Optimization of Menu Layouts by Means of Genetic Algorithms

Luigi Troiano[1], Cosimo Birtolo[2], Roberto Armenise[2], and Gennaro Cirillo[2]

[1] RCOST – University of Sannio
Viale Traiano – 82100 Benevento, Italy
troiano@unisannio.it
[2] Poste Italiane s.p.a. – Chief Information Office
Centro Sviluppo Servizi Innovativi Napoli
Piazza Matteotti 3 – 80133 Naples, Italy
{birtoloc,armenis5,ciril127}@posteitaliane.it

Abstract. Menu systems are key components in modern graphical user interfaces (GUIs), either for traditional desktop applications, or for the latest web applications. The design of interface layout must consider different aspects resulting in a trade-off between often conflicting requirements. This trade-off is aimed at making effective use of interfaces in order to meet user preferences and to conform to standard guidelines at the same time. Assuming we are able to quantify such a trade-off, the problem of finding a menu system able to maximize it figures as a combinatorial optimization problem. In this paper we investigate the application of genetic algorithms as a viable approach to identifying solutions that can be used as a starting point for further fine-tuning.

Keywords: GUI design, menu layout, optimization, search based software engineering.

PREPRINT

1 Introduction

At the time of their introduction, menu systems represented a major shift from command-line interfaces. The early systems were very simple and not hierarchical in structure. Since then, they have been innovating continuously. Menu systems have been evolved not only in structure, functionality and purpose. Today, the menu system is a component of fundamental importance for making GUIs attractive and usable, and special care is paid to their design and implementation. This is not only related to traditional desktop applications. In modern web applications, the menu systems still play a key role in helping the user to navigate functionalities, especially after the advent of AJAX and Rich Internet Applications (RIA) more recently. The menu system layout is a basic ingredient for increasing productivity.

In designing a menu layout of good quality, engineers have to consider many aspects including how effectively functionalities are retrieved and activated, what standard guidelines suggest, and what are the preferences of users. These aspects

J. van Hemert and C. Cotta (Eds.): EvoCOP 2008, LNCS 4972, pp. 242–253, 2008.
© Springer-Verlag Berlin Heidelberg 2008

are translated into several design requirements, that often are conflicting. For instance, although having flat hierarchical structures improves accessability, a limitation to the number of items is necessary in order not to have long lists. At the same time, users could have preferences for the item order. A trade-off between these different requirements must be found in order to maximize the menu system quality.

The problem of finding the layout that maximizes quality is combinatorial in nature, as it depends on the arrangement of each item in different positions onto the menu structure, with no construction rules for building an optimal solution. This suggests that the problem is NP-hard. Nowadays, this task is not yet supported by search techniques, and it is left to the experience of engineers. In this paper we investigate the application of genetic algorithms in exploring the space of menu systems at the search of solutions that can maximize the requirements trade-off. The remainder of this paper is organized as follows: in Section 2 we introduce the issues related to menu system design and we formally define the optimization problem; in Section 3 we describe the characteristics of the genetic algorithm used in our experimentation; Section 4 is aimed to present some experimental results; Section 5 outlines conclusions and future work.

2 Designing and Optimizing a Menu System

Although menu systems are components each user has widely experienced, for the sake of clarity, it can be useful to define terms in order to disambiguate definitions. In the remainder of this paper we will refer to *menu layout* as the hierarchical structure by which the user gains access to application functionalities. A menu layout is made of menus; each *menu* is made of a list of *items* referring to *submenus* or to *actions*. The first are menus at lower level, the latter are aimed to activate functionalities, thus they represent the menu system leaves (i.e. terminals).

In designing a menu layout we have to take into the account:

- *accessibility*, as the ease of reaching desired actions.
- *guidelines*, as a set of best practices in organizing the menu layout
- *preferences*, as a wish list made explicit or implicit by the end user

As the menu system aims to quickly activate functionalities, we will consider accessibility as the optimization driver, whilst we will refer to guidelines and preferences as optimization preferences (constraints, when mandatory). Therefore, the problem solution relies on finding a menu layout that maximizes accessibility and compliance to guidelines and user preferences.

Menu selection involves most aspects of human information processing. Indeed, the user is asked for visually inspect the menu, reading and comprehending items in order to find a path that will lead to the desired functionality; choosing the best option until the action is not reached and task accomplished. The menu system layout is expected to better support the user in this task.

So the main issue in designing a good menu system is about how to organize menu items in the hierarchy in order to make actions and to easy accessible.

PREPRINT

In early stages, researchers focused on functional features a menu system is demanded to have in order to improve accessibility. For instance, Walker and Smelcer [1] investigated the relationship between the structure made of walking menus or cascading menus and the time required by an user to reach the target action. In our work, we assumed this issue solved by the current standard implementations, so our focus is mostly on the relationship between the menu hierarchical structure (layout) and accessibility.

Various models for predicting the selection time have been proposed, and research aimed to find the structure that minimizes it. If items are sorted, e.g. alphabetically, search time can be predicted by Hick's Law [2], which states that the time to locate an item is a logarithmic function of the menu size. When menus are not alphabetically ordered, users have to scan them in a linear fashion to locate an item. However, if the user has memorized the position of items in a menu, search time becomes constant. Thus, selection times is reduced to the time needed to reach the item position. Fitts' Law [3,4] predicts the time required to move the cursor to a particular item. It describes the movement time taken to acquire, or point to, a visual target, stating that the movement time needed to acquire a target is a logarithmic function of the ratio between the target distance d and the target width w, known as the task's Index of Difficulty (ID). According to Fitts' law, menu items that appear further down the menu have a greater ID. As this model does not consider constraints in the motion trajectory, Fitts' law cannot accurately predict the movement time in cascading pull-down menus. If the cursor has to be steered along a tunnel, movement time is better modeled by Steering Law [5]. According to this law, movement time is determined by the ratio between the tunnel distance td and the tunnel width tw.

Hollink and van Someren [6] reviewed the assumptions underlying prediction models for the selection time, and proposed a method to validate these assumptions off-line. In their method, after the relationship between the path followed through the menu system and the navigation time, this last is determined by two structural properties of the path: the number of menu items a user has to open and for each navigation step the number of menu items the user has to read. Furthermore the prediction is based on the users' choice strategy, the node opening function and the node choice function.

Recently, Bernard [7] has presented a further model for predicting the selection time. The Hypertext Accessibility Index measure (H_{HAI}) is defined as

$$H_{HAI(x)} = \sqrt{\sum_{i=1}^{L} \sum_{j \in N_i} \log_2(b_j + 1) \log_2(d_j + 1)} \tag{1}$$

where

- x is the menu structure
- L is the maximum number of levels of x
- N_i is the set of menus and menu items at level i
- b_j is the number of children of j
- d_j is the depth of j, assuming for the root and all menu item $d = 1$

PREPRINT

It can be easily verified that $H_{HAI} \in [1, +\infty)$: the lower H_{HAI} is, the more the menu items are accessible; when all menu items are assigned to the root menu (i.e. no submenu is considered) $H_{HAI} = 1$. An interesting characteristic of this model is that H_{HAI} index predicts the expected navigation time on the basis only of the menu system layout. Bernard's model shows that though broader trees in general tend to have better search efficiency than deeper trees, topological shape has also an important effect. The H_{HAI} metric has been validated by comparing predictions with the empirical results found by others and Bernard himself.

Bernard's findings are in accordance with results of other researchers. For instance, Botafogo et al. [8] found that imbalance might indicate a poorly designed hypertext hierarchy, though this is sometimes unavoidable in some domains. They propose two metrics for imbalance, namely the "depth imbalance" and "child imbalance". The depth imbalance metric measures the variance in depth of a node's children; the child imbalance measures the variance in the number of descendants (i.e. sections, subsections, pages, etc.) of a node's children.

In the last years, other techniques have been introduced in order to improve the selection time in cascading pull-down menus, focusing on the selection of first-level items. Shorter selection times have been reached by either decreasing the distance to the menu items, or by increasing the size of the menu item. A split menu adapts to user behavior and relocates the menu items according to usage. Frequently selected items are moved into the top split of the menu and seldom selected items are pushed downward, i.e. the distance to an item depends on selection probability [9]. Ahlström [10] modeled and improved cascading menu selection times through the use of 'force-fields', a variant of sticky widgets, that attracts the cursor towards the cascading menu. The evaluation did not investigate whether the technique caused an adverse effect on selecting non-cascading items.

Designers usually use guidelines to organize the menu structure. They provide a collection of best practices in organizing and structuring the menu layout. Examples are Apple's Human Interface Guidelines [11] and Sun's Java Look and Feel Guidelines [12]. Guidelines are either too specific or too vague, so they do not always apply to the problem at hand [13]. For instance an Apple's Human Interface Guidelines suggests putting on menu bar some particular menus that an user expects to find such as "File", "View" and "Help". Guidelines say, as a general rule, to avoid creating long menus, in fact they are difficult for the user to scan and can be overwhelming, from other side it has not to put many items in a single menu and it needs to regrouping them in other menus. In most guidelines, it is suggested not to go further two levels of cascading menus, although in some cases it is convenient to violate this rule.

According to the current literature, building of menu hierarchy and optimization is a challenging task, whose applications go further the desktop and web applications. However, building a quality menu system requires a large group of users (e.g. focus groups) and a large number of trials in order to find the best way or structuring the menu layout. Search techniques, can provide a valuable

PREPRINT

support in screening alternatives and in providing starting point that can be refined more efficiently.

3 Algorithm

The algorithm is inspired to the Simple GA given by Goldberg [14]. The structure is outlined in Fig.1.

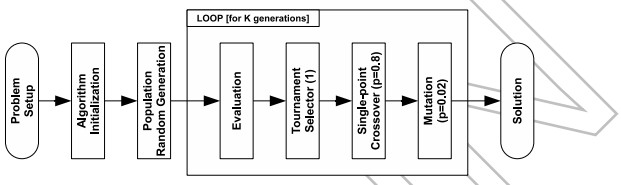

Fig. 1. Algorithm structure

After the problem is setup in terms of menu items and preferences, the algorithm is instanced and the initial population is randomly generated. The algorithm body is made of following stages:

- *evaluation*: a fitness score is assigned to each population individual.
- *genetic processing*: here individuals are genetically processed by selection, crossover and mutation

After K generations the best individual is obtained. Valid (i.e. legal) individuals are compliant with mandatory preferences (constraints), invalid not.

Preferences are given as a set of relational and structural properties, each with an assigned priority. In our case, we assumed priorities on a scale of five: 1- very important, 2- important, 3- medium, 4- not important, 5- not very important. Preferences are facultative. Besides them, we assumed mandatory preferences (i.e. constraints), with priority 0- mandatory.

3.1 Chromosome Structure and Genetic Operators

Among the different ways of representing a tree structure by a chromosome, we choose a coding in which each gene represents the path from root to a menu item as depicted by Fig.2.

The number of genes is not necessarily equal to the number of action (i.e. terminal) items. In some cases, an action could be accessed by different paths. Therefore the chromosome is as long as the sum of allowed occurrences of each action. For example, if an action is allowed twice, there is a need for two genes to that action, each representing a different path. The mapping between genes and actions is kept by an association table. When the path is empty, the action item is associated to the root (e.g. gene N in figure); if the path is null, that action item occurrence is not considered in the menu layout (e.g. gene 1). Such a

PREPRINT

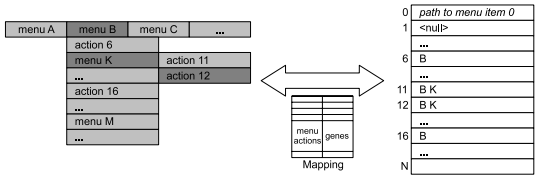

Fig. 2. Chromosome with mapping to the menu layout

chromosome structure is more robust to genetic operations than others, allowing a better control of action items, whose best placement in the menu layout is the ultimate goal of the optimization algorithm. The algorithm is based on three genetic operations:

- selection: a tournament selection has been preferred in order to be less sensitive to the fitness scaling
- crossover: single point crossover
- mutation: gene mutation with a random choice of insertion, deletion and modification of single items.

In particular, the algorithm adopts elitism by random substitution with the best individuals. Tournament is implemented by selecting the best individual after t pairwise comparisons, as described in [14]. Crossover and mutation are described in Fig.3.

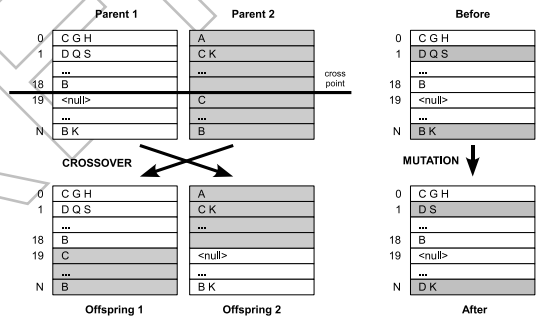

Fig. 3. Crossover and mutation

The menu layout is built by adding paths in the order they occur in the chromosome genes. So, the actual order of items in a certain menu depends on the order they occur in the chromosome, given the same path to them. For instance, if A-B-L precedes A-B-M, L will come first in the menu A-B, otherwise the opposite. A permutation of the mapping entries would allow to obtain different placements. However, the mapping is fixed and not processed by genetic operators. The reason is that the initial path building is purely random. Thus, different placement are considered by the initial production of alternatives. Considering

PREPRINT

the mapping permutation would not be beneficial, adding only an additional degree of freedom to control by the genetic algorithm.

3.2 Fitness Function

The fitness function of an individual x is aimed to model the trade-off between accessability and preference compliance. Thus it is defined as convex combination

$$fitness(x) = \sigma \cdot H(x) + (1 - \sigma) \cdot C(x) \tag{2}$$

where $\sigma \in [0, 1]$, $H(x)$ is the degree of accessability, and $C(x)$ is the degree of constraints' compliance. In particular, $H(x)$ is defined as

$$H(x) = e^{k(1-H_{HAI}(x))} \tag{3}$$

where $H_{HAI}(x)$ is defined by Eq.(1). The constant k controls the exponential decay. Instead the degree of preference compliance is defined as the weighted mean

$$C(x) = \frac{\sum_{i=1}^{m} \bar{p_i} c_i(x)}{\sum_{i=1}^{m} \bar{p_i}} \tag{4}$$

where m is the number of preferences, $\bar{p_i} = 1 - p_i$ is the constraint importance, and $c_i(x)$ is the compliance of x to the preference c_i. Therefore, we assumed a compensation between optimization criteria.

The problem of finding an optimal menu layout consists in placing all action items by maximizing accessability and preference compliance. Preferences can be of different kinds. In our experimentation we considered the following types:

- **Path ordering** (*ancestor* , *successor*): defines a ordering relation between *ancestor* and *successor* along a path
- **Menu ordering** (*predecessor*, *follower*): defines a ordering relation between *predecessor* and *follower* whenever they coexist within the same menu
- **Number of menu items** (*menu*, *min*, *max*): defines the *min* and *max* number of items present in *menu*
- **Occurrence** (*item*, *min*, *max*): defines the *min* and *max* number of occurrences of *item*
- **Level** (*item*, *min*, *max*): defines the *min* and *max* level for *item*
- **Menu belonging** (*item*, *menu*): *item* should belong to *menu*

Each preference has a priority $p_i \in [1, 5]$, where 1 is the highest priority (i.e. very important), 5 the lowest (i.e. not very important). The degree of compliance of x to each preference is computed as

$$c_i(x) = 1 - \frac{v_i(x)}{mv_i(x)} \tag{5}$$

with $v_i(x)$ giving the number of criterion violations of x, and $mv_i(x)$ the maximum number of possible violations.

PREPRINT

Level	Items Min - Max	Priority
Root	4-5	2
Level 1	2-4	3
Level 2	2-5	3
Level 3	1-2	4

Ordering	Priority
A-B-C-D	2
F-G	3

Menu	Level Min - Max	Priority	Repetition Min - Max	Priority
A	1-1	2	1-1	2
B	1-1	2	1-1	2
C	1-1	2	1-1	2
D	1-2	3		
E	1-3	4		
F			0-1	3
G			0-1	3
H	2-2	3	0-1	3
I			0-1	3
J	2-4	3	1-3	2
K			1-2	4
L	3-4	4	1-2	4

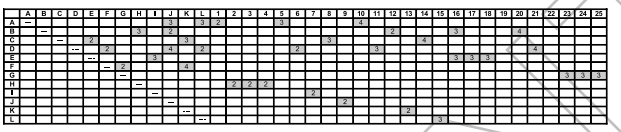

Fig. 4. Preferences. The top-left table provides the number of items at each level, the middle-left table provides two path ordering preferences, the top-right table provides for each menu the desired level and the number of occurrences, the bottom table provides menu belonging preferences specifying the priority of each preference.

4 An Example of Application

As an example of application let us compose a menu layout made of 25 action items $Z1..Z25$ and 12 submenus $A..L$. Layout generation is driven by 60 preferences, outlined in Figure 4. The algorithm was setup with standard parameters: tournaments = 1, crossover probability = 0.8, mutation probability = 0.02, elitism = 2.[1] Figure 5 and Figure 6 show the layout of the best individual respectively after 10 and 1000 generations. Figure 5 depicts a layout with fitness = 0.7949. Indeed, this structure does not satisfy some preferences. In particular,

- the number of children in the root is more than 5
- the tree does not meet any level preference
- menus A and B should not have any repetition

On the other side, A, B and C are in the right order on the menu bar (at level 1) as expected. Instead layout presented in Figure 6 meets better the preference set, thus its fitness value is 0.9717.

When some preferences become mandatory (constraints), the problem becomes harder to solve. In Figure 7 we show a solution layout in this case. In particular, all preferences with priority 2 have been considered mandatory.

The low fitness value (0.566) is due to conflicts between constraints and some other preferences, as it can be easily noticed by observing tables in Figure 4. Obviously, when constraints are in contradictory, there is no solution to the optimization problem. Therefore, it is important to choose the preference and constraint set appropriately, in order to avoid conflicts and contradictions. This is the situation depicted in Figure 8.

[1] Parameters have been chosen by a simple qualitative analysis, according to common values adopted for them, without any in-depth quantitative analysis for their optimization.

PREPRINT

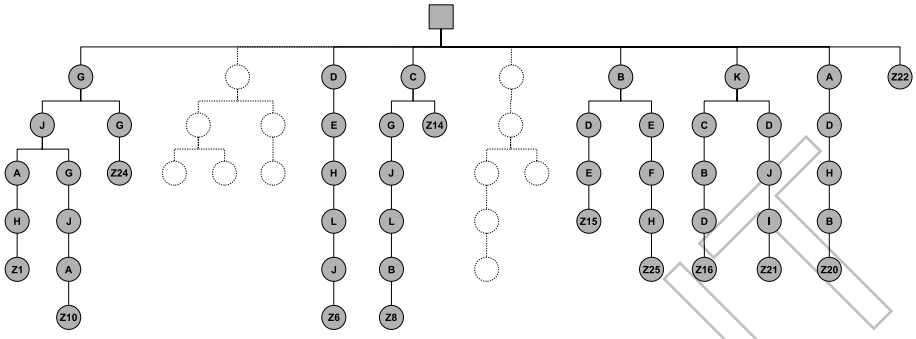

Fig. 5. Layout of the best individual after 10 generations (fitness 0.7949)

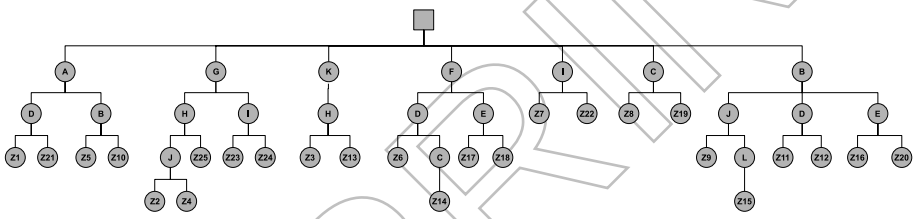

Fig. 6. Layout of the best individual after 1000 generations (fitness 0.9717)

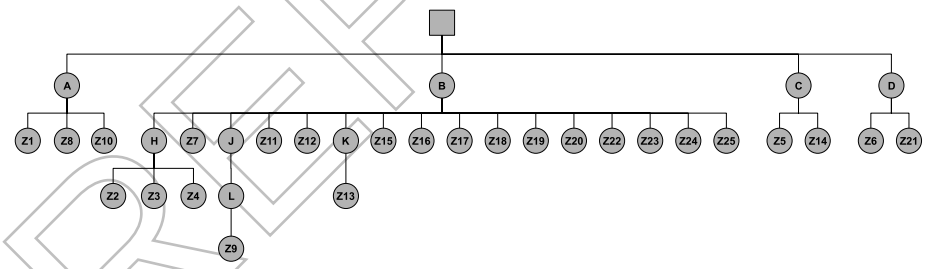

Fig. 7. Layout with constraints (fitness = 0.566)

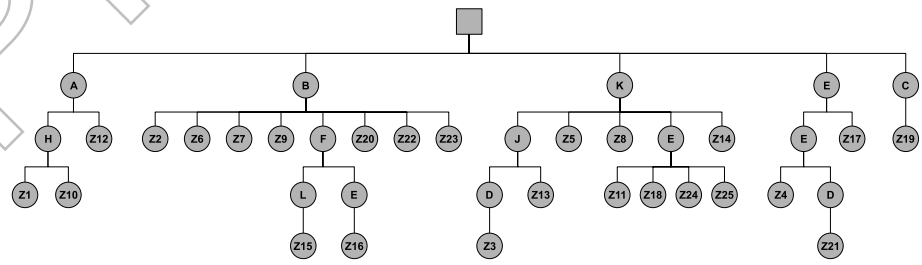

Fig. 8. Layout with compatible constraint and preference sets (fitness = 0.9952)

PREPRINT

In this case we defined a legal individual when the menu bar has 4 or 5 items, and A, B, C are on the menu bar with no repetition. Furthermore we imposed that item $Z1$ has to be an action of menu A. These conditions are expressed by 10 constraints: 3 level constraints, 3 occurrence constraints, 1 number of children constraint, 2 path ordering constraints, 1 belonging constraint. The algorithm run 500 generations on population with 1000 individuals. At the end, legal individuals were 102 (i.e. 898 illegal). Fitness of the best legal individual was 0.9952, with $H = 0.9498$.

We can note that action $Z10$ is in A, action $Z21$ in menu D as expected by the menu belonging preferences. Furthermore, some menu (namely F,G,H,I) are allowed to occur more than once, whilst L no more than twice. We can verify that in layout of Figure 8 these preferences are fully satisfied. In particular, F and H occur once, whilst G and L never. Moreover, the number of children of level 2 are between 2 and 5, and between 1 and 2 at level 3.

These performances are not episodic, as it could be argued. We run the algorithm several times with a different number of preferences in order to study quantitatively the convergence. In Figure 9, we report the median of best fitness with a different cardinality of the preference set and population size (100, 200, 500, 1000 individuals). Preference sets of different cardinality (15, 30, 45, 60 preferences) have been chosen with the same distribution of priorities, so that analysis is independent on this factor.

We can notice that the algorithm reached high values of fitness in all cases, although the behavior differs qualitatively according to the number of preferences considered at a time, thus according to the problem difficulty. So, if in the case of 15 preferences, convergence is reached pretty soon, an increasing time is required

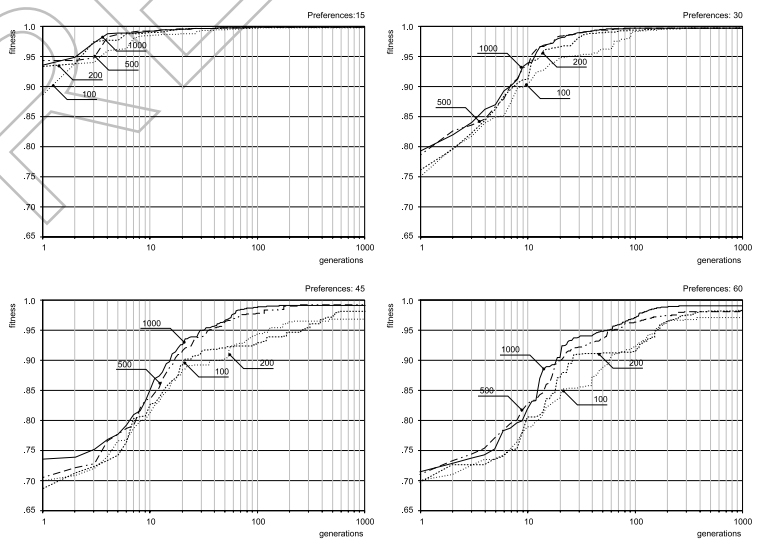

Fig. 9. Algorithm convergence

PREPRINT

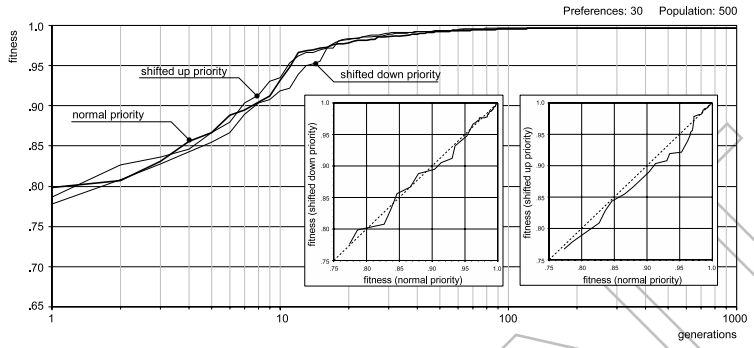

Fig. 10. Algorithm convergence with priority shift

in the case of 30, 45 and 60 preferences. Also population size has an impact on convergence when the number of preferences increases. Indeed, we can observe how the algorithm is not able to converge properly in the case of 60 preferences with 100 and 200 individuals. Another point of interest is the convergence of the algorithm when priority changes. In Figure 10 is outlined the median fitness of the best individual when preference priority is increased (1-3) and decreased (3-5) against the nominal case (2-4).

Obviously the fitness value cannot be the same, as the priority ratios change. However, we can notice how convergence is not heavily affected by a shift of priority, thus resulting robust tho this situation. This means that priority magnitude does not represent a sensible aspect to take into consideration.

5 Conclusions and Future Work

In this paper we presented a genetic algorithm for optimizing the layout of a GUI menu system, keeping into the account accessability, user preferences and standard guidelines. The resulting solution can be used as a robust starting point aimed to be refined by software engineers. Experimentation provided very encouraging results, proving the ability of a simple genetic algorithm in converging towards solutions with high fitness, also in presence of mandatory constraints. Moreover, the algorithm has been proven to scale the number of constraints, and to be robust to constraint preference variations. However, we aim to investigate two main directions in the future. The first is how to integrate mandatory constraints into the solution generation, so there will be no illegal individuals to deal with. Genetic programming seems to be a promising solution to this problem. The second is about how to find out implicit preferences by analyzing the usage of generated menus, instead of forcing the user to make explicit all her/his preferences. In this case, making the genetic algorithm interactive poses additional interesting questions to be answered regarding to how to sample the search space and to gather user feedback.

PREPRINT

Acknowledgments. This work was partially supported by MIUR Project Automatic System For The Visually Impaired (SAPI), n.1642-2006.

References

1. Walker, N., Smelcer, J.B.: A comparison of selection times from walking and pull-down menus. In: Proceedings of ACM CHI 1990 Conference on Human Factors in Computing Systems, pp. 221–225 (1990)
2. Hick, W.E.: On the rate of gain of information. Quarterly Journal of Experimental Psychology 4, 11–26 (1952)
3. Fitts, P.M.: The information capacity of the human motor system in controlling the amplitude of movement. J. Exp. Psychol. 47, 381–391 (1954)
4. Cockburn, A., Gutwin, C., Greenberg, S.: A predictive model of menu performance. In: CHI 2007: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 627–636. ACM, New York (2007)
5. Accot, J., Zhai, S.: Beyond fitts' law: models for trajectory-based hci tasks. In: CHI 1997: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 295–302. ACM, New York (1997)
6. Hollink, V., van Someren, M.: Validating navigation time prediction models for menu optimization. In: Althoff, K.D., Schaaf, M. (eds.) LWA. Hildesheimer Informatik-Berichte, University of Hildesheim, Institute of Computer Science, vol. 1, pp. 47–52 (2006)
7. Bernard, M.L.: Examining a metric for predicting the accessibility of information within hypertext structures. PhD thesis, Wichita, KS, USA (2002) Adviser-Charles G. Halcomb
8. Botafogo, R.A., Rivlin, E., Shneiderman, B.: Structural analysis of hypertexts: identifying hierarchies and useful metrics. ACM Trans. Inf. Syst. 10(2), 142–180 (1992)
9. Ahlström, D.: Modeling and improving selection in cascading pull-down menus using fitts' law, the steering law and force fields. In: CHI 2005: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 61–70. ACM, New York (2005)
10. Ahlström, D., Alexandrowicz, R., Hitz, M.: Improving menu interaction: a comparison of standard, force enhanced and jumping menus. In: CHI 2006: Proceedings of the SIGCHI conference on Human Factors in computing systems, pp. 1067–1076. ACM, New York (2006)
11. Apple Computer Inc.: Apple human interface guidelines. Technical report, Apple Computer Inc. (2006)
12. Inc, S.M.: Java look and feel design guidelines: advanced topics. Addison-Wesley Longman Publishing Co., Inc., Boston (2001)
13. Oliver, A., Regragui, O., Monmarché, N., Venturini, G.: Genetic and interactive optimization of web sites. In: The 11th International World Wide Web Conference, Honolulu, Hawaii, USA, pp. 7–11 (2002)
14. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Reading (1989)

PREPRINT