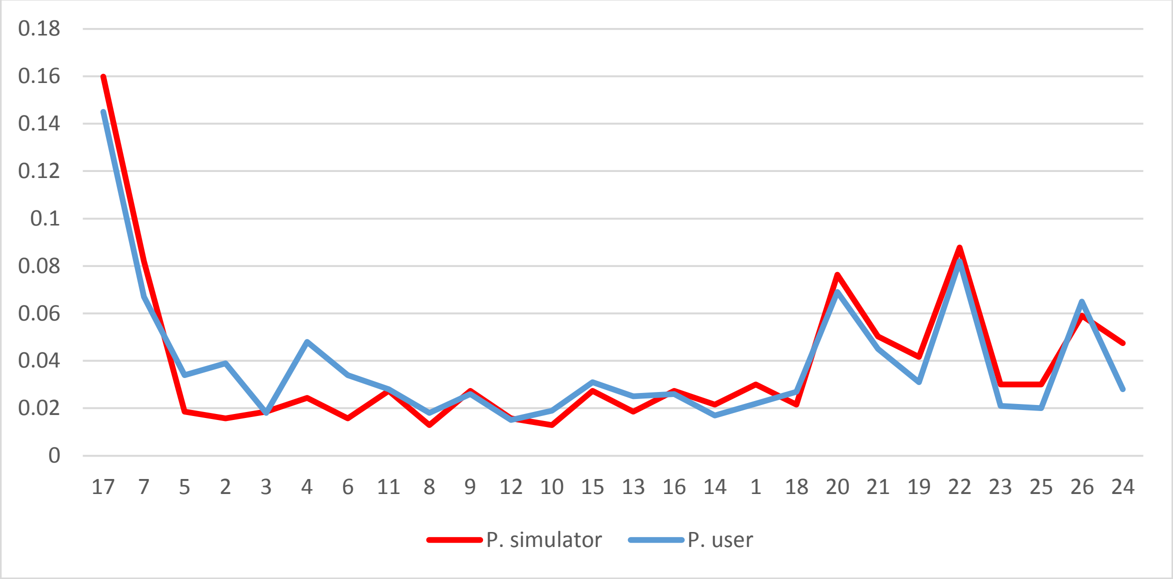


Figure 28: Comparison of probability distributions, test case II

From Figure 28, it is observed that two probability tables appear to be similar to each other. However, a couple of key points are identified. For instance, *Poke Badge* the second button in menu, holds probabilities of 0.067 in user selection table but 0.082 in simulator calculated table. This indicates that the position of *Poke Badge* has not been frequently changed. As previously discussed, each modification done to menu layout (e.g. position of button) would incur a cost which is governed by the cost function, hence reduce a corresponding amount of selections from modified buttons. Because of this, it is understood that the position of button *Poke Badge* may likely be static subsequent to its related optimization modification. Additionally, button *Star Badge* shares similar fate as observed from Figure 28. On the other hand, *Grape* positioned as the forth button in menu, holds an opposite outcome as its probabilities are 0.034 and 0.018 in user and simulator probability table respectively, which implies the fact that its position has been changed various times during simulation runtime.

As previously mentioned, a total amount of 54 optimization attempts have gained improvements regarding menu performance, these attempts and their corresponding menu performance index are illustrated as the following,

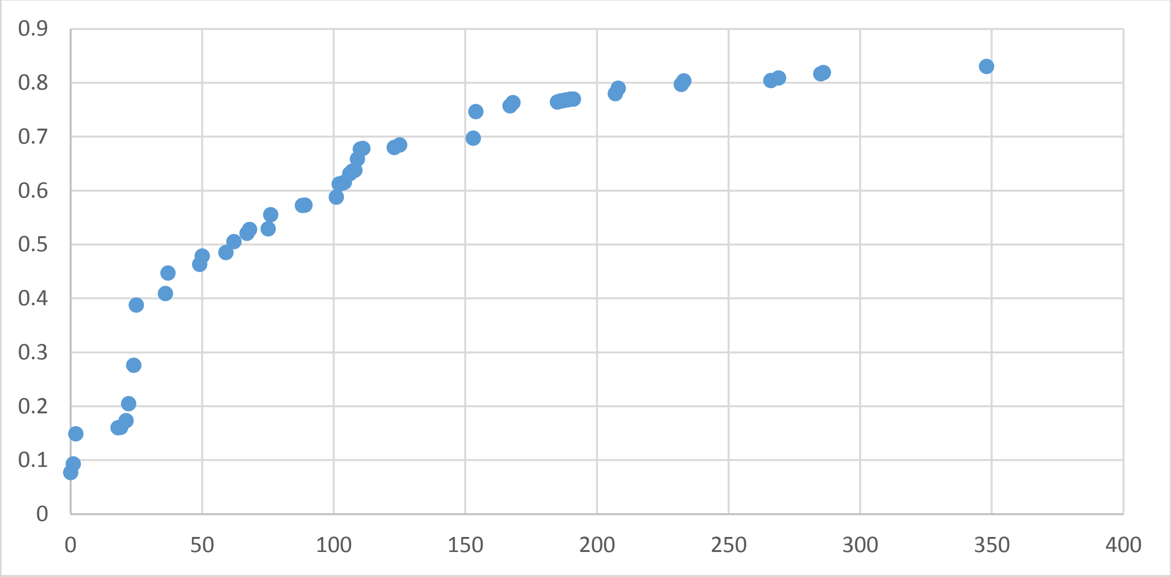


Figure 29: Optimization attempts, test case II

A couple of key points may be noticed from Figure 29. First, a large amount of successful optimization attempts are performed during early part of simulation runtime. Additionally, algorithm has provided less improvement as the simulation keeps running. Hence, optimization algorithm has achieved the highest performance index during test case II at optimization attempt 348, and its corresponding value is 0.8305. Second, the optimization process has demonstrated several “climb up” phases (e.g. from attempt 100 to 120) as successful optimization attempts are more densely performed during these phases. This may be partially explained by the fact that the optimization algorithm has determined a superior menu layout based on semantic performance. However, it is competing with the previous menu layout as their related performance indexes are also continuously influenced by the SDP model. Hence, a temporary conflict is arisen between two performance models. However, one menu layout would ultimately defeat the other one, thus, the optimization algorithm would continue from these “climb up” phases. It is worth noticing that the simulation is limited to 1000 selections, hence, the optimization algorithm has not achieved a genuine *Optimal* result, but rather *Approximately optimal* results [4].

As previously discussed, the *Graphical Simulator* attempts to further optimize the menu layout by several other approaches based on the data generated and received from the *Background Simulator*.

Method two

As previously, Method two attempts to optimize menu layout by dividing current menu into two. Consequently, buttons belong to the second part of menu are relocated from the beginning of menu. As a result, these buttons may be more quickly accessed. The menu layout modified by Method two is illustrated as the following,

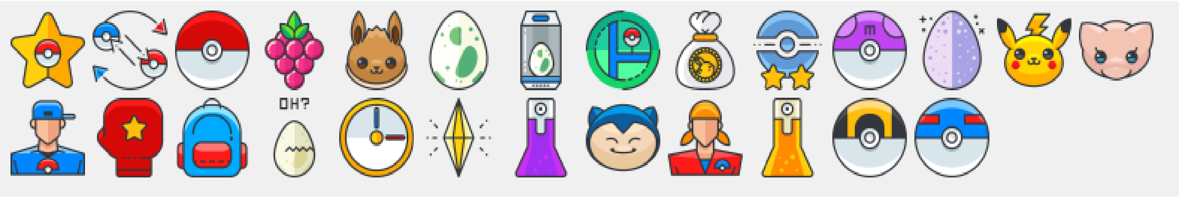


Figure 30: Method two, test case II

Additionally, the amount of selections contained in each row are displayed as the following,

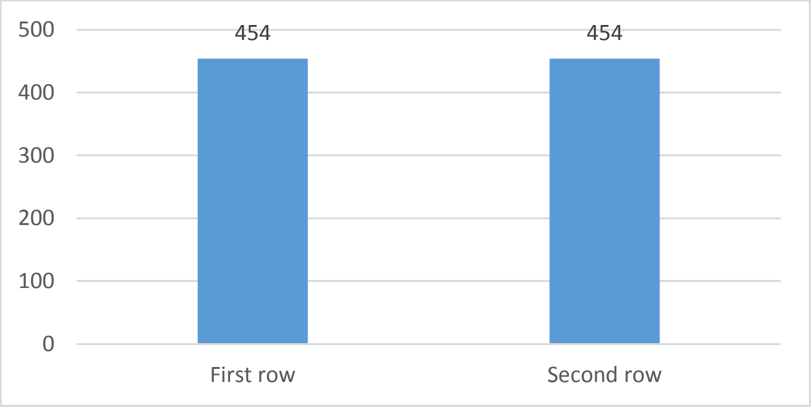


Figure 31: Row ratio, test case II

Where both row equally contain 454 selections. Method two yields a performance index of 0.878 which is approximately 5.7% better compared to Method one. Moreover, the amount of buttons reside in both rows appear to be very well balanced. Hence, the outcome of method two may be indeed feasible for real users.

Method three

As discussed, method three attempts to further optimize menu performance based on the result of method one by utilizing a variety of strategies, including increasing item size, reducing pointing distance and enhance group saliency. Additionally, for testing purpose, buttons are always positioned as a group of five in this thesis. The modified menu layout of Method three is illustrated as the following,

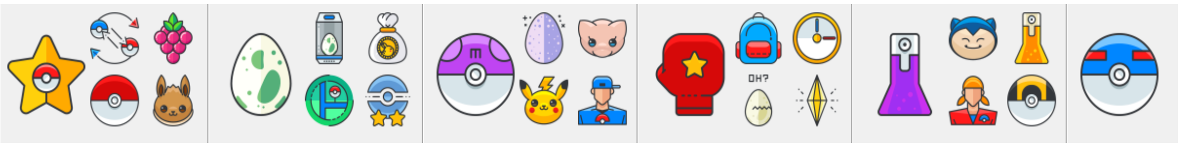


Figure 32: Method three, test case II

According to the *Graphical Simulator*, Method three yields a performance index of 0.875 which is around 5.4% better than method one.

Method four

The selection distribution and menu layout constructed by method four is illustrated as the following,

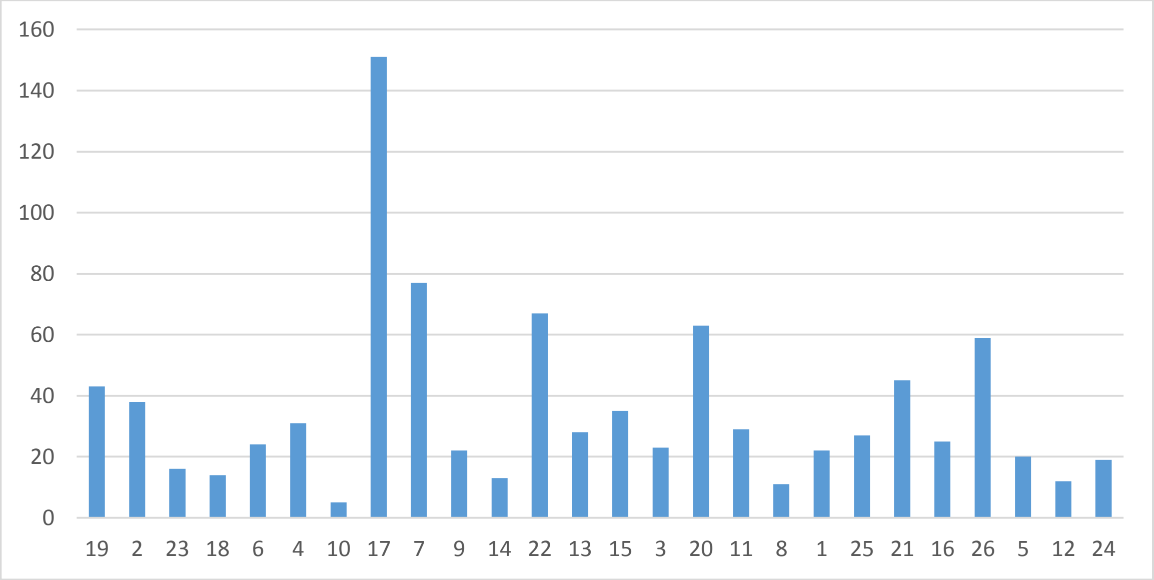


Figure 33: Selection distribution, Method four, test case II

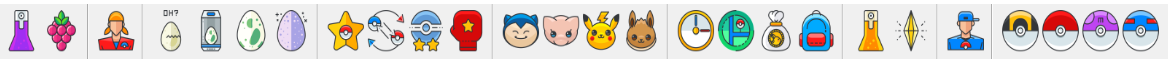


Figure 34: Method four, test case II

As opposite to other methods, method four is executed on a parallel process as method one by utilizing the same data base. Moreover, method four aims to group items by their semantic relationships and a separator is drawn between each group for enhancing group saliency. As a result, the SDP model contributes little regarding the modification of menu layout in method four. Furthermore, the weight for method four in test case II is adjusted to be much higher compared to the one in test case III. Hence, the outcome of method four in test case II may be much more prominent.

Based on Figure 34, it is observed that the menu layout has been fundamentally crafted with semantic relations considered. Indeed, the menu layout is divided into 9 chunks while 5 of them may be understood as *optimal chunks*. That is, all buttons reside in these chunks possess strong relations between each other, thus provide a high semantic performance index. On the other hand, method four largely neglects the influence of the SDP model which may be noticed in Figure 34, where the amount of selections of buttons have rather little effect regarding button positioning compared to the one in method one.

According to the *Graphical simulator*, method four offers a performance index of 0.932 which is approximately 12% higher than method one.

Method five

The method five aims to improve menu performance index by attempting to enlarge the most *significant buttons* based on the data received from method one. Hence, method five shall naturally offer a better performance index compared to method one if no design constraint is violated. The modified menu layout of method five is illustrated as the following,

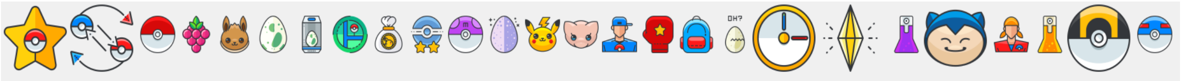


Figure 35: Method five, test case II

As noticed from Figure 35, six buttons have their sizes enlarged by method five. According to the design constraints of method five, a total amount of eight buttons may be enlarged without altering the sizes of other buttons. Because of this, Method five may still offer improvements if the simulation may be executed for a longer duration. According to the

Graphical Simulator, the performance index of method five is 0.86 which is 3.6% better than method one.

Summary: test case II

This section concludes the results obtained from all five methods in the light of the original hypothesizes of test case II. The overall results of test case II are displayed as the following,

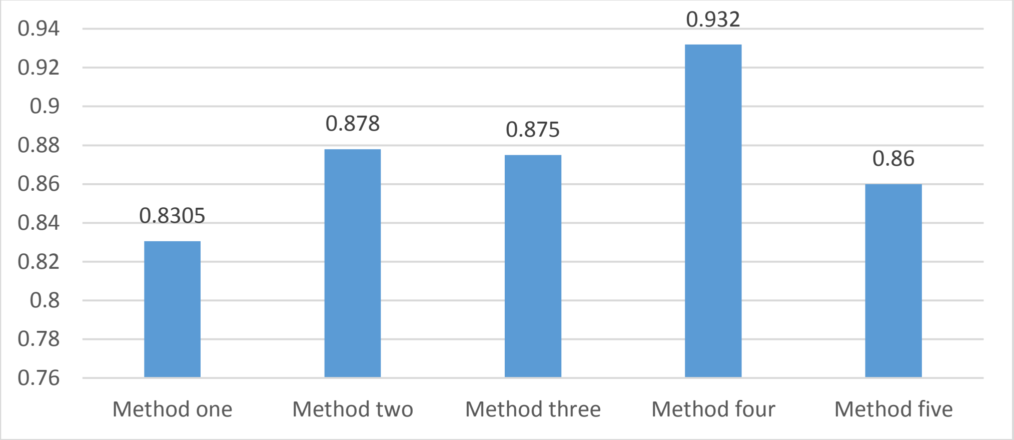


Figure 36: Overall result, test case II

1. Method two offers a significant improvement compared to method one

Indeed, method two offers a 5.7% improvement over method one. Additionally, its menu layout appears to be very well balanced compared to the ones in test case I. Because of this, if design constraints are not violated, the result of method two shall be considered in real use cases.

2. The performance of method three is considerably worse than Method four

As observed, due to the clearly defined relations between buttons and appropriate weights adjusted to performance models, method four is able to offer a significant performance improvement compared to method three (i.e. 6.5%). Hence, this hypothesis is proven to be true. Additionally, due to the clear layout provided by method two and strong semantic model performance weight, method three shall not be considered in real user cases.

3. Method five performs better than method three

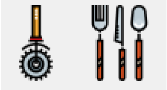
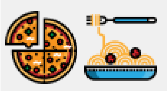
It is obvious that method five has offered a weaker performance index compared to Method three (i.e. 1.7%) despite providing better menu learnability. Hence, this hypothesis is proven to be false. This may be partially explained by the fact that method five benefits from long simulation durations since menu modifications are performed infrequently in later optimization phases. Because of this, items with high probability are likely to regain their selections which are reduced due to menu modifications occurred in early optimization phases. Conversely, method three receives less benefits from extended simulation duration.

Depends on the favor of designers, two final designs may be considered in test case II. If short item access distance is preferred, then the result of method two shall be utilized. On the other hand, if semantic grouping provides more value to users, then the layout of method four shall be considered.

5.3.3 Test case III: Theme Restaurant

Restaurant has been one of the most famous places to visit for many people. Everyone has probably enjoyed dinners with their families in restaurant. Goods and services of restaurant are mostly delivered via menu to its customers. Hence, this test case is designed based on the theme of a restaurant menu.

This test case consists of three purposes. First, the skew of Zipfian distribution has been decreased to a lower degree in order to further analyze the reaction of the *Background Simulator*. Second, test case III attempts to identify the key factors which hold the most influence regarding the performance of semantic model, thus, the semantic table provided in test case III has been increased to eight categories. Furthermore, each category contains different number of items instead of four items per group configured in test case II. Third, a "catch up" feature has been implemented into the performance weight factors of method one. This feature modifies performance models in such way that it may reflect an actual restaurant operation strategy. That is, restaurants tend to sell the most profitable goods in the beginning of a supply cycle. Conversely, superfluous goods should be sold with high priority in the end of a supply cycle in order to minimize wastes. This "catch up" feature is referred as *cumulative weight* which is discussed later. The buttons utilized and their corresponding classification are displayed as the following,

Vegetables	
Tools	
Dishes	
Drinks	
Seafood	

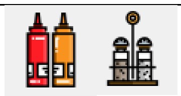
Flavors	
Fruits	
Desserts	

Figure 37: Buttons and categories, test case III

As previously mentioned, the relationships between categories are represented by different grades. The grading system utilized in test case III is somewhat identical to the one in test case II. That is, four relationship grades are available for each button, including strong, natural, weak and poor. No category is considered as special category, hence, a total amount of eight categories are configured. The semantic table is illustrated as the following,

	Vegetable	Drink	Tool	Seafood	Fruit	Dessert	Dish	Flavor
Vegetable	**S**	**P**	**P**	**P**	**N**	**P**	**W**	**N**
Drink	**P**	**S**	**P**	**N**	**W**	**P**	**N**	**P**
Tool	**P**	**P**	**S**	**W**	**P**	**N**	**N**	**P**
Seafood	**P**	**N**	**W**	**S**	**P**	**P**	**P**	**N**
Fruit	**N**	**W**	**P**	**P**	**S**	**N**	**P**	**P**
Dessert	**P**	**P**	**N**	**P**	**N**	**S**	**P**	**W**
Dish	**W**	**N**	**N**	**P**	**P**	**P**	**S**	**P**
Flavor	**N**	**P**	**P**	**N**	**P**	**W**	**P**	**S**

Table 5: Semantical table, test case III

In test case III, simulator runtime parameters are defined as the following,

Total simulated selections	1000
Optimization frequency	1
performance model weight (SDP : Semantic)	6:4
Cost factor	0.04

Skew of Zipfian distribution	0.4
Cumulative weight	Enabled

Table 6: Simulation parameters, test case III

As noticed, the **Skew of Zipfian distribution** has been reduced to 0.4 in test case III. This should result in a more uniformed probability table compared to previous test cases. The sorted probabilities of items are displayed as the following,

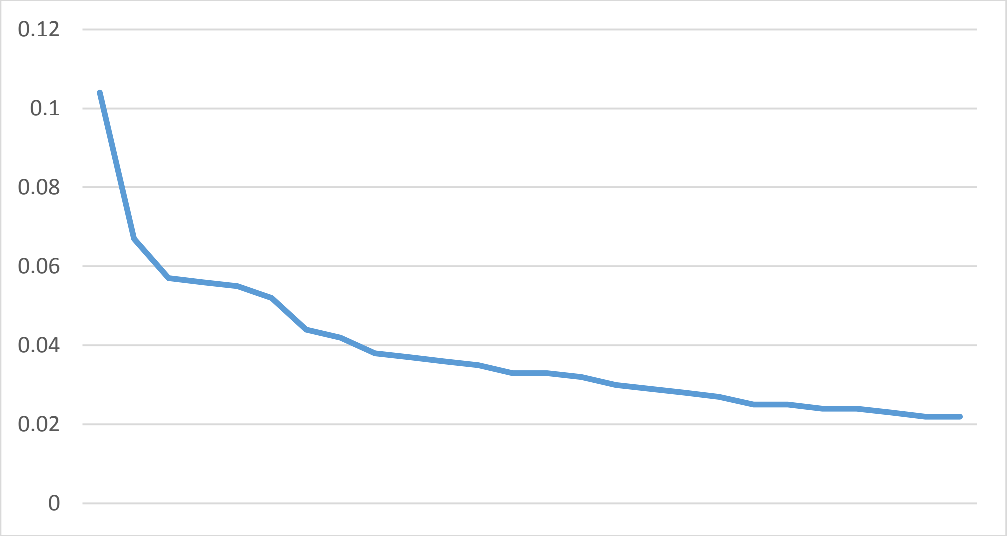


Figure 38: Item probability distribution sorted, test case III

It is noticed that the long tail effect still exists, however, the curve of probability distribution appears to be less sharp. Furthermore, a rough observation indicates that at least 7 items have achieved a significant amount of probabilities.

The **weight of performance models** is defined as 6:4 in all five methods including method four. On the other hand, the **cumulative weight** is applied to method one, hence the performance model weights in method one may be referred to as **target performance model weight**. As previously mentioned, the **cumulative weight** factor creates a “catch up” effect for models. That is, initially one model holds a weight of 1 while the other 0. As more items are continuously being selected. The **cumulative weight** increases for the other model, until it reaches the **target performance model weight**. The parameter describes the fact that subsequent to each item selection, the number of the corresponding item is reduced in the storage of restaurant. As a result, items located in the end of menu receive less attention which ultimately create waste. The **cumulative weight** attempts to minimize this effect. It is calculated as the following,

$$Cumulative\ weight = \frac{W_{se}}{target\ selection}$$

where *target selection* is the number of selections required to reach the **target performance weight**. The valid value for this parameter is between one and **simulation duration**. In test case III, the **cumulative weight** is implemented in method one.

The **simulation duration** and **optimization frequency** are defined as 1000 and 1 respectively according to similar reasons in test case II. Moreover, **cost factor** has also been adjusted to 0.04 in order to facilitate the frequency of optimization attempts.

Based on the simulation parameters in test case III, several **hypothesizes** are presumed as the following,

1. *Method one and four provide different results*
2. *The result of method two is somewhat more feasible compared to previous test cases*
3. *Method five provides better result than method three*

The initial menu layout and item probability distribution are displayed as the following,

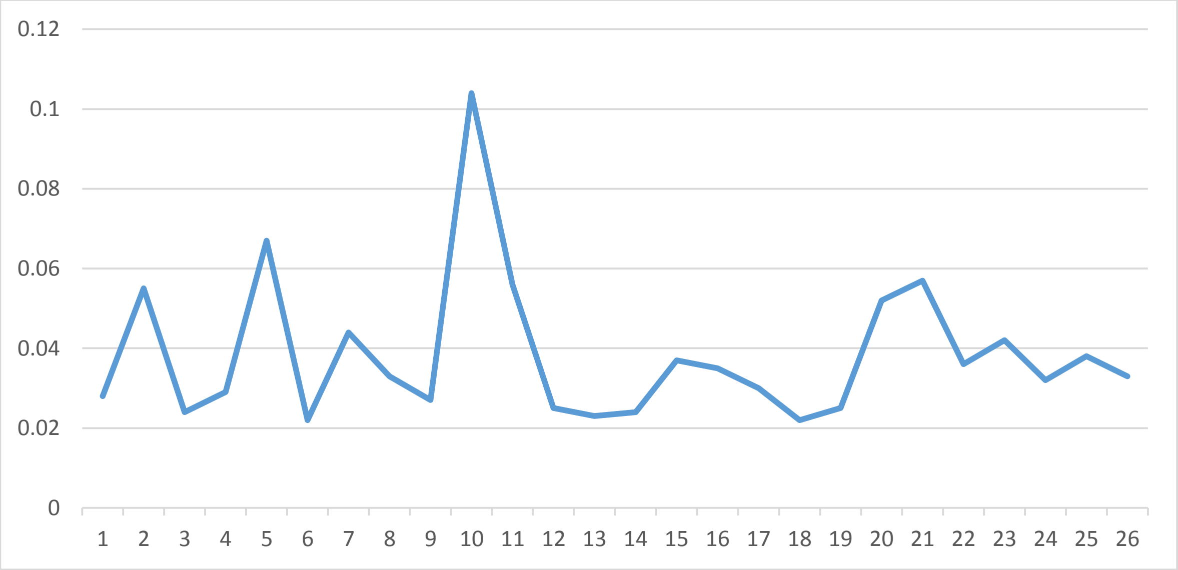


Figure 39: Item probability distribution, test case III

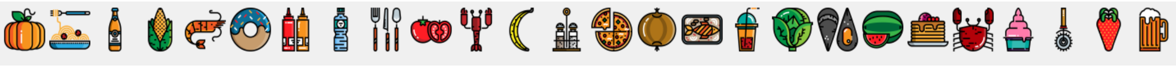


Figure 40: Initial menu layout, test case III

Initial observation indicates that several buttons hold superior probabilities, including *tomato* , *shrimp* *and pancake* . Additionally, the initial menu layout is not in a sorted order. Furthermore, no items are grouped close to another item of the same category. Hence, the performance of this menu is in its initial state.

Method one

The cumulative weight parameter is applied to the semantic model. That is, the performance weight of the SDP model largely dominates in the early optimization phases. Because of this, despite having a rather balanced ratio in both models (6:4), the semantic model receives less attention in the beginning of the simulation. However, the performance weight is continuously being increased for the semantic model, hence, some relation patterns may be identified. The menu layout as well as the item selection distribution of method one is displayed as the following,

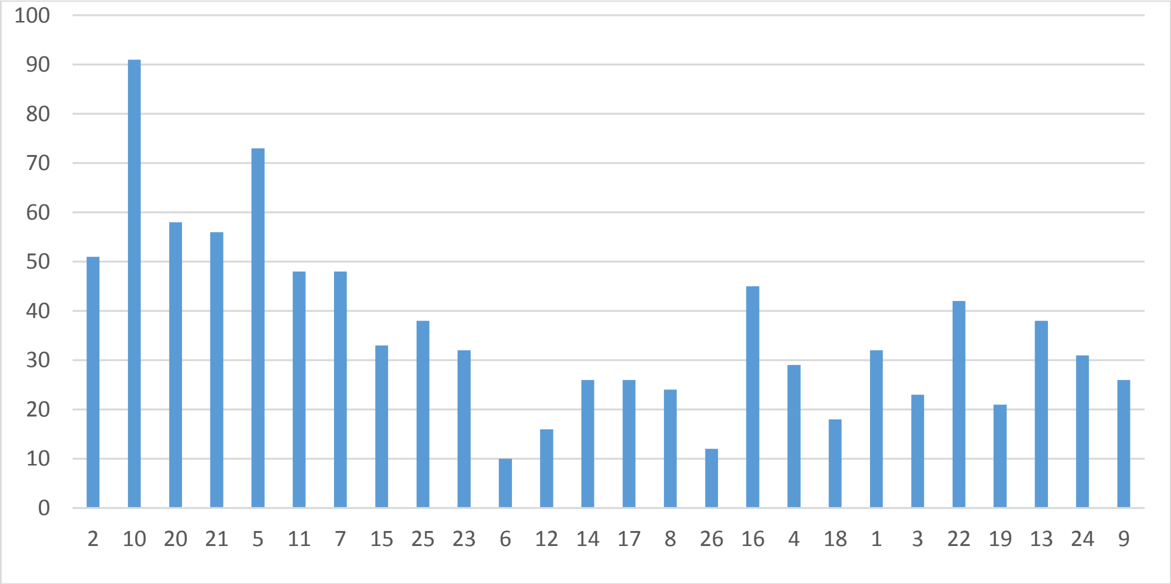


Figure 41: Selection distribution, method one, test case III

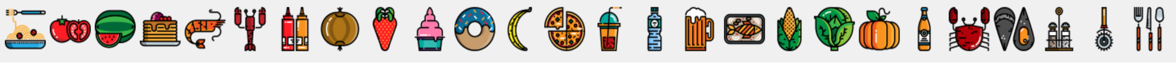
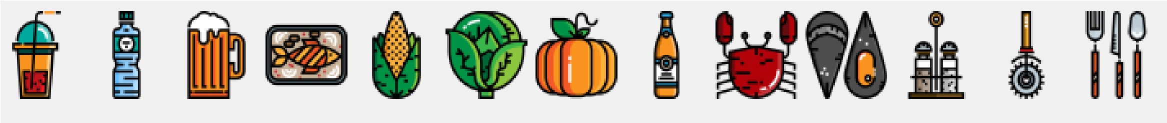


Figure 42: Method one, test case III

Based on Figure 42, it is observed that the menu layout appears to be somewhat similar to the one in test case II. This indicates that several buttons which hold decent amount of selections, have their positions located close to another button for semantic improvement, such as *water* *and cabbage* . It is worth noticing that the weight of semantic model is extremely low in the beginning of simulation. Because of this, users are more likely to have already become experienced with the menu layout prior to semantic model begins to influence the menu layout. Despite the fact of this, the semantic model has been able to motivate method one to modify the menu layout in later simulation phases. Several relation patterns may be identified, for instance *drinks* , *vegetables* , *seafood* *and tools* . It may be slightly ambiguous, however, the second half of the menu holds a rather high overall semantic performance, their relationships may be demonstrated as the following,



S	**S**	**N**	**P**	**S**	**S**	**P**	**N**	**S**	**N**	**P**	**S**

Figure 43: Semantical relationships, method one, test case III

On the other hand, the first half of menu layout has not been primarily optimized based on the semantic model. This phenomenon expresses two points. First, the first half of menu layout has been processed by the SDP model for a somewhat long duration prior to the semantic model. Second, buttons with significant amount of selections are discouraged to be modified due to the high user expertise in later optimization phases, unless the modification receives superior benefits regarding semantic relations which may diminish the negative effect of the cost function. Because of this, the second half menu holds relatively superior semantic grades compared to the first half menu. The *Background Simulator* reports the performance index is 0.826.

Method two

As previously, method two attempts to optimize menu layouts by dividing it into two rows. A feasible solution would result in quick access for menu items, thus increases the performance of menu layout. In test case III, method two decides to modify the menu layout as the following,

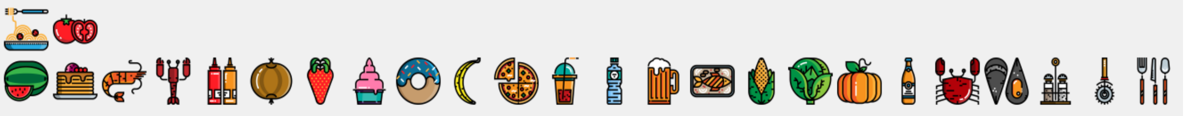


Figure 44: Method two, test case III

The *Graphical Simulator* indicates a performance index of 0.869 for this menu layout, which is 5.2% improvement compared to method one. However, it is obvious that the menu layout is rather unbalanced, hence this menu layout is very well infeasible in real user cases.

Method three

Similar to test case II, method three attempts to group buttons by the overall results of method one. In the light of testing, buttons are positioned in a group of five. The modified menu layout is illustrated as the following,

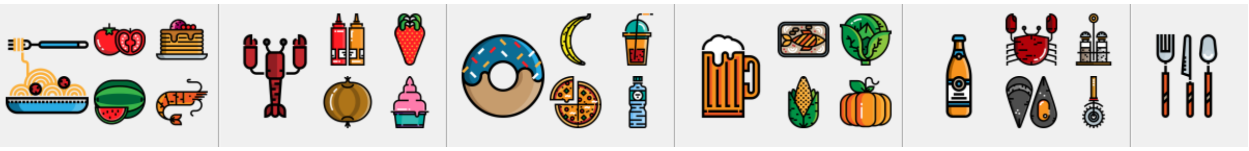


Figure 45: Method three, test case III

Based on the *Graphical Simulator*, method three offers a performance index of 0.836, which is approximately improved by 1.2% compared to method one. Hence, method three shall be considered in real user cases where the result of method two appears to be infeasible.

Method four

In contrast to method one, method four attempts to group items based on their semantic relations. That is, the semantic model in method four holds more performance weight than the SDP model in normal situations. However, in test case III the performance weights are identical in both methods (i.e. SDP (6) : Semantic (4)). On the other hand, the **cumulative weight** parameter is applied in method one, hence affects its overall performance. Because of this, the results of method one and four shall be different despite utilizing the exact same data base and holding the same performance model weights. The item selection distribution and the menu layout constructed by method four are illustrated as the following,

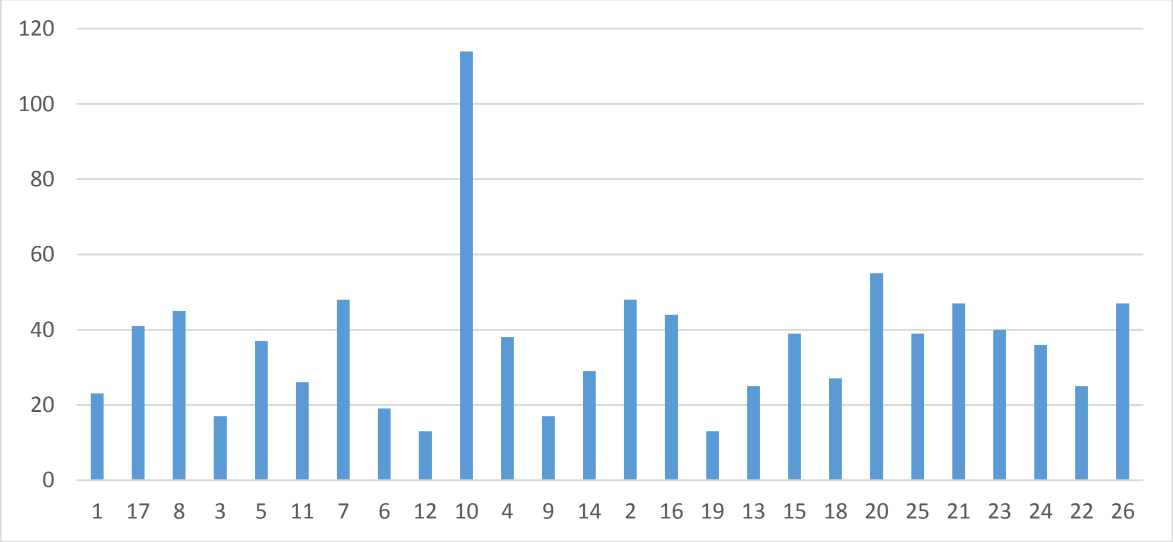


Figure 46: Selection distribution, Method four, test case III

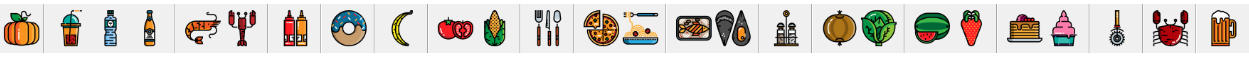


Figure 47: Method four, test case III

Despite still having a higher performance weight in SDP model, the menu layout represents few to no similarities compared to the one in method one. This is because the **cumulative weight** parameter is absent in method four. Consequently, the semantic model begins to influence the menu layout immediately from beginning of the simulation. On the other hand, menu performance model contributes less due to the lack of data in very early simulation phases. Because of these, the menu layout of method four appears to be more influenced by semantic model instead of performance model.

The *Graphical Simulator* reports a performance index of 0.8317 which is approximately 0.7% higher than method one.

Method five

As previously mentioned, method five attempts to improve menu performance by enlarging the sizes of the most *significant buttons*. The total amount of enlarged *buttons* is defined as eight in order to prevent design constraint violation. On the other hand, it is also possible to reduce the size of *insignificant buttons* to allow more *significant buttons* to be enlarged. The menu layout crafted by method five is illustrated as the following,

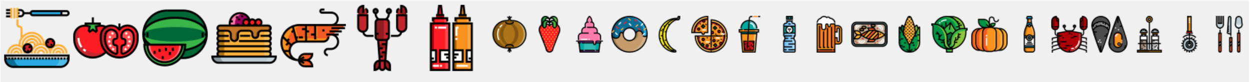


Figure 48: Method five, test case III

As noticed, a total of seven buttons have their size enlarged, hence, one more button may still be enlarged if the duration of simulation is extended as method five benefits from longer simulation runs. According to the *Graphical Simulator*, the performance index offered by method five is 0.8477 which is 2.7% higher than method one.

Summary: test case III

1. Method one and four provide different results

Indeed, method one and four provide completely different menu layouts despite utilizing the same data base. This is caused by the **cumulative weight** parameter applied in method one. As previously mentioned, **cumulative weight** is utilized in order to delay the effect of another performance model. Consequently, this reflects the fact that restaurants typically would like to offer their customers different goods at different supply cycle. For instance, goods with high profitability are sold in early supply cycle while supplementary goods are better to be sold in late supply cycle in order to minimize wastes. Because of this, the menu performance weight for a restaurant shall be flexible to perform this strategy.

The overall results of test case III are displayed as the following,

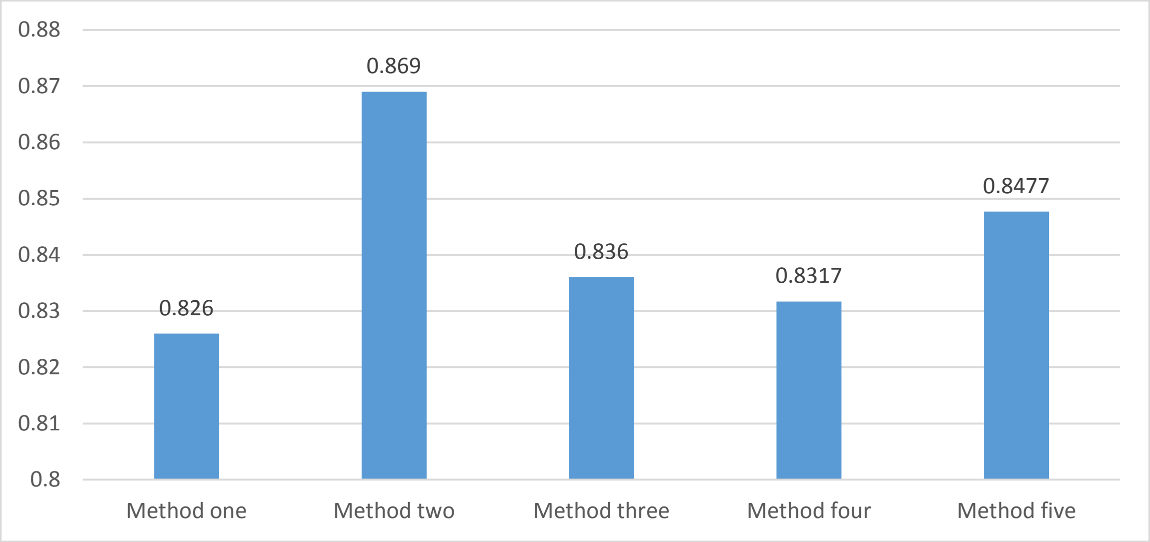


Figure 49: Overall result, test case III

2. The result of method two is somewhat more feasible compared to previous test cases

It may be assumed that the menu layout of method two shall be more feasible due to a more uniformly distributed item probabilities in test case III. However, method two still failed to provide a feasible solution because of unbalanced menu layouts.

3. Method five provides better result than method three

Indeed, the result of method five is slightly better than method three. This may be explained by the fact that the most *significant buttons* are already very well optimized by method one. Conversely, in other test cases, enlarged buttons are located all over the menu layout instead of in the beginning of menu, hence, the effect of enlargement is much stronger in method five, test case III, thus provides better performance than method three. Because of this, the result of method five shall be considered as the final design in test case III.

Chapter summary

As discussed, method one has been able to improve menu layout in all test cases. On the other hand, method two has provided only one feasible menu layout despite offering superior performance index in most situations. Conversely, method three is able to provide usable menu layouts in all test cases although having a lower performance index compared to method two. On the other hand, due to the superior performance index gain, method four shall be considered if semantic table is available. Moreover, designers may force the simulator to produce menu layouts completely based on semantic model. This feature may be useful in circumstances similar to test case restaurant. However, if semantic table is absent, method five may always improve menu performance by enlarging the most *significant buttons*. However, the number of buttons enlarged shall not exceed the maximum amount configured by its design constraints.

Several key points are discovered based on the test results,

- Optimization attempts are more successfully performed in early phases than in later phases. This may be caused by the fact that users have already gained relatively high user expertise in later phases, thus reducing the needs to further improve menu performance by modifying menu layout.
- Cost factor greatly reduces the likelihood of successful optimization attempt. This phenomenon is observed in test case I where the cost factor for *Run 2* is of the twice of *Run 1*. As a result, *Run 2* has performed less than half of the amount of successful optimizations compared to *Run 1*. Because of this, the cost factor is a highly important variable which shall be carefully defined.
- Based on the comparison between method five and two, button size enlargement appears to contribute less than reducing the movement required to reach a button. This may be explained by the fact that the increment of button size is insufficient in these test cases due to design constraints. On the other hand, the amplitude of movement of buttons have been greatly reduced by method two despite providing infeasible menu layouts. Hence, these two parameters shall equally contribute if not limited by design constraints.

- Pointing time and semantic model are not affected by user expertise. As a result, designers may be able to exploit this fact by forcibly locating buttons with strong semantic relations to the beginning of menu for quicker access and superior semantic menu performance. This action may potentially increase search time for novice users, thus decrease general menu performance. However, this may greatly improve menu performance once user have become experienced with the menu layout. This may be explained by the fact that the performance indexes of semantic model and pointing time have been maximized in early phase while the decision and search time may be improved overtime by user expertise. However, this approach requires further work to be verified.

6. Conclusion and future work

This thesis has studied the fundamental aspects regarding menu performance. Centered on these aspects, a predictive menu performance model has been proposed to evaluate the performance of menu designs in this thesis. This model is based on the Search-Decision-Pointing model introduced by Cockburn et.al in previous works. In addition to the original parameters defined in the SDP model, the model of this thesis has taken consideration of two additional parameters, i.e. the cost of modifying an item and the degree of semantic relations between items. Moreover, the exhaustive search algorithm has been implemented to explore the design space of the simulator software programmed in this thesis. The simulator software emulates the user interface of a famous editor application known as Notepad++. There, buttons are horizontally organized on a linear layout. Centered on this user interface, several optimization methods have been designed. The performance of these methods vary depends on the test cases where they are executed. Based on the result of the evaluation chapter, the optimization methods introduced in this thesis have improved menu performance in all test cases. In fact, the results indicate that the improvement is rather significant depends on methods and test cases. Hence, it may be concluded that optimization methods proposed in this thesis are able to improve to menu performance in general. On the other hand, several key points and limitations have also been identified,

- The user expertise parameter in the objective function is calculated in the same convention regardless of menu properties. However, the process for novice users to become experts should vary depends on the characteristics of a menu system. For instance, a user is likely to require more practices to become familiar with a menu of many items [2].
- Furthermore, the cost for changing the position of an item shall scale with other parameters, including item probability and utilization duration. For instance, modifications should cost significantly more for an item that is frequently selected compared to an item that is rarely selected.
- The search algorithm implemented is a variation of the exhaustive search algorithm. This algorithm is sufficient only for menu with few items. Its efficiency greatly fails

for larger menus. Hence, the exhaustive search algorithm should be replaced by other heuristic algorithm, such as the simulated annealing or genetic algorithm. [11] [22]

- The simulator assumes that all users are initially novices. Moreover, the learning curve and process are also considered to be the same for all users. However in reality, users may be defined by three distinctive groups, such as neophyte user, inexperienced user and expert user [13]. Each user group has its own learning curve, thus shall be provided with different objective functions and optimization methods.
- Only one modality is considered in this study, i.e. items are executed by mouse. However, other modalities, such as shortcuts and mnemonics have been widely applied in commercial applications. These modalities are often utilized by expert users and offer significant menu performance improvements. [16] [20]
- Increasing the size of items offers surprisingly little benefit compared to other strategies. This point has also been proved in the study conducted by Cockburn et.al. [2].

Overall, this thesis has been able to improve menu performance on a linear menu layout by implementing a variety of optimization methods centered on the SDP model introduced by Cockburn et.al and optimization strategies proposed by Bailly et.al in a simulated environment. Centered on the findings of this study, some possible future work could be performed. For instance, McGrenere et.al has proposed a study regarding the performance comparison between static, adaptive and adaptable menus. In their research, users have favored the static and adaptable menu instead of the adaptive menu. Moreover, they have concluded that the static and adaptable menus perform significantly better than the adaptive menu where menu optimizations are executed automatically by the system [21]. Because of this, the menu performance models and optimization methods of this thesis shall be evaluated in real user cases on a commercial software in addition to be performed on a simulator.

The optimization methods have largely concentrated on improving menu performance regarding item access speed and selection accuracy. On the other hand, other criteria have been determined to be as vital as speed and accuracy, such as interface satisfaction and menu learnability [13] [22]. In addition, the item selections are performed by simulated users in this study, hence, a more appropriate learning model shall be considered in real user cases. [17]

Moreover, user preference shall also be considered when designing optimization methods [22] [24]. Furthermore, the influence of item semantic relations may be further studied to better balance the weight between performance models. Because of these, a more comprehensive menu performance model is required to balance the focus between different criteria depending on the user profile and application in question.

References

[1] Andrew Sears and Ben Shneiderman. 1994. Split menus: effectively using selection frequency to organize menus. *ACM Trans. Comput.-Hum. Interact.* 1, 1 (March 1994), 27-51.

[2] Andy Cockburn, Carl Gutwin, and Saul Greenberg. 2007. A predictive model of menu performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '07). ACM, New York, NY, USA, 627-636.

[3] Anthony J. Hornof and David E. Kieras. 1997. Cognitive modeling reveals menu search in both random and systematic. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems* (CHI '97). ACM, New York, NY, USA, 107-114.

[4] Antti Oulasvirta, Andreas Karrenbauer. 2017. Combinatorial optimization for interface design.

[5] BAILI LIU, GREGORY FRANCIS, GAVRIEL SALVENDY, Applying models of visual search to menu design, International Journal of Human-Computer Studies, Volume 56, Issue 3, 2002, Pages 307-330

[6] David Ahlström, Andy Cockburn, Carl Gutwin, and Pourang Irani. 2010. Why it's quick to be square: modelling new and existing hierarchical menu designs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '10). ACM, New York, NY, USA, 1371-1380.

[7] David Ahlström. 2005. Modeling and improving selection in cascading pull-down menus using Fitts' law, the steering law and force fields. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '05). ACM, New York, NY, USA, 61-70.

[8] D.F Jones, S.K Mirrazavi, M Tamiz, Multi-objective meta-heuristics: An overview of the current state-of-the-art, European Journal of Operational Research, Volume 137, Issue 1, 16 February 2002, Pages 1-9

[9] Don Hu. Notepad++. Available: https://notepad-plus-plus.org/

[10] Eric Lee and James Macgregor. 1985. Minimizing User Search Time in Menu Retrieval Systems. Human Factors. Pp. 157 - 162

[11] Gilles Bailly, Antti Oulasvirta, Timo Kötzing, and Sabrina Hoppe. 2013. MenuOptimizer: interactive optimization of menu systems. In *Proceedings of the 26th annual ACM symposium on User interface software and technology* (UIST '13). ACM, New York, NY, USA, 331-342.

[12] Gilles Bailly, Antti Oulasvirta, Duncan P. Brumby, and Andrew Howes. 2014. Model of visual search and selection time in linear menus. In *Proceedings of the SIGCHI*

Conference on Human Factors in Computing Systems (CHI '14). ACM, New York, NY, USA, 3865-3874.

[13] Gilles Bailly, Eric Lecolinet, and Laurence Nigay. 2016. Visual Menu Techniques. *ACM Comput. Surv.* 49, 4, Article 60 (December 2016), 41 pages.

[14] Hick, W. E. (1952). On the rate of gain of information. Quarterly Journal of Experimental Psychology, 4, 11-26.

[15] Hollink, V. & van Someren, M., 2006. Validating Navigation Time Prediction Models for Menu Optimization. 14. GI-Workshop "Adapitivität und Benutzermodellierung in interaktiven Softwaresystemen".

[16] Joey Scarr, Andy Cockburn, Carl Gutwin, and Philip Quinn. 2011. Dips and ceilings: understanding and supporting transitions to expertise in user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '11). ACM, New York, NY, USA, 2741-2750.

[17] Jussi P. P. Jokinen, Sayan Sarcar, Antti Oulasvirta, Chaklam Silpasuwanchai, Zhenxin Wang, and Xiangshi Ren. 2017. Modelling Learning of New Keyboard Layouts. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (CHI '17). ACM, New York, NY, USA, 4203-4215.

[18] Kashyap Todi, Daryl Weir, and Antti Oulasvirta. 2016. Sketchplore: Sketch and Explore with a Layout Optimiser. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems* (DIS '16). ACM, New York, NY, USA, 543-555.

[19] Krzysztof Z. Gajos, Mary Czerwinski, Desney S. Tan, and Daniel S. Weld. 2006. Exploring the design space for adaptive graphical user interfaces. In *Proceedings of the working conference on Advanced visual interfaces* (AVI '06). ACM, New York, NY, USA, 201-208.

[20] Lane, D., Napier, A., Peres, C., and Sandor, A. The Hidden Costs of Graphical User Interfaces: The Failure to Make the Transition from Menus and Icon Tool Bars to Keyboard Shortcuts. INTERNATIONAL JOURNAL OF HUMAN–COMPUTER INTERACTION, 2005, 133 – 144.

[21] Leah Findlater and Joanna McGrenere. 2004. A comparison of static, adaptive, and adaptable menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '04). ACM, New York, NY, USA, 89-96.

[22] Luigi Troiano, Cosimo Birtolo, Roberto Armenise, and Gennaro Cirillo. 2008. Optimization of menu layouts by means of genetic algorithms. In *Proceedings of the 8th European conference on Evolutionary computation in combinatorial optimization* (EvoCOP'08), Jano Van Hemert and Carlos Cotta (Eds.). Springer-Verlag, Berlin, Heidelberg, 242-253.

[23] Mikhail V. Goubko and Alexander I. Danilenko. 2010. An automated routine for menu structure optimization. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems* (EICS '10). ACM, New York, NY, USA, 67-76.

[24] Mikhail Goubko and Alexander Varnavsky. 2016. Users' preference share as a criterion for hierarchical menu optimization. In *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (EICS '16). ACM, New York, NY, USA, 305-310

[25] Oracle. Java Documentation Standard Edition (Java SE) 8. Available: http://docs.oracle.com/javase/8/

[26] P. M. Fitts, "The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement," Journal of Experimental Psychology, Vol. 47, No. 6, 1954, pp. 381-391.

[27] Shouichi Matsui and Seiji Yamada. 2008. Optimizing hierarchical menus by genetic algorithm and simulated annealing. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation* (GECCO '08), Maarten Keijzer (Ed.). ACM, New York, NY, USA, 1587-1594.

[28] Xiuli Chen, Gilles Bailly, Duncan P. Brumby, Antti Oulasvirta, and Andrew Howes. 2015. The Emergence of Interactive Behavior: A Model of Rational Menu Search. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (CHI '15). ACM, New York, NY, USA, 4217-4226.

[29] Simon Fraser University. What is Mathematical Modeling? Chapter 1. Available: https://www.sfu.ca/~vdabbagh/Chap1-modeling.pdf

[30] Tim Halverson and Anthony J. Hornof. 2008. The effects of semantic grouping on visual search. In *CHI '08 Extended Abstracts on Human Factors in Computing Systems* (CHI EA '08). ACM, New York, NY, USA, 3471-3476.

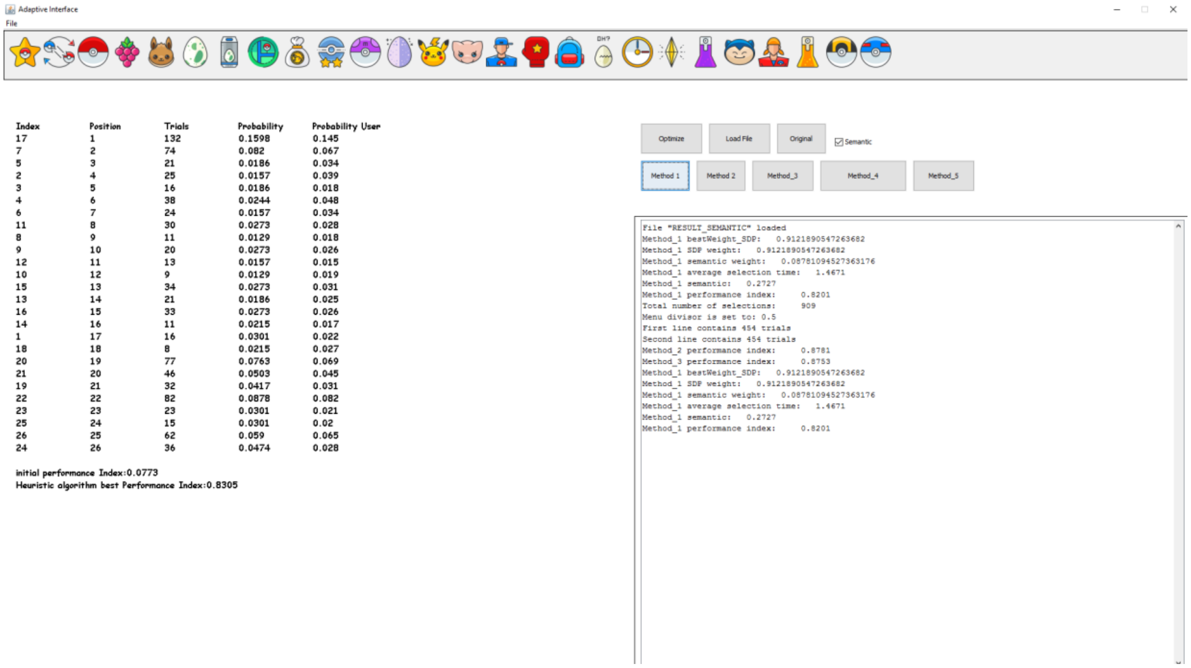
[31] FLATICON. Pokémon go. Available: http://www.flaticon.com/search?word=pokemon%20go

[32] FLATICON. Food and Restaurant. Available: http://www.flaticon.com/packs/food-and-restaurant-3

[33] FLATICON. Office Supplies. Available: http://www.flaticon.com/packs/office-supplies

Appendix

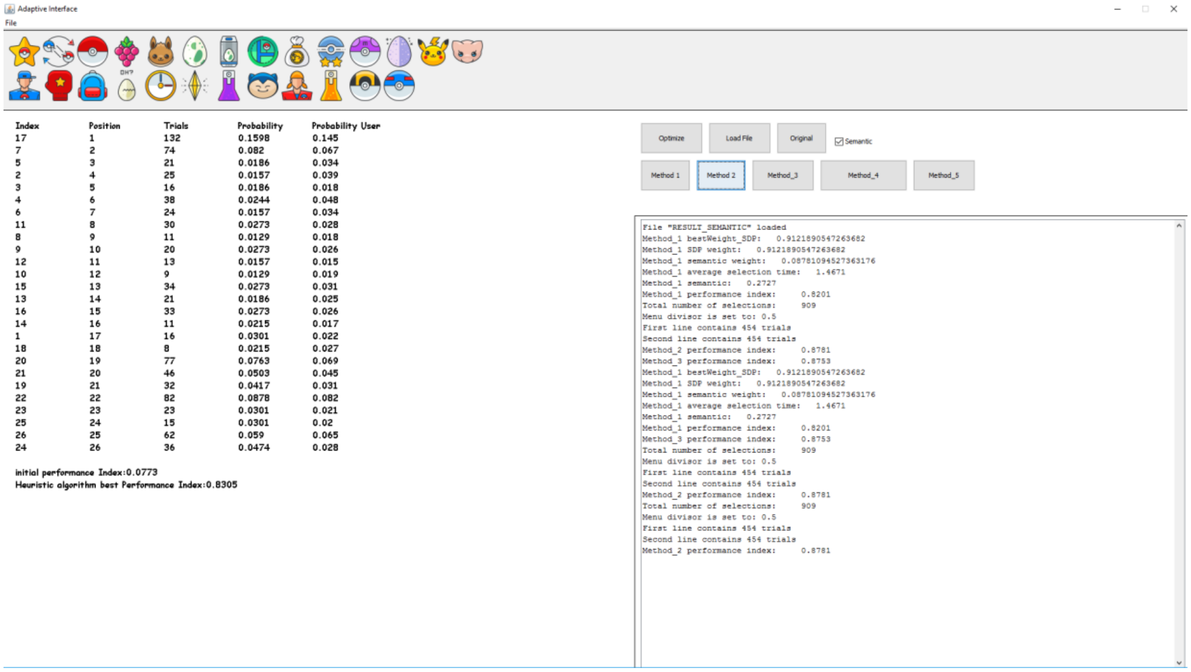
Simulator runtime screenshot



Index	Position	Trials	Probability	Probability User
17	1	132	0.1598	0.145
7	2	74	0.082	0.067
5	3	21	0.0186	0.034
2	4	25	0.0157	0.039
3	5	16	0.0186	0.018
4	6	38	0.0244	0.048
6	7	24	0.0157	0.034
11	8	30	0.0273	0.028
8	9	11	0.0129	0.018
9	10	20	0.0273	0.026
12	11	13	0.0157	0.015
10	12	9	0.0129	0.019
15	13	34	0.0273	0.031
13	14	21	0.0186	0.025
16	15	33	0.0273	0.026
14	16	11	0.0215	0.017
1	17	16	0.0301	0.022
18	18	8	0.0215	0.027
20	19	77	0.0763	0.069
21	20	46	0.0503	0.045
19	21	32	0.0417	0.031
22	22	82	0.0878	0.082
23	23	23	0.0301	0.021
25	24	15	0.0301	0.02
26	25	62	0.059	0.065
24	26	36	0.0474	0.028

initial performance Index:0.0773
Heuristic algorithm best Performance Index:0.8305

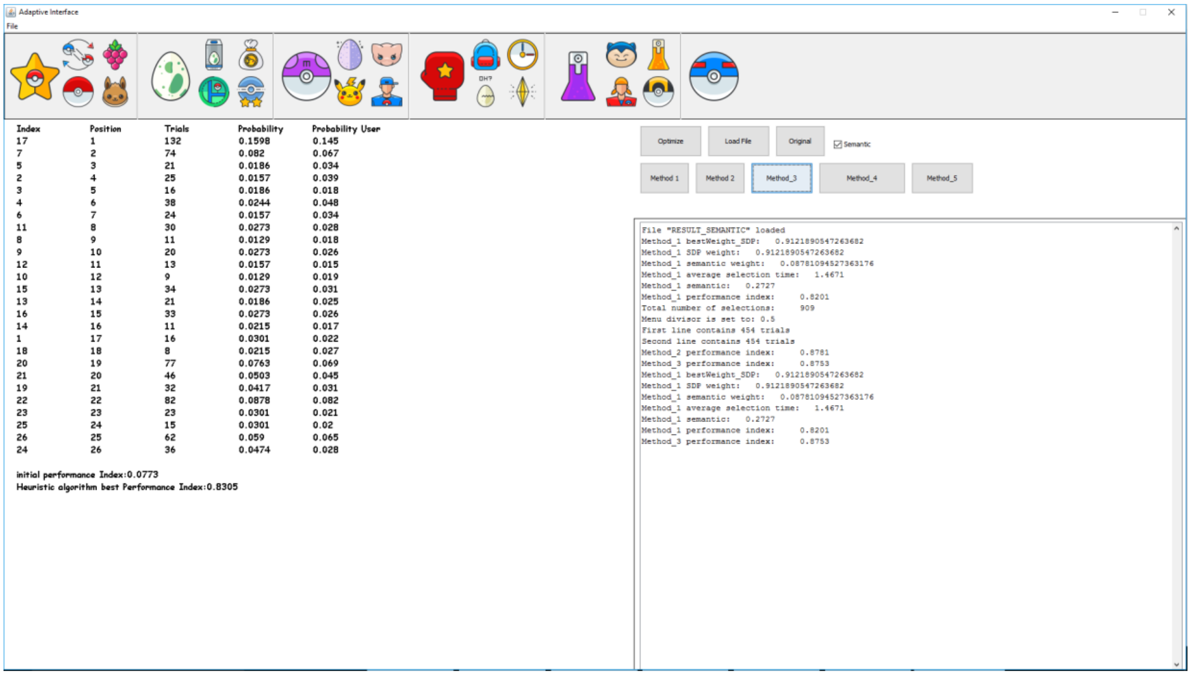
```
File "RESULT_SEMANTIC" loaded
Method_1 bestWeight_SDP:   0.9121890547263682
Method_1 SDP weight:   0.9121890547263682
Method_1 semantic weight:   0.08781094527363176
Method_1 average selection time:   1.4671
Method_1 semantic:   0.2727
Method_1 performance index:     0.8201
Total number of selections:     909
Menu divisor is set to: 0.5
First line contains 454 trials
Second line contains 454 trials
Method_2 performance index:     0.8781
Method_3 performance index:     0.8753
Method_1 bestWeight_SDP:   0.9121890547263682
Method_1 SDP weight:   0.9121890547263682
Method_1 semantic weight:   0.08781094527363176
Method_1 average selection time:   1.4671
Method_1 semantic:   0.2727
Method_1 performance index:     0.8201
```



Index	Position	Trials	Probability	Probability User
17	1	132	0.1598	0.145
7	2	74	0.082	0.067
5	3	21	0.0186	0.034
2	4	25	0.0157	0.039
3	5	16	0.0186	0.018
4	6	38	0.0244	0.048
6	7	24	0.0157	0.034
11	8	30	0.0273	0.028
8	9	11	0.0129	0.018
9	10	20	0.0273	0.026
12	11	13	0.0157	0.015
10	12	9	0.0129	0.019
15	13	34	0.0273	0.031
13	14	21	0.0186	0.025
16	15	33	0.0273	0.026
14	16	11	0.0215	0.017
1	17	16	0.0301	0.022
18	18	8	0.0215	0.027
20	19	77	0.0763	0.069
21	20	46	0.0503	0.045
19	21	32	0.0417	0.031
22	22	82	0.0878	0.082
23	23	23	0.0301	0.021
25	24	15	0.0301	0.02
26	25	62	0.059	0.065
24	26	36	0.0474	0.028

initial performance Index:0.0773
Heuristic algorithm best Performance Index:0.8305

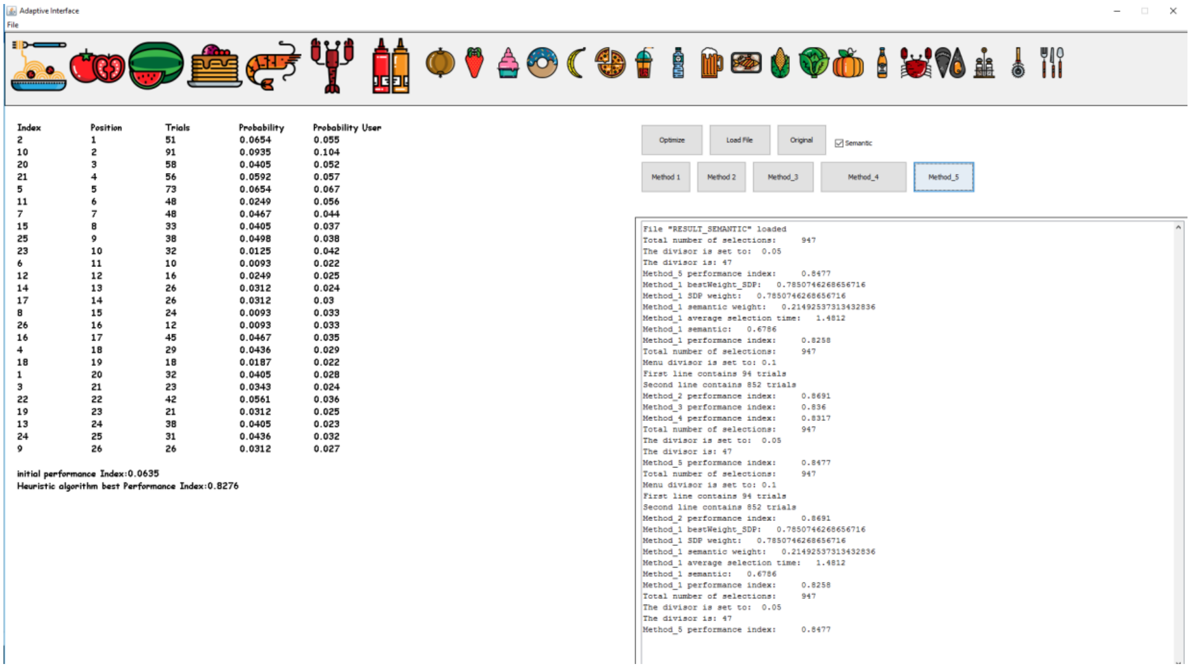
```
File "RESULT_SEMANTIC" loaded
Method_1 bestWeight_SDP:   0.9121890547263682
Method_1 SDP weight:   0.9121890547263682
Method_1 semantic weight:   0.08781094527363176
Method_1 average selection time:   1.4671
Method_1 semantic:   0.2727
Method_1 performance index:     0.8201
Total number of selections:     909
Menu divisor is set to: 0.5
First line contains 454 trials
Second line contains 454 trials
Method_2 performance index:     0.8781
Method_3 performance index:     0.8753
Method_1 bestWeight_SDP:   0.9121890547263682
Method_1 SDP weight:   0.9121890547263682
Method_1 semantic weight:   0.08781094527363176
Method_1 average selection time:   1.4671
Method_1 semantic:   0.2727
Method_1 performance index:     0.8201
Method_3 performance index:     0.8753
Total number of selections:     909
Menu divisor is set to: 0.5
First line contains 454 trials
Second line contains 454 trials
Method_2 performance index:     0.8781
Total number of selections:     909
Menu divisor is set to: 0.5
First line contains 454 trials
Second line contains 454 trials
Method_2 performance index:     0.8781
```



```
Index     Position     Trials     Probability     Probability User
17        1            132        0.1598          0.145
7         2            74         0.082           0.067
5         3            21         0.0186          0.034
2         4            25         0.0157          0.039
3         5            16         0.0186          0.018
4         6            38         0.0244          0.048
6         7            24         0.0157          0.034
11        8            30         0.0273          0.028
8         9            11         0.0129          0.018
9         10           20         0.0273          0.026
12        11           13         0.0157          0.015
10        12           9          0.0129          0.019
15        13           34         0.0273          0.031
13        14           21         0.0186          0.025
16        15           33         0.0273          0.026
14        16           11         0.0215          0.017
1         17           16         0.0301          0.022
18        18           8          0.0215          0.027
20        19           77         0.0763          0.069
21        20           46         0.0503          0.045
19        21           32         0.0417          0.031
22        22           82         0.0878          0.082
23        23           23         0.0301          0.021
25        24           15         0.0301          0.02
26        25           62         0.059           0.065
24        26           36         0.0474          0.028

initial performance Index:0.0773
Heuristic algorithm best Performance Index:0.8305
```

```
File "RESULT_SEMANTIC" loaded
Method_1 bestWeight_SDP:   0.9121890547263682
Method_1 SDP weight:   0.9121890547263682
Method_1 semantic weight:   0.08781094527363176
Method_1 average selection time:   1.4671
Method_1 semantic:   0.2727
Method_1 performance index:     0.8201
Total number of selections:     909
Menu divisor is set to: 0.5
First line contains 454 trials
Second line contains 454 trials
Method_2 performance index:     0.8781
Method_3 performance index:     0.8753
Method_1 bestWeight_SDP:   0.9121890547263682
Method_1 SDP weight:   0.9121890547263682
Method_1 semantic weight:   0.08781094527363176
Method_1 average selection time:   1.4671
Method_1 semantic:   0.2727
Method_1 performance index:     0.8201
Method_3 performance index:     0.8753
```



```
Index     Position     Trials     Probability     Probability User
2         1            51         0.0654          0.055
10        2            91         0.0935          0.104
20        3            58         0.0405          0.052
21        4            56         0.0592          0.057
5         5            73         0.0654          0.067
11        6            48         0.0249          0.056
7         7            48         0.0467          0.044
15        8            33         0.0405          0.037
25        9            38         0.0498          0.038
23        10           32         0.0125          0.042
6         11           10         0.0093          0.022
12        12           16         0.0249          0.025
14        13           26         0.0312          0.024
17        14           26         0.0312          0.03
8         15           24         0.0093          0.033
26        16           12         0.0093          0.033
16        17           45         0.0467          0.035
4         18           29         0.0436          0.029
18        19           18         0.0187          0.022
1         20           32         0.0405          0.028
3         21           23         0.0343          0.024
22        22           42         0.0561          0.036
19        23           21         0.0312          0.025
13        24           38         0.0405          0.023
24        25           31         0.0436          0.032
9         26           26         0.0312          0.027

initial performance Index:0.0635
Heuristic algorithm best Performance Index:0.8276
```

```
File "RESULT_SEMANTIC" loaded
Total number of selections:     947
The divisor is set to:  0.05
The divisor is: 47
Method_5 performance index:     0.8477
Method_1 bestWeight_SDP:   0.7850746268656716
Method_1 SDP weight:   0.7850746268656716
Method_1 semantic weight:   0.21492537313432836
Method_1 average selection time:   1.4812
Method_1 semantic:   0.6786
Method_1 performance index:     0.8258
Total number of selections:     947
Menu divisor is set to: 0.1
First line contains 94 trials
Second line contains 852 trials
Method_2 performance index:     0.8691
Method_3 performance index:     0.836
Method_4 performance index:     0.8317
Total number of selections:     947
The divisor is set to:  0.05
The divisor is: 47
Method_5 performance index:     0.8477
Total number of selections:     947
Menu divisor is set to: 0.1
First line contains 94 trials
Second line contains 852 trials
Method_2 performance index:     0.8691
Method_1 bestWeight_SDP:   0.7850746268656716
Method_1 SDP weight:   0.7850746268656716
Method_1 semantic weight:   0.21492537313432836
Method_1 average selection time:   1.4812
Method_1 semantic:   0.6786
Method_1 performance index:     0.8258
Total number of selections:     947
The divisor is set to:  0.05
The divisor is: 47
Method_5 performance index:     0.8477
```